Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

РОЗРОБКА ВЕБ_ЗАСТОСУНКУ З ВИКОРИСТАННЯМ БІБЛІОТЕКИ REACT

Текстова частина до курсової роботи

за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи

с.в. _Борозенний С.О.__

(прізвище та ініціали)

_____

(підпис)

“____” _____ 2023 р.

Виконала студентка _____

___Желізняк А.О._____

(прізвище та ініціали)

“____” _____ 2023 р.

Київ 2023

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри мультимедійних систем,

доцент, к.ф-м.н.

_____О. П. Жежерун (підпис)

„____”_____2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студентці Желізняк Анні Олександрівні факультету інформатики 3-го курсу

ТЕМА Розробка веб-застосунку з використанням бібліотеки React

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Вступ

1 Загальні відомості та роль React у сучасній веб-розробці

2 Особливості технології React

3 Розробка застосунку

Висновки

Список літератури

Додатки (за необхідністю)

Дата видачі „___” _____ 2023 р. Борозенний О.С. _____ (підпис)

Завдання отримав _____ (підпис)

Тема: Розробка веб-застосунку з використанням бібліотеки React

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	25.07.2022	
2.	Аналіз матеріалів за темою	11.09.2022	
3.	Розробка та програмування алгоритму	23.02.2023	
4.	Написання текстової частини до курсової роботи	28.04.2023	
5.	Коригування виконаної роботи	05.05.2023	
6.	Створення слайдів для доповіді та написання доповіді.	08.05.2023	
7.	Остаточне оформлення роботи та слайдів	13.05.2023	
8.	Захист курсової роботи	24.05.2023	

Желізняк А. О. _____

Борозенний О. С. _____

“_____” _____

Зміст

Анотація

Курсова робота присвячена дослідженню особливостей роботи бібліотеки React для побудови веб-застосунків на прикладі застосунку діагностичного центру. Під час розробки було використано такі технології : React, Node.js, Express, MongoDB, Mongoose, React Router, React Responsive Carousel, three.js, react-three-fiber, @react-three/drei.

Ключові слова : React, веб-застосунок, Mongoose, бібліотеки front end.

Вступ

Сучасну веб-розробку складно уявити без використання front end фреймворків та бібліотек. React є одним з найбільш вживаних інструментів для розробки високоефективних та багатофункціональних застосунків. В цій роботі буде досліджено принципи створення веб-застосунку із використанням React та буде розглянуто головні особливості цієї технології.

Основна частина складається з трьох розділів.

У першому розділі загальний огляд бібліотеки, порівняння з іншими популярними технологіями, визначення місця React в сучасній веб-розробці і розгляд його майбутніх перспектив.

Другий розділ присвячено особливостям React, в ньому буде розглянуто новий підхід бібліотеки з декларативним стилем визначення елементів сторінки, рендерингом лише тих компонентів, в які вносились зміни, особливості компонентної побудови сторінки та способи побудови компонентів. Також буде розглянуто особливості життєвого циклу та нові функції, які з’явились в React нещодавно. В цьому розділі будуть зазначені популярні бібліотеки React та особливості їхнього підключення і використання.

У третьому розділі буде розглянуто особливсті розробки застосунку та буде показано, як імплементувати фішки React в проекті.

Основна частина

Розділ 1 : Загальні відомості та роль React у сучасній веб-розробці

1.1 Огляд бібліотеки React

React - це інструмент для веб-розробки, який був створений компанією Meta (Facebook) 2013 року. Головною особливістю створення застосунку з використанням React є розбиття сторінки на окремі компоненти [2] - складові, які дозволяють більш ефективно будувати архітектуру застосунку і надають можливість повторного використання коду. Завдяки віртуальному DOM, React має покращені функції рендерингу сторінок. На відміну від класичних підходів для розробки застосунку з використанням “чистого” JavaScript, де при будь-якій маніпуляції зі складовими інтерфейсу необхідно відрендерити всю сторінку, у застосунку з використанням React під час взаємодії з компонентом буде відбуватись рендеринг лише для нього, а не для всієї сторінки. Ці властивості React дозволяють ефективно будувати інтерфейс для застосунків з багатьма компонентами та великим обсягом даних, які постійно оновлюються [3].

React є бібліотекою – набором коду, який використовується для виконання певних завдань. Саме тому React надає розробникам більше гнучкості та контролю над загальною архітектурою застосунку, і це є його перевагою над іншими популярними інструментами (Angular, Vue тощо)[1], які є фреймворками і вимагають певної архітектури та виконання специфічних шаблонів.

Від часу створення і донині бібліотека розвивається та доповнюється новим функціоналом. 2019 року було додано нові можливості для передачі даних між компонентами (Hooks) та можливості для асинхронної роботи (Suspence), команда продовжує роботу над вдосконаленням бібліотеки.

1.2 Місце React в сучасній веб-розробці

React є популярним інструментом веб-розробки, який використовується великими компаніями для реалізації застосунків. Згідно зі статистикою кількість використань React зростає [4].

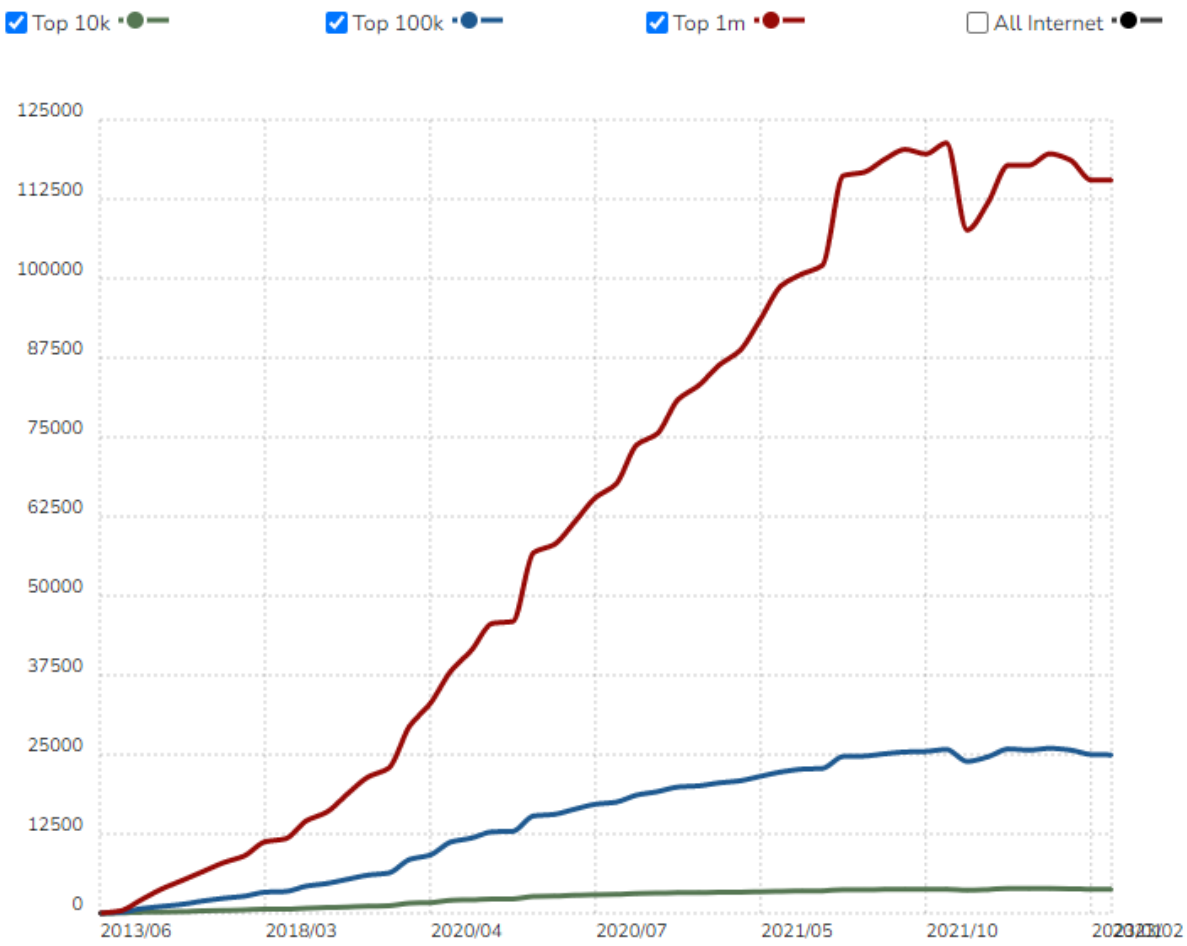

Рис. 1 Статистика використання React [4]

React є лідером серед front end фреймворків та біблотек і частка застосунків з використанням цієї технології сягає 80% [5], що значно випереджує інші технології.

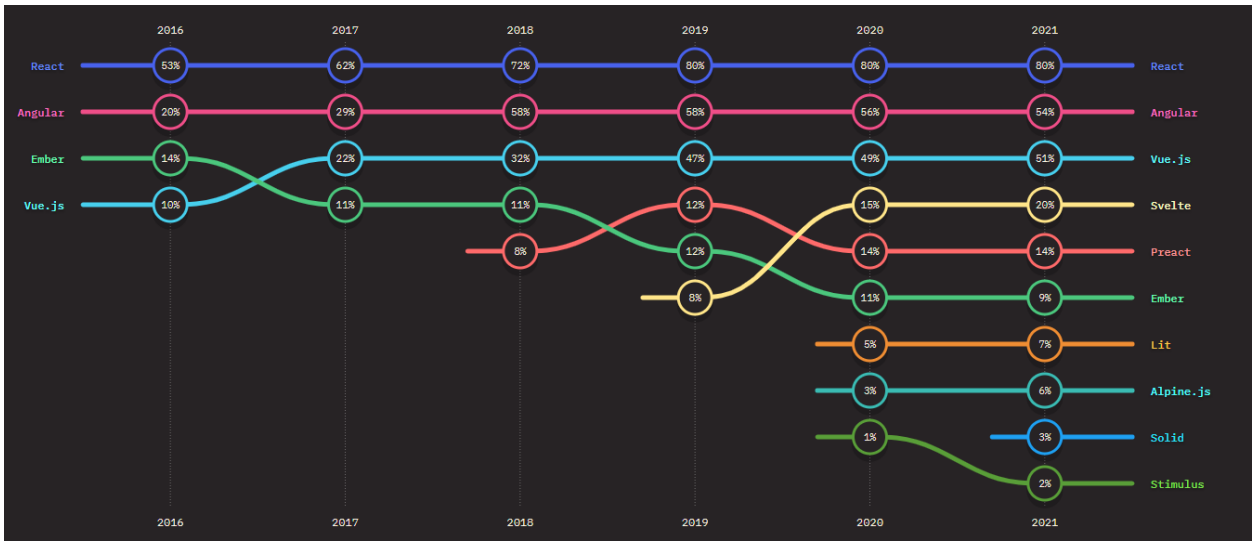

Рис.2 Порівняльна статистика використання front end технологій [5]

Завдяки високій ефективності, легкості використання та можливостям до масштабування React використовувався для розробки багатьох відомих онлайн платформ [6], таких як Facebook, Instagram, Netflix, PayPal, Reddit, WhatsApp, Dropbox та інші. Випуск React Native значно збільшив кількість користувачів React, оскільки адаптував підходи основної бібліотеки для розробки мобільних інтерфейсів. Можна прийти до висновку, що технологія активно розвивається та розширюється, тож очікується, що її позиція лідера зберігатиметься й надалі.

Розділ 2 : Особливості технології React

2.1 Віртуальний DOM

Віртуальний DOM – це концепція, яка дозволяє покращити ефективність оновлення DOM дерева. Хоча цей патерн нині використовується в багатьох технологіях, він став широковідомим саме завдяки React. Віртуальний DOM перетворює стиль взаємодії з деревом об'єктів в декларативний : програма визначає стан, і DOM робить все необхідне, щоб відповідати йому. VDOM буде створено після виклику методу render(), і він матиме вигляд JavaScript об'єкту певної структури.

Головне призначення VDOM полягає в більш ефективному перемальовуванні сторінки, він надає алгоритм, який дозволяє мінімізувати кількість операцій взаємодії з реальним DOM. Операція рендерингу сторінки у застосунках, які мають великий обсяг даних, є досить дорогою, тому виникає багато складнощів, при використанні стандартного методу взаємодії з DOM, де при зміні об'єкту автоматично відбуваються зміни і в усіх його нащадків. Натомість при використані VDOM процес перемальовування сторінки поділяється на три основні етапи : створення VDOM, обчислення різниці, оновлення інтерфейсу.

- Створення VDOM - під час рендерингу елементу створюється структура його компонентів та взаємозв'язків, яка зберігається в браузері та не впливає на відображення, оскільки є лише легшою копією DOM.
- Обчислення різниці – під час наступного оновлення даних буде створена інша копія DOM, яка буде відображати всі внесені до інтерфейсу зміни. Ця копія буде порівнюватись з попередньою збереженою за алгоритмом визначення різниці (diff) [7]. За цим алгоритмом, якщо кореневий вузол змінює свій тип, дерево буде перебудовано повністю, інакше в зміни будуть вноситись лише властивості, які оновили свої значення. Для більш ефективної ітерації між нащадками вузла рекомендується додавати до них атрибут ключ (key). В такому разі під час зіставлення двох дерев буде

зрозуміло, які саме нащадки зазнали змін і не буде необхідності замінювати їх всіх.

- Оновлення інтерфейсу – ReactDOM на основі зібраної інформації про різницю вносить зміни до реального DOM, оновлюючи інформацію лише про ті вузли, які увійшли до різниці.

Завдяки цим крокам кількість операцій зміни реального DOM буде мінімальною і як результат робота сторінки пришвидшиться та використовуватиме менше ресурсів.

2.2 JSX

JSX – це розширення JavaScript для React, яке дозволяє визначати вигляд сторінки та є дещо схожим до HTML. В межах JSX окрім стандартних тегів та компонентів можна використовувати змінні та навіть функції (в такому разі ці значення необхідно розміщувати у фігурних дужках). Браузер буде перетворювати JSX в JavaScript код в процесі транспайлингу, за замовчуванням для цього використовується Babel. Код, отриманий в результаті транспайлингу виконуватиме ті ж самі функції, що й JSX, і такий код також можна писати в програмі. Використання JSX не є обов’язковим, проте надає функції render більш компактного та легшого для розуміння структури сторінки вигляду.

JSX дає змогу декларативно визначити складну структуру сторінки, проте варто пам’ятати, що метод render() може повертати лише один вузол, тож структура має бути загорнута у певний загальний тег, який і буде кореневим вузлом дерева, яке повертатиметься. Також при використанні JSX необхідно дотримуватись кількох правил. По-перше, для найменувань краще використовувати камел кейс, і саме в такий формат JSX буде перетворювати назви, написані через дефіс, які запозичуються з інших файлів. По-друге, всі теги мають бути закритими, це обумовлено тим, що найближчим відповідником JSX є XHTML, де це є обов’язковим правилом. По-третє, всі створені компоненти рекомендується називати з великої літери для легшої читабельності коду.

Окрім декларативного стилю, легкості розуміння та написання, серед переваг JSX можна виділити перевірку помилок на етапі компіляції (переважно синтаксичних), та додаткові повідомлення та методи для дебагу, що значно спрощує розуміння помилок на етапі написання коду. Також JSX підвищує безпеку клієнта [8], оскільки запобігає XSS атакам. Принцип XSS атаки полягає у вставленні зловмисного коду, що має на меті захопити дані. Проте React перед рендерингом сторінки уникає всіх скриптів. JSX ігнорує всі потенційно небезпечні символи (наприклад, ‘<’) і перетворює ці значення в звичайну стрічку. Якщо необхідно обробити певний HTML, який прийшов на вхід, існує метод dangerouslySetInnerHTML, який буде здійснювати подібну перевірку.

2.3 Компоненти

Компоненти є основними складовими інтерфейсу, побудованого за допомогою React. Вони утворюють дерево компонентів, яке потім перетворюється браузером в DOM дерево. Завдяки компонентам одну структуру можна використовувати в різних місцях програми з іншими наборами даних та реакціями на дії користувача.

Компоненти можна визначати за допомогою функцій, або створюючи клас, який унаслідується від React.Component або React.PureComponent. Різниця між Component та PureComponent полягає в реакції на оновлення даних : для Component при будь-якій зміні даних автоматично буде викликатись оновлення для компоненту, а для PureComponent використовується метод shouldComponentUpdate(), який проводить неглибоке порівняння props та state і оновлює компонент, коли знаходить різницю. PureComponent буде працювати ефективніше через неглибоке порівняння, проте через нього ж можуть пропускатись зміни в складних структурах (наприклад, в масивах), а Component буде здійснювати повне порівняння, що займатиме більше часу, проте відслідковуватиме всі зміни. Переважно компоненти створюють за допомогою функцій.

2.3.1 Передача та зберігання даних

Під час створення компонентам можна передавати певні дані, і потім використовувати їх для налаштування компонента. Ці параметри називаються пропсами (props), і для передачі необхідно вказати назву параметру та його значення як атрибути компонента. Якщо компонент створюється як функція, то props перелічуються як параметри функції. Якщо компонент створюється як клас, то props необхідно передати як параметри конструктора. Параметри не можна змінювати в компоненті, можна лише використовувати їхні значення.

Якщо необхідно зберігати дані, які можна модифікувати, використовується state. Дані зі state доступні лише всередині компоненту, батьківський компонент не має до них доступу. Після зміни state компонент автоматично має викликати метод render та інтерфейс має оновитись. Визначати поля state необхідно в конструкторі, це може бути довільна кількість параметрів записаних у вигляді “ключ : значення”. Для використання значень зі state необхідно викликати this.state, а для зміни значень this.setState.

Властивості, описані вище, працюють при стандартній передачі даних від батьківського компонента до нащадка. Проте для операцій, пов’язаних з фокусом, роботою з бібліотеками чи певними видами анімацій необхідно змінювати компонент поза межами цього стандартного зв’язку. Для цього використовують посилання (refs). Оскільки посилання пов’язані з екземплярами, їх можна створювати лише для компонентів-класів. Якщо є необхідність створити посилання на компонент визначений функцією, єдиний вихід – це переписати його у вигляді класу. Посилання, як і попередні властивості, визначаються у конструкторі за допомогою React.createRef(), а для присвоєння посилання об’єкту використовуємо ref атрибут. Для того, щоб доступитись до значень, збережених в посиланні застосовують властивість current. За замовчуванням current зберігає базовий об’єкт DOM, проте після присвоєння ref значення змінюється на конкретний екземпляр. Посилання рекомендується

використовувати лише в ряді причин, згаданих вище, а в інших випадках краще замінити їх іншими властивостями React.

2.3.2 Життєвий цикл компонента

Від моменту створення і до моменту зникнення з екрану кожен компонент проходить через 3 основні етапи життєвого циклу. Для кожного етапу існує набір функцій, за допомогою яких можна дізнаватись про стан об’єкту та вносити зміни. Саме завдяки особливостям життєвого циклу можна ефективно завантажувати дані зі сторонніх ресурсів, керувати поведінкою оновлення та багато іншого.

Давайте розглянемо етапи життєвого циклу та функції, які їм відповідають.

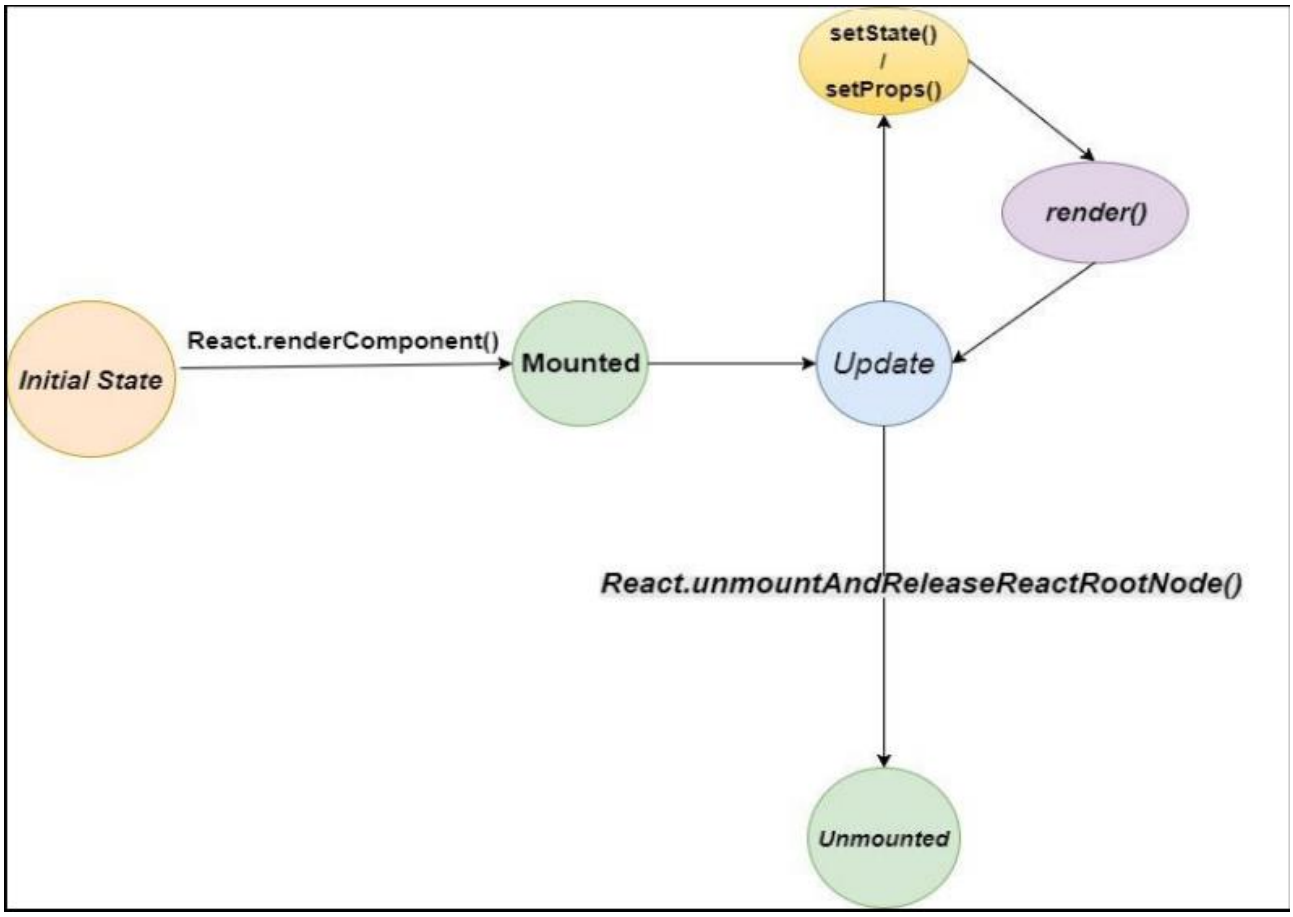

Рис.3 Життєвий цикл компонента [9]

Першою стадією життєвого циклу є монтування (mounting), де відбувається ініціалізація, створення та вставка компонента в DOM. На цьому етапі ми можемо послуговуватись такими методами :

- Конструктор – як згадувалось вище, конструктор використовується для встановлення значень параметрів, посилань та стану. Він викликається перед тим, як компонент буде змонтовано і є необов’язковим етапом (якщо ми не встановлюємо жодного значення не має необхідності створювати його).
- getDerivedStateFromProps – ця функція використовується і для етапу монтування, і для етапу оновлення компонента. Вона буде викликатись до методу render і є необхідною, якщо стан залежить від значень параметрів компонента [10]. Функція буде повертати об’єкт або стан, який слід змінити. Оскільки функція має дуже обмежене застосування, вона не є обов’язковою.
- Рендеринг (render) – на відміну від попередніх функцій, render є обов’язковою для всіх компонентів. Для компонентів, визначених як класи це буде функція render(), для компонентів, визначених як функції, ці дії будуть відбуватись з кодом, розміщеним в return. Функція render обов’язково має бути чистою [10], тобто не має на пряму взаємодіяти з браузером чи сторонніми ресурсами (для цього необхідно використати інші функції життєвого циклу), має завжди повертати значення одного типу і не має змінювати значення зі стану. Зазвичай функція повертає елемент, створений за допомогою JSX, проте також вона може повертати інші типи (числа, булеві значення, масиви, стрічки, портали та фрагменти).
- componentDidMount – цей метод викликається одразу після вставки компонента до DOM і зазвичай використовується для завантаження даних зі сторонніх ресурсів або створення підписок.

Після створення компоненту настає стадія оновлення, яка триває до моменту зникнення компоненту зі сторінки. В ній використовуються дві функції, про які вже було згадано, : getDerivedStateFromProps та render. Проте також є кілька особливих функцій : shouldComponentUpdate, getSnapshotBeforeUpdate, componentDidUpdate. Оновлення буде відбуватись тоді, коли метод

shouldComponentUpdate поверне true. В цьому методі можна перевизначити базову поведінку порівняння стану та параметрів, за замовчуванням цей метод буде повертати true, коли буде знайдено будь-яку зміну в параметрах чи стані.

getSnapshotBeforeUpdate викликається після внесення змін в DOM і дозволяє дізнатись інформацію про вузли перед оновленням. Цей метод має вузьке коло застосування, найпоширенішим прикладом є відслідковування позиції під час скролу сторінки. componentDidUpdate є ще одним додатковим необов'язковим методом цього етапу, який дозволяє вносити зміни одразу після оновлення компоненту.

Останнім етап життєвого циклу є знищення компоненту (unmounting). Для цього етапу існує лише один необов'язковий метод : componentWillUnmount. В ньому необхідно скасувати всі підписки, запити, таймери тощо, які використовувались в компоненті перед тим, як він зникне з екрану. Якщо нічого з переліченого не використовувалось цьому методу можна залишити поведінку за замовчуванням.

2.4 Використання хуків

Хуки (hook) стали доступними в React віднедавна – з версії 16.8. Але саме з того часу принципи створення застосунку з використаннямReact змінились. Як було згадано в попередньому розділі, нині більшість розробників надають перевагу компонентам, які визначені саме за допомогою функцій, і це стало можливим після створення хуків.

Почнемо з розгляду причини створення хуків та їхніх переваг. Довгий час класи були єдиним способом для створення компонентів, і хуки з'явились як альтернатива, яка розв'язує всі головні недоліки, які були в класах. По-перше, класи дозволяють повторно використовувати лише повну структуру, проте досить часто необхідно використати лише логіку передачі даних. Це питання намагались розв'язати різними шляхами, проте лише хуки дозволили зробити

це без значних змін загальної структури проекту. По-друге, все керування даними в класах розміщується у різних відповідних функціях життєвого циклу, і якщо ми працюємо з багатьма станами водночас може вийти досить масивний і нечитабельний код, який не можна розбити на менші компоненти. Хуки розв'язують й цю проблему, на відміну від класу, функцію цілком можна розбити на менші функції за логікою виконання, що полегшує процес написання і розуміння коду. По-третє, команда React планує впроваджувати AOT-компіляцію компонентів для проекту, і дослідження показали, що класові компоненти можуть бути неефективними при такому підході, тож використання хуків є чудовим рішенням.

Хуки є еквівалентним замінником класовим компонентам, вони можуть послуговуватись основними принципами та методами React. Їхні основні переваги проявляються саме в легкості написання і розуміння та інкапсуляції логіки, проте з точки зору ефективності вони будуть ідентичні класам.

Розглянемо найчастіше вживані хуки [11].

- Робота зі станом – для додавання стану до компоненту, визначеного функцією, використовується хук useState. Він складається зі змінної та функції, яка буде оновлювати значення змінної. Необхідно вказати лише початкове значення змінної, всі подальші маніпуляції будуть виконуватись за допомогою її функції. Виконання функції set буде асинхронним, тож у функції, де встановлюється нове значення, не варто використовувати змінну, визначену за допомогою хуку, оскільки вона ще може зберігати значення до оновлення. Після виклику set буде відпрацьовувати оновлення, якщо значення відрізнятиметься від попереднього. Як можна побачити логіка досить схожа до класових компонентів, але при застосування хуків також можливе групування [11] кількох змін для одного виклику render, що підвищує ефективність. Альтернативним варіантом є використання хуку useReduce [11]. Як і useState він має параметри змінну та функцію, але цей хук повертає стан

пов’язаний з функцією dispatch. useReduce краще працює з великим набором даних, або якщо стан залежить від попередніх значень.

- Робота з даними – для роботи з даними, підписками використовується хук useEffect. Він виконуватиме ті ж функції, що й методи componentDidMount, componentDidUpdate, componentWillUnmount в класових компонентах, тож і цілі його використання є схожими. Цей хук буде використовуватись після кожного рендерингу компонента, проте цю поведінку можна змінити. Для цього необхідно передати до хуку ще один компонент – масив зі значенням, після оновлення яких необхідно використати useEffect. Якщо цей масив передавати без жодного значення, useEffect використається лише двічі – при створенні та знищенні компоненту. Цікавим є те, що на відміну від класових компонентів, де для створення та знищення існують різні методи, у функціональних компонентах всю роботу виконує єдиний хук useEffect. В ньому необхідно одночасно створювати підписки і скасовувати їх в return. Таким чином під час кожного оновлення компонента буде скасовуватись стара підписка та створюватиметься нова, або ж, якщо useEffect викликається лише двічі, буде ідентична поведінка, як і в класових компонентах.

  Як і useState, useEffect буде виконуваись асинхронно, тож зміни не одразу можуть бути помітними, і на це необхідно зважати під час розробки. Існує також інструмент з подібним функціоналом, який виконуватиме обробку синхронно перед відмальовуванням інтерфейсу, і це хук useLayoutEffect.
- Робота з контекстом – для роботи з контекстом можна використати хук useContext. Перед оновленням буде викликано значення, яке було вкзане в контексті.
- Робота з посиланнями – для цього можна використати хук useRef. Як і в класових компонентах, під час використання функцій нам можуть знадобитись посилання на конкретні об’єкти для встановлення фокусу,

роботи зі сторонніми бібліотеками тощо. Робота useRef також є аналогічною до роботи ref, описаною в попередньому розділі.

- Робота з callback – хук useCallback має параметри функцію та масив параметрів і повертає виклик цієї функції від заданих параметрів. Хук викликатиметься кожного разу, коли будуть відбуватись зміни з параметрами. Це допомагає інкапсулювати логіку та прив’язати роботу функцій до певних змінних.
- Робота з кешуванням – для часто вживаних складних обчислень або операцій, які використовують багато ресурсів можна зберігати результат виконання використовуючи хук useMemo. Результат буде оновлюватись лише за умови зміни хоча б одного параметра функції.

Використання хуків є зручним та простим, проте існує два головні обмеження щодо роботи з ними. По-перше, хуки можна викликати лише з компонентів, визначених за допомогою функцій. По-друге, хуки можна викликати лише на найвищому рівні. Окрім стандартних хуків можна налаштовувати власні хуки, які будуть виконувати логіку, яка часто використовується в проекті.

2.5 Робота з асинхронністю

Хуки – не єдина новація, яка з’явилась у React. Також додались можливості, які допомагають контролювати компоненти під час асинхронних функцій. Найпоширеніший приклад їхнього використання – встановлення певного компоненту-лоадеру, поки дані не завантажаться. Для цієї мети було створено Concurrent Mode та Suspense.

Для використання Suspence необхідно два компоненти : головний компонент з даними, який має відображатись, і компонент який буде відображатись до закінчення завантаження даних (fallback). Додавання цього інструменту стало в нагоді розробникам, оскільки раніше для схожого функціоналу необхідно було будувати складну систему компонентів, а з

Suspence це є автоматизованим. При використанні Suspense разом з ‘лінивим’ завантаженням (lazy) отримується значна перевага через ефективніше використання ресурсів. Ліниве завантаження полягає в тому, що будуть завантажуватись лише необхідні елементи, а весь інший JavaScript код буде довантажуватись лише за необхідності.

На прикладі застосунку можна показати, як буде працювати Suspense. До закінчення завантаження даних з БД на екрані буде відображатись спінер.

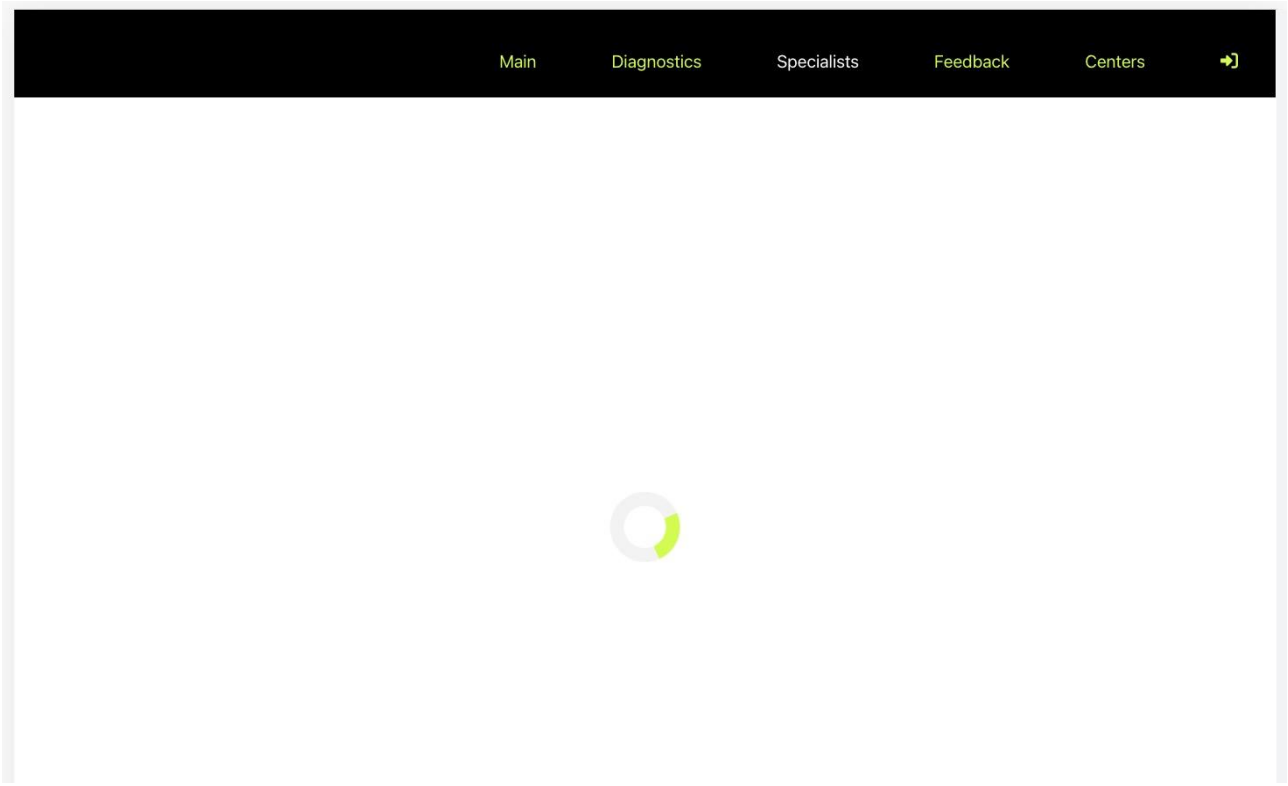

Рис.4 Відображення fallback при використанні Suspense

Після завантаження даних вони відобразяться, а спінер зникне.

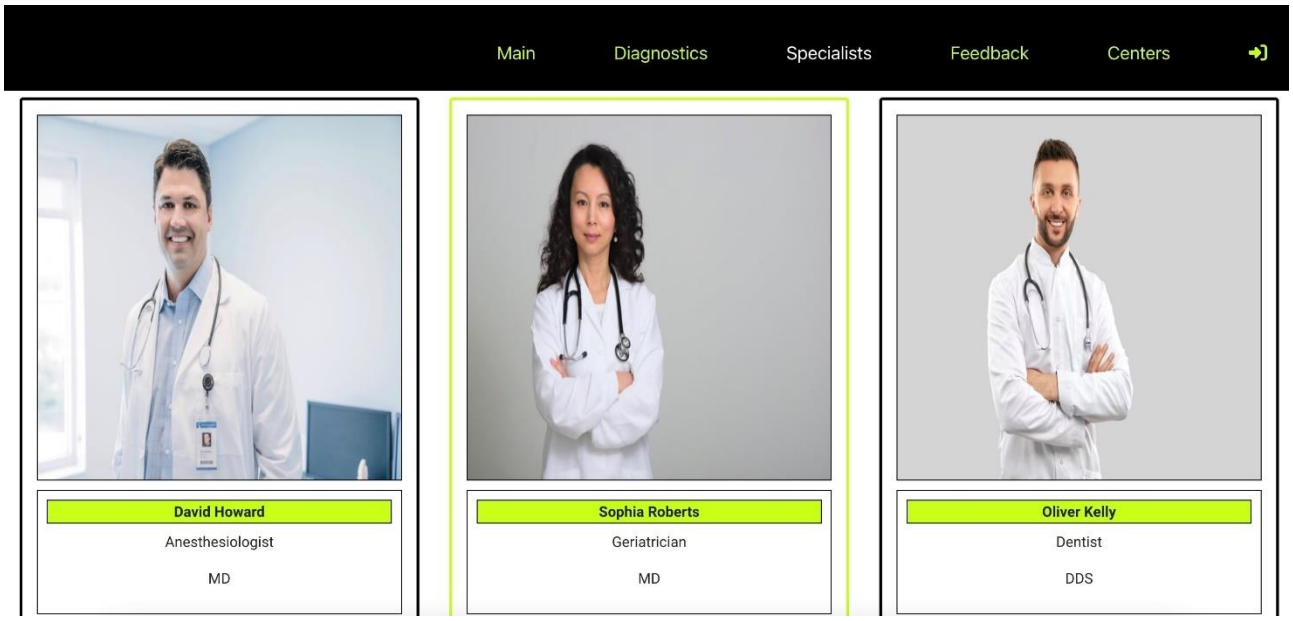

Рис.5 Відображення даних після завантаження при використанні Suspense

React продовжує розвиватись і зараз з’явилась опція нового підходу [12] для роботи з асинхронністю - Concurrent Mode. Зараз ця функція доступна лише для експериментального режиму, але вона має великі перспективи і з часом стане частиною бібліотеки. На відміну від класичного рендерингу сторінки, який зупиняє свою роботу лише після завершення, з використанням Concurrent Mode рендеринг може призупинитись, обробити певні події, і після цього продовжити роботу. Це дозволяє виконувати певні дії не блокуючи інтерфейс та виконувати спершу більш пріоритетні завдання.

2.6 Бібліотеки та інструменти

Під час розробки застосунку з використанням React застосовуються бібліотеки React та сторонні бібліотеки для React, і цей розділ містить короткий огляд на бібліотеки, які використовувались для написання застосунку з практичної частини.

2.6.1 React Router

React Router – це бібліотека React, яка відповідає за налаштування навігації для застосунку. Через свій функціонал бібліотека є найчастіше вживаною бібліотекою React. Для SPA навігація між підсторінками полягає у визначенні

окремих шляхів для частин сторінки, запити до яких будуть викликатись користувачем у відповідь на певні дії. Таким чином кожен фрагмент сторінки має власний шлях [13], за допомогою якого можна доступитись до фрагменту за відповідним запитом у браузері. Для визначення шляхів в App необхідно додати елемент Router, в якому в Routes визначити кожен шлях. Для шляху також необхідно зазначити, якому компоненту він має відповідати. Шляхи можна робити захищеними (PrivateRoute) для того, щоб певні сторінки могли відображатись лише в разі успішної авторизації. Бібліотека пропонує на вибір декілька типів маршрутизаторів : StaticRouter, NativeRouter, MemoryRouter, HashRouter, BrowserRouter (використовується найчастіше). Розглянемо різницю [14] між цими видами.

- MemoryRouter не взаємодіє з браузером (користувач не може ввести адресу конкретного фрагменту і отримати його). Натомість цей маршрутизатор зберігає в пам’яті історію переходів користувача. Як можна побачити, цей підхід не зручний для розробки веб-застосунків, тому переважно він застосовується у мобільних застосунках з використанням React Native.
- HashRouter є застарілим підходом, який використовується лише в тих випадках, коли нові підходи не підтримуються (наприклад, при хостингу сайту на github). Він використовує hash в URL в клієнтській частині. При обробці запиту сервер буде надсилати html файл при цьому ігноруючи хеш частину.
- StaticRouter використовується при SSR, і фактично не змінює своєї позиції.
- NativeRouter використовується для мобільної розробки.
- BrowserRouter є найкращим рішенням для розробки веб-застосунків і використовує HTML 5 історію, браузер обробляє URL і залежно від нього змінює вміст кореневого вузла.

Отже, для веб-застосунків обираємо BrowserRouter. Важливим нюансом є те, що визначити маршрутизацію в додатку можна лише в одному місці, найчастіше це роблять в кореневому вузлі. React Router також дозволяє створювати вкладені шляхи (які складаються з кількох пов’язаних компонентів), що дозволяє батьківському компоненту відслідковувати і керувати оновленням дочірнього.

Для того, щоб використовувати задані шляхи необхідно описати відповідну операцію переходу за потрібною адресою у відповідь на дію користувача. Найпростіший спосіб це зробити – використати Link або NavLink компоненти з React Router, в яких потрібно вказати атрибутом шлях, який має збігатись з одним із шляхів, визначених в маршрутизаторі. Додатково атрибутами можна передати певні параметри або стан. Також можнна примусово викликати навігацію до іншої сторінки за допомогою компоненту Redirect. Іншим підходом є використання навігації за допомогою хуків.

Розглянемо хуки, які використовуються зараз для створення навігації. Завдяки маршрутизатору React завжди знає, на якому саме фрагменті він знаходиться. Для того, щоб дістати це значення можна скористатись хуком useLocation, який поверне об’єкт з даними про поточне положення. Окрім маршруту та параметрів там зберігається інформація про хеш та ключем. За допомогою useNavigate хуку можна переходити на інші підсторінки і передавати дані та стан. Переданий стан можна буде отримати на відповідній сторінці за допомогою описаного вище хуку useLocation. useNavigate є заміною більш ранньому хуку useHistory, який є застарілим в 6 версії React Router. Проте за допомогою useNavigate також можна використовувати історію переходів користувача. Для цього параметром до хуку необхідно передати цифру, а не шлях (наприклад, -1 відповідає попередній сторінці, а 1 наступній).

2.6.2 React Responsive Carousel

Карусель є зручним способом представити певний набір даних з анімацією гортання сторінок. React Responsive Carousel надає готовий

компонент Carousel, в якому необхідно вказати джерело даних та структуру інтерфейсу для відображення. Карусель має анімацію гортання стрічки а також за замовчуванням автоматично гортає стрічку з певним часовим інтервалом. Також карусель надає дефолтну поведінку на скрол користувача (натискання на відповідні клавіші також опрацьовується). Карусель автоматично адаптується під різні розміри сторінки та різні типи пристроїв. Бібліотека генерує повноцінний компонент і навіть його дефолтна реалізація часто використовується в проектах. Проте карусель також можна додатково налаштувати [17] під власні потреби. Можна змінювати властивість автовідтворення (за потреби його можна скасувати або ж встановити інший часовий проміжок для анімації), циклічність каруселі, відповідь на користувацькі дії та оформлення каруселі (відображення елементів та допоміжних компонентів (кнопок для скролу, нумерації тощо).

Карусель є готовим ідеальним рішенням для циклічного відображення елементів з автовідтворенням. Також варто зазначити, що її інтеграція в проект є легкою, і використання не завантажує багато додаткового коду, тож ефективність роботи програми не знизиться.

2.6.3 Робота з 3D об’єктами

Для роботи з 3D в React необхідно використати три бібліотеки. Знадобляться react-three-fiber [16], three.js [17], @react-three/drei [18].

Основним інструментом, який виконує всі завдання, є tree.js, яка надає можливості для відображення браузером 3D моделей завдяки використанню WebGL. За допомогою цієї бібліотеки можна повністю налаштувати об’єкт для «чистого» JavaScript, проте це не буде підходити для React проекту з декларативним стилем описання інтерфейсу. Тут в нагоді стає react-three-fiber, яка надає готові компоненти для роботи з 3D графікою для React. Для цих компонентів будуть працювати всі можливості React для компонентів, а також є доповнений функціонал хуків для доступу до сцени. 3D об’єкти необхідно розміщувати всередині компоненту Canvas, який є кореневим для роботи з

подібними завданнями. За допомогою @react-three/drei можна імплементувати до проекту певну логіку роботи зі світлом, текстурою, діями користувача (наприклад, OrbitControl для обертання об’єкту).

Хоча використання 3D об’єктів може забирати дещо більше ресурсів, ніж звичайний застосунок, використання додаткових інструментів дозволяє оптимізувати цей процес настільки, що для користувача різниця буде не помітною. Також варто зазначити, що під час розробки використовуються звичайні концепції React та компонентна архітектура, які є інтуїтивно зрозумілими.

Розділ 3 : Розробка застосунку

Застосунок має 5 основних сторінок та додаткові можливості після авторизації.

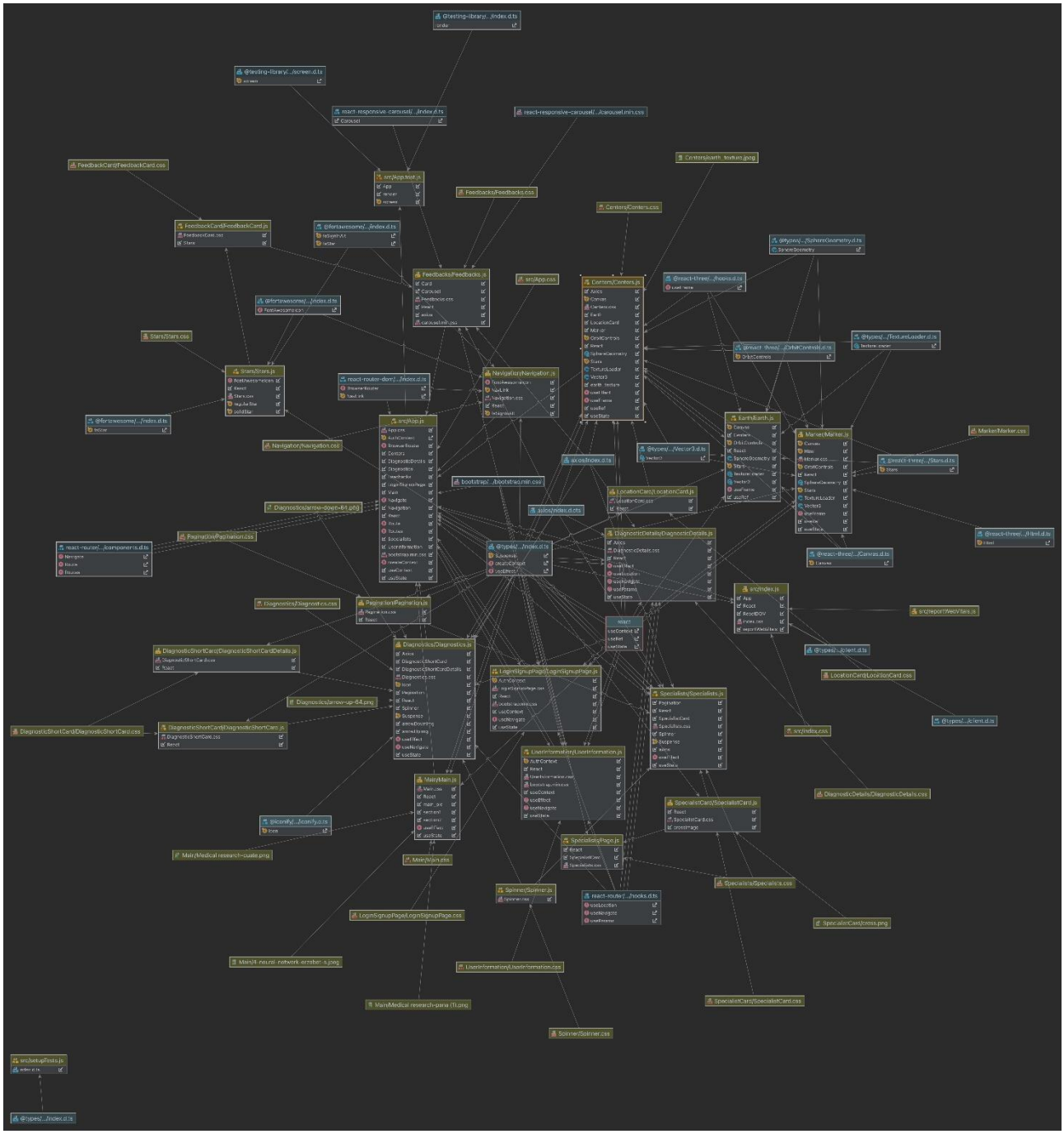

Рис.5 Діаграма класів та зв’язків застосунку

Кожна сторінка та блок навігації є окремими компонентами. Для переміщення по сторінці існує верхня панель навігації, яка використовує компоненти Link з React Router. Всі шляхи визначені в BrowserRouter, який

знаходиться в корені функції App.

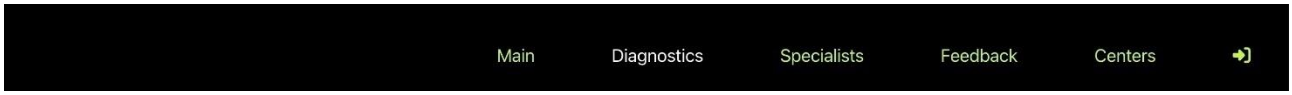

Рис.6 Панель навігації

Головна сторінка містить блок загальної інформації про проект. Для її фрагментів створена анімація, яка створює ефект переміщення до наступного блоку, коли користувач доскролює поточний блок до кінця. Для цього використовуються хуки useState та useEffect, про які було розказано в розділі 2. В useEffect створюється і скасовується підписка на прокручування колеса мишки (wheel), а також відбувається зміна стану, який вказує чи прокручується сторінка, а також стану, який вказує на номер активного фрагменту. Це залежить від різниці між положенням по осі Y, яке було зафіксовано завдяки прослуховуванню мишки.

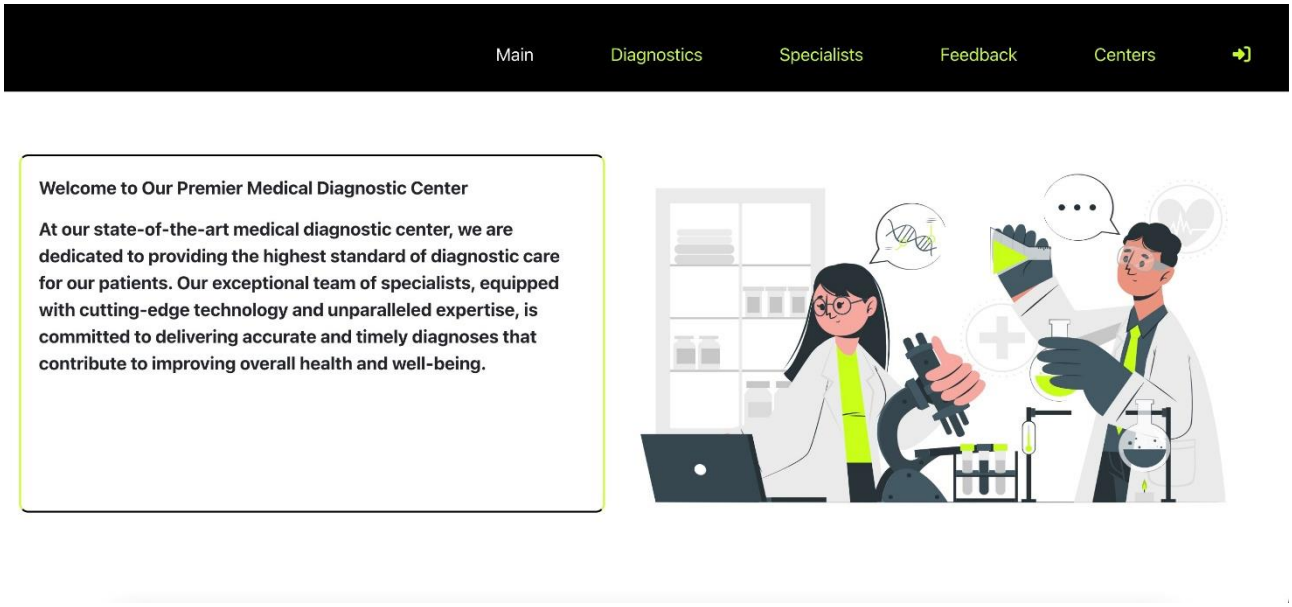

Рис.7 Фрагмент головної сторінки

На сторінці Діагностики користувач може переглянути наявні методи діагностики, здійснити пошук, фільтрацію та сортування.

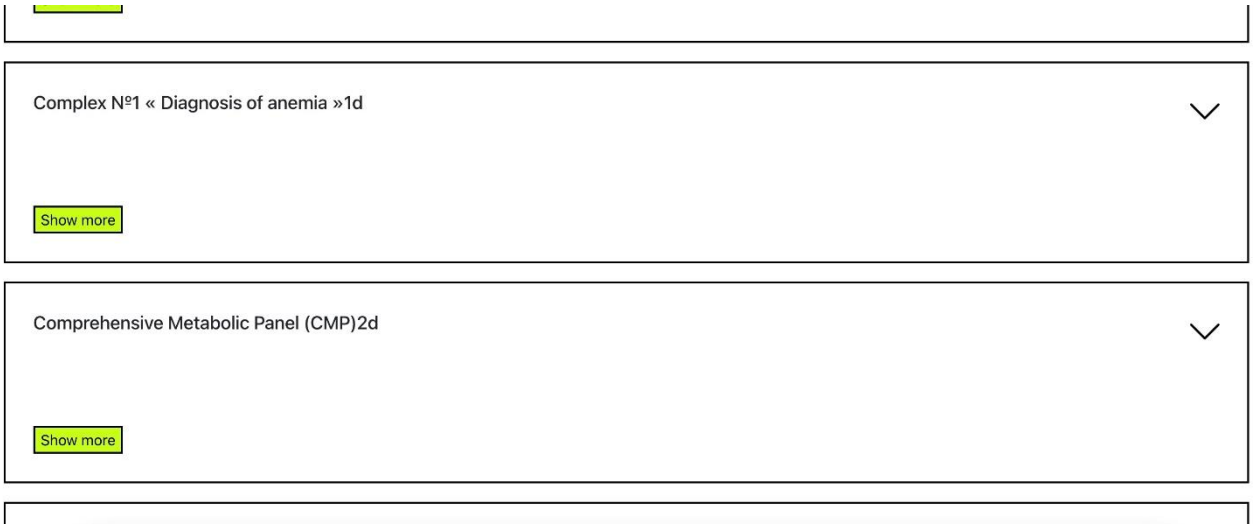

Рис.8 Перегляд методів діагностики

Дані отримуються з таблиці diagnostics за допомогою Axios, який буде здійснювати запит за відповідним endpoint і отримуватиме результат про всі дослідження. Для збереження цих даних використовувався хук useState, таким чином інтерфейс оновлювався при будь-якій взаємодії з користувачем. Для більш зручного представлення даних було додано функціональний компонент пагінації.

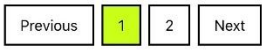

Рис.9 Перегляд методів діагностики

Основний вміст сторінки також розбитий на компоненти (картка діагностики та більш детальної інформації) задля більшої ефективності оновлення сторінки і зручності повторного використання коду.

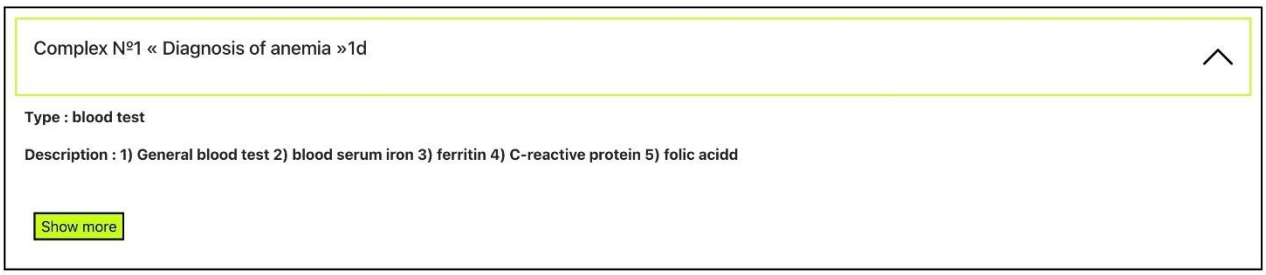

Рис.10 Компонент діагностики та компонент більш детальної інформації

При натисненні на кнопку задля отримання всіх деталей відбувається перехід на окрему сторінку. Навігація виконується за допомогою хуків useNavigate та useLocation з передачею даних про конкретну діагностику, як це було описано в попередньому розділі.

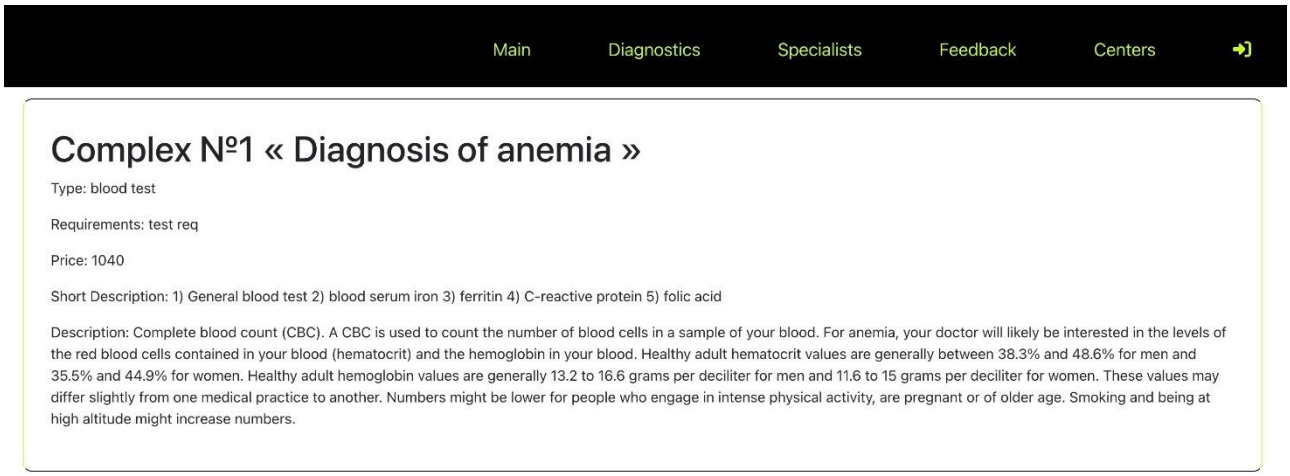

Рис.11 Перехід на сторінку з повною інформацією

Дані сторінки з діагностикою будуть змінюватись кожного разу. Коли змінюватиметься вміст полів фільтрації, сортування та пошуку. Дані цих полів прив’язані до відповідних станів і оновлюватимуться після зміни вмісту. Оновлення стану в свою чергу викликає оновлення компоненту. Таким чином користувач завжди матиме актуальний інтерфейс.

Рис.12 Поля для взаємодії з користувачем

Також є сторінка з спеціалістами, яка також побудована як функціональний компонент. Картки спеціалістів та детальної інформації є окремими компонентами, компонент пагінації повторно використовується з попередньої сторінки.

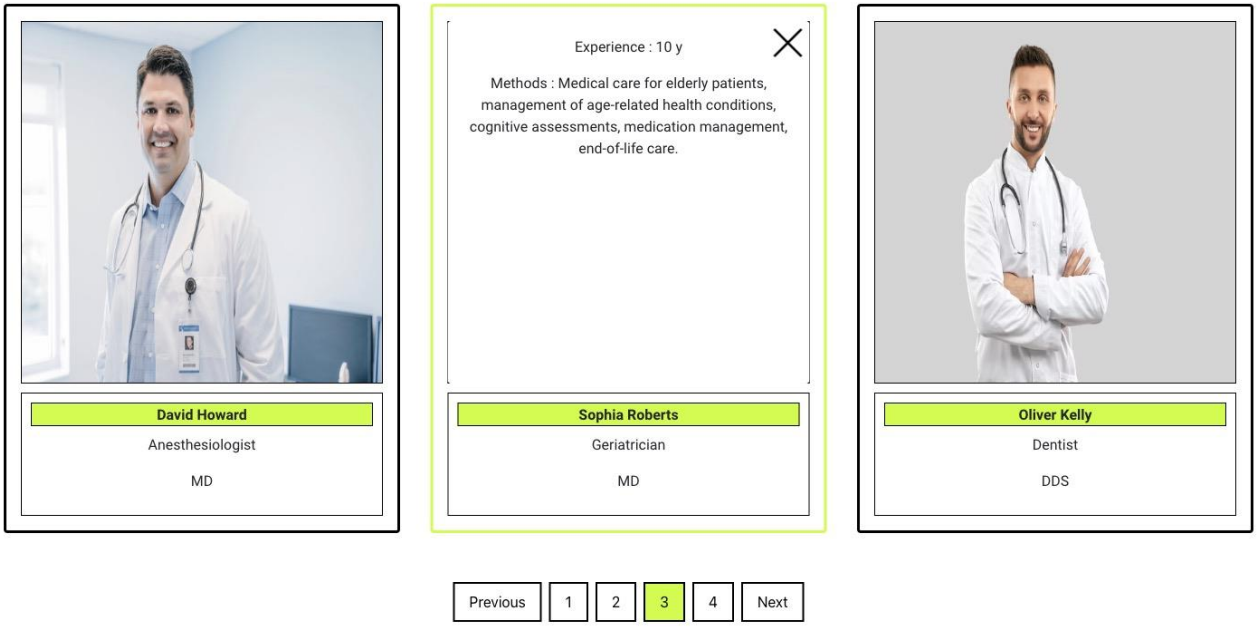

Рис.13 Компоненти сторінки спеціалістів

Для реалізації сторінки відгуків я використовувала React Responsive Carousel. На відміну від попередніх сторінок. Які були реалізовані як функціональні компоненти, сторінка відгуків реалізована за іншим принципом – наслідування від React.Component. Завантаження даних здійснюється за допомогою методу життєвого циклу компонента componentDidMount. Для контролю поточного елементу використовується стан, який зберігає це значення і змінює його після натиску кнопок вперед/назад.

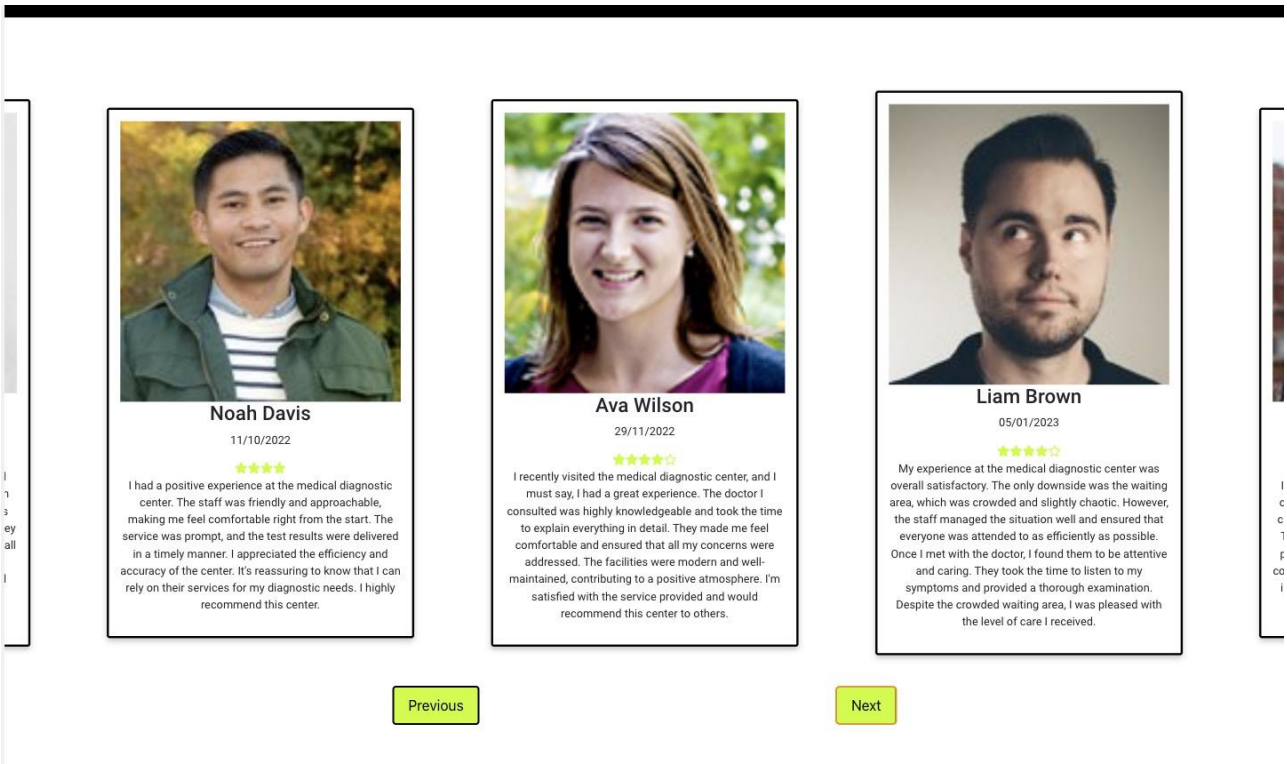

Рис.14 Карусель відгуків

Окрім компоненту картки додатково я реалізувала компонент оцінки, який перетворює числове значення у відображення кількості заповнених зірок.

Для компоненту відображення центрів використовувалась 3D модель землі та 3D точки локацій. Земля представлена сферою з текстурою земною поверхні. Завдяки зберіганню посилання на об’єкт можна встановлювати значення обороту за віссю Y для створення анімації обертання Землі навколо своєї осі. Компонент локацій є сферою зеленого кольору, для якої вираховується позиція на сфері Землі за допомогою перетворення параметрів координат відповідно до радіусу сфери та об’єкту Vector3 бібліотеки tree.js. Також при натисненні на локацію на екрані з’являється інформація про неї. Для цього до локації передається параметр, який вказує, чи відображати інформацію, та функція, яка має реагувати на натискання локації. Сторінка центрів створена як функціональний компонент, який містить компонент Землі, завантажує інформацію з БД про локації і додає відповідні компоненти локацій за координатами. Завдяки OrbitControl з @react-three/drei користувач має змогу самостійно обертати компонент, а компонент Stars додає фон для Canvas.

Рис.15 Компоненти Землі та локацій

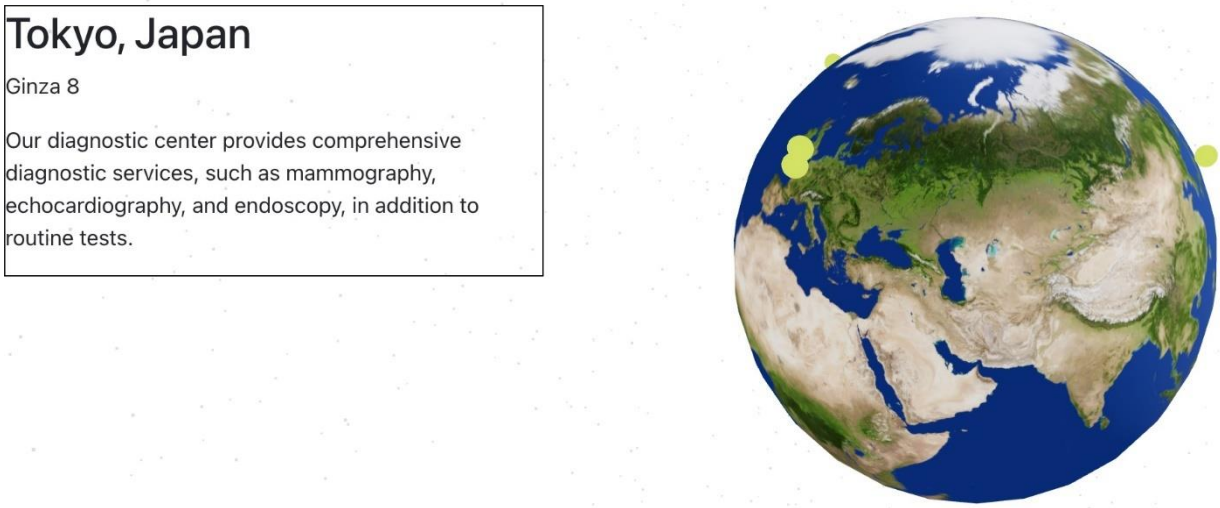

Рис.16 Відображення інформації про натиснену локацію

Для того, щоб запам’ятати поточного зареєстрованого користувача використовується useContext, а підсторінки, які мають бути доступними лише для авторизованих користувачів захищаються PrivateRoute.

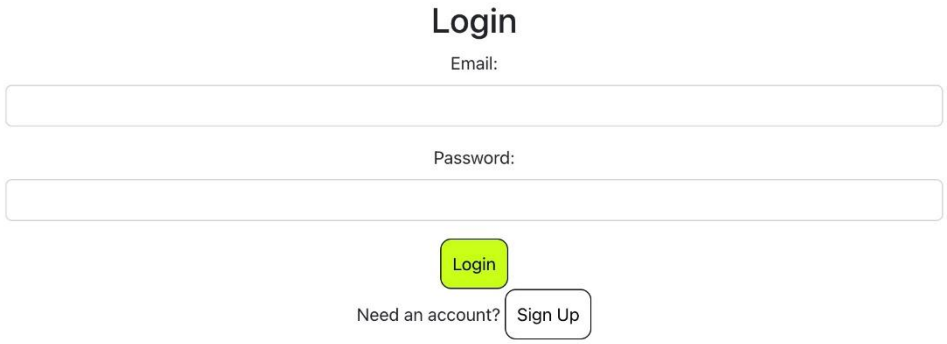

Рис.17 Сторінка реєстрації та логіну

Для безпеки збереження паролів я використала бібліотеку bcrypt, яка шифрує паролі для збереження в БД, а потім зіставляє введені користувачем дані із записом в базі. Після авторизації для користувача стає доступна сторінка з більш детальною формою про медичні дані, а також користувач може отримати список необхідних досліджень на основі введених даних.

Рис.18 Сторінка з особистою інформацією

Висновок

У ході цієї курсової роботи було розглянуто особливості React для розробки веб-застосунків, його переваги над іншими популярними технологіями та ключові принципи роботи.

React значно розширює можливості веб-розробки завдяки своєму декларативному стилю та особливостям оновлення інформації. Саме завдяки принципу опису бажаного вигляду елементу на екрані засобами JSX та віртуальному DOM React надає значну оптимізацію рендерингу лише того елементу, в якому відбулися зміни. Розробка застосунку з використанням React є зручною, оскільки можна будувати структуру компонентів та їх взаємозв’язків між собою. Окремі компоненти можна повторно використовувати за необхідності, а завдяки хукам можна винести окремо часто вживану логіку і застосувати до багатьох компонентів. Існує багато підходів та інструментів для контролю кожного етапу створення та знищення компоненту, що робить розробку більш гнучкою і дає багато однаково ефективних альтернатив для розв’язку поставленого завдання. Разом зі своїми бібліотеками React може вирішити будь-які задачі, що стосуються розробки інтерфейсу. React має дуже якісну та чітку документацію, тож завжди можна знайти відповіді на свої питання.

Отже, React є ефективним рішенням для розробки завдяки своїм новим підходам до побудови застосунку. Він має менше обмежень ніж інші інструменти, оскільки є бібліотекою, та займає лідерську позицію серед інструментів для розробки інтерфейсу. Команда постійно вносить зміни до бібліотеки, досліджує тенденції розробки та додає нові рішення та принципи, які пришвидшують роботу додатку. Тож можна стверджувати, що React і надалі буде одним із лідерів ринку серед інструментів веб-розробки.

Список прийнятих скорочень

DOM – Document Object Model

VDOM – Virtual Document Object Model

MVC – Model View Controller

Props – скорочення від properties

Refs – скорочено від references

Effect – скорочено від effectful code

SPA – Single Page Application

SSR – Server Side Rendering

Список використаних джерел

[1] Framework vs Library in software development: What is the difference? [Електронний ресурс] – Режим доступу до ресурсу: https://kruschecompany.com/framework-vs-library/.

[2] Wieruch R. The Road to React: Your journey to master plain yet pragmatic React.js. Independently published, 2018. 226 с.

[3] Gackenheimer C. Introduction to React. Apress, 2015. 148 с.

[4] React Usage Statistics [Електронний ресурс] – Режим доступу до ресурсу: https://trends.builtwith.com/javascript/React.

[5] Front-end frameworks and libraries [Електронний ресурс] – Режим доступу до ресурсу: https://2021.stateofjs.com/en-US/libraries/front-end-frameworks/.

[6] Top 10 Big Companies Using React [Електронний ресурс] – Режим доступу до ресурсу: https://careerkarma.com/blog/companies-that-use-react/.

[7] The Diffing Algorithm [Електронний ресурс] – Режим доступу до ресурсу: https://legacy.reactjs.org/docs/reconciliation.html#the-diffing-algorithm.

[8] Introducing JSX [Електронний ресурс] – Режим доступу до ресурсу: https://legacy.reactjs.org/docs/introducing-jsx.html.

[9] Modern Web-Development using ReactJS [Електронний ресурс] – Режим доступу до ресурсу: http://ijrra.net/Vol5issue1/IJRRA-05-01-27.pdf.

[10] React.Component [Електронний ресурс] – Режим доступу до ресурсу: https://legacy.reactjs.org/docs/react-component.html.

[11] Deep dive: How do React hooks really work? [Електронний ресурс] – Режим доступу до ресурсу: https://www.netlify.com/blog/2019/03/11/deep-dive-how-do-react-hooks-really-work/.

[12] Introducing Concurrent Mode [Електронний ресурс] – Режим доступу до ресурсу: https://17.reactjs.org/docs/concurrent-mode-intro.html.

[13] Banks A., Porcello E. Learning React: Functional Web Development with React and Redux. O'Reilly Media, 2017. 350 с.

[14] ReactJS Types of Routers [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/reactjs-types-of-routers/.

[15] react-responsive-carousel [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/leandrowd/react-responsive-carousel.

[16] react-three-fiber [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/pmndrs/react-three-fiber.

[17] three.js [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/mrdoob/three.js/.

[18] drei [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/pmndrs/drei.