Смолянюк В'ячеслав,
здобувач першого (бакалаврського) рівня вищої освіти
фізико-математичного факультету
Науковий керівник: ***Мосіюк Олександр,***
кандидат педагогічних наук, доцент,
доцент кафедри комп'ютерних наук та інформаційних технологій,
Житомирський державний університет імені Івана Франка,
м. Житомир, Україна

ПЕРЕВАГИ ВИКОРИСТАННЯ БІБЛІОТЕКИ REACT У ПРОЄКТАХ ВЕБРОЗРОБКИ

Актуальність. У сучасній веброзробці створення ефективних, масштабованих та інтерактивних інтерфейсів є важливою задачею для програмістів. З розвитком технологій і збільшенням вимог до продуктивності вебзастосунків, використання сучасних інструментів, таких як бібліотека React, стає особливо актуальним. Вона дозволяє значно полегшити процес створення складних інтерфейсів завдяки компонентному підходу, гнучкому управлінню станом та високій продуктивності. А поєднання React з програмними рішеннями для управління станом (Redux) [3] та інструментами для оптимізації коду (Webpack і Babel) [2, 5] дають змогу створювати застосунки, які відповідають вимогам швидкості завантаження й зручності використання.

У цьому контексті **метою статті** є розкриття основних особливостей використання React у проєктах веброзробки.

Виклад основного матеріалу. Головною особливістю бібліотеки React є компонентний та реактивний підходи до розробки, що дозволяє швидко

створювати масштабовані, продуктивні та адаптивні вебзастосунки. До основних переваг варто віднести технології JSX, Virtual DOM і Hooks. Вони допомагають забезпечити легкість створення програм та якісну тривалу підтримку навіть складних проєктів [1]. Розглянемо їх детальніше.

Компонентний підхід React надає змоги розділити сайт на окремі частини, де кожеа відповідає за певний елемент інтерфейсу. Це дозволяє розробникам змінювати або ж вдосконалювати один окремо взятий компонент без необхідності переписування всього сайту.

Такий підхід також забезпечує повторне перевикористання коду, що спрощує процес підтримки. У великих проєктах окремі фахівці можуть працювати над своїми компонентами, що також значно прискорює розробку. Наприклад, кнопка з однаковим дизайном, але з різними функціями, може бути універсальним компонентом, який налаштовується залежно від місця використання. Зокрема є можливість задати текст на кнопці й визначити дію при натисканні.

Virtual DOM та продуктивність. Однією з ключових інновацій React є Virtual DOM, який знижує кількість оновлень реальної моделі документа. Під час зміни стану React значення задаються спершу для віртуальної моделі, а потім вони синхронізуються з реальним DOM. Це знижує навантаження на браузер і прискорює час відгуку, що особливо важливо для великих додатків із великою кількістю операцій.

JSX є синтаксичним розширенням JavaScript, яке дозволяє вставляти HTML-подібний код у JavaScript, що зручно і підвищує його читабельність. Завдяки JSX розробники можуть об'єднувати розмітку й логіку в одному місці. Крім того, інтеграція JSX з інструментами розробки, такими як Babel, допомагає виявляти помилки ще на етапі компіляції.

Hooks для управління станом і побічними ефектами. Hooks – це набір функцій, які надають змогу використовувати стан і інші можливості React у функціональних компонентах. Наприклад, useState надає простий спосіб створення локального стану, а useEffect дозволяє додавати побічні ефекти (такі як запити до сервера) без необхідності в класових компонентах. Завдяки своїй простоті й гнучкості, Hooks стали популярним інструментом для організації коду.

React Context для управління глобальним станом – це інструмент, який дозволяє створювати та ділитися глобальним станом або параметрами між компонентами у додатку без потреби передавати його на кожному рівні. Це особливо корисно, коли певні дані використовуються у багатьох частинах додатка, наприклад, зміна теми, налаштування мови або інформацію про користувача.

Context складається з «постачальника» (Provider) та «споживача» (Consumer). Provider визначає «контекст» – значення або стан, доступний іншим компонентам, і обгортає в собі ту частину додатка, де ці значення будуть використовуватися. У ньому задається значення, доступне всім дочірнім компонентам. Consumer – компонент, що підписується на контекст і дозволяє отримати доступ до значень з Provider. У функціональних компонентах

підключення до контексту зазвичай реалізується через хук useContext, що робить доступ до глобальних даних ще простішим і гнучкішим [4].

Наприклад, у застосунку можна створити «глобальну тему» (світлу або темну) і використовувати її в різних компонентах для єдиного відображення інтерфейсу. Context API дозволяє змінювати «тему» в одному місці, автоматично оновлюючи її у всіх компонентах, що її використовують.

Зовнішні бібліотеки для управління станом. У масштабних проєктах для управління станом часто використовують зовнішні бібліотеки, такі як Redux або MobX. Вони пропонують централізоване сховище даних, яке дозволяє всім компонентам застосунку отримувати доступ до необхідної інформації без необхідності передавати її через «пропси». Це особливо корисно, коли стан є складним або його потрібно використовувати в багатьох частинах інтерфейсу.

Однією з головних переваг Redux є його передбачуваність: зміни стану виконуються через спеціальні функції, які називаються reducers. Reducers обробляють дії (actions) і створюють новий стан на основі попереднього [3]. Це дозволяє точно контролювати, як і коли змінюється стан додатка. Кожен action містить опис змін, які повинні бути внесені, що дає можливість легко відслідковувати зміни і запобігати помилкам у роботі з даними.

За допомогою таких інструментів, як Redux DevTools, розробники можуть здійснювати повний моніторинг змін стану, що робить налагодження набагато простішим. Крім того, передбачуваність обробки стану в Redux спрощує тестування, оскільки кожна зміна стану може бути відтворена на основі тих самих даних і дій.

У порівнянні з Redux, MobX є більш гнучким, оскільки дозволяє використовувати «реактивний» підхід до управління станом, де компоненти автоматично підписуються на зміни у певних даних і оновлюються, коли ці дані змінюються. Це знижує складність написання коду, але може бути менш передбачуваним, ніж Redux, у великих проєктах, де важливо мати чітко визначену структуру змін стану.

Окремо необхідно звернути увагу на процес «компіляції» (об'єднання всіх компонентів і залежностей в один JavaScript-файл). Процес розробки вебзастосунку за допомогою React передбачає створення численних окремих файлів, зокрема компонентів, стилів, зображень та інших залежностей. Це надає змогу зберігати код організованим та модульним, що полегшує його підтримку і розширення. Однак, коли сайт готовий до публікації, важливо об'єднати їх в один оптимізований JavaScript-файл, що сприятиме кращій продуктивності і швидкому завантаженню сторінки. Для цього часто використовуються інструменти, такі як Webpack і Babel.

Webpack – інструмент для збірки JavaScript-застосунків, який допомагає звести різні файли в один, а також оптимізує їх для швидшого завантаження [5]. Він дозволяє визначити, які файли повинні бути включені у фінальну збірку, і обробляє їх таким чином, щоб браузер завантажував тільки необхідні частини коду. Це зменшує обсяг завантажуваних даних і забезпечує швидке відображення вебсторінок. Webpack також дозволяє використовувати додаткові

плагіни для оптимізації, наприклад, для мінімізації файлів, що зменшує їхній розмір.

Babel представляє собою транспайлер JavaScript, який перетворює сучасний JavaScript-код (наприклад, код, написаний з використанням ES6+ функцій) на формат, який підтримується більшістю браузерів. Оскільки React використовує JSX – синтаксичне розширення для JavaScript, яке не завжди підтримується безпосередньо, то Babel надає змогу перетворити JSX-код у звичайний JavaScript, сумісний з різними версіями браузерів [3].

Підсумовуючи викладене вище зауважимо, що бібліотека React є одним із найпопулярніших інструментів для створення інтерфейсів користувача. Однією з основних її переваг є компонентний підхід, що дозволяє структурувати застосунок у вигляді автономних блоків, які легко підтримувати, змінювати і повторно використовувати. Кожен компонент відповідає за свою частину інтерфейсу, що сприяє швидкому розробленню та зручності в масштабуванні проєктів.

Ще однією важливою перевагою є Virtual DOM, який забезпечує високу продуктивність. Коли стан додатка змінюється, React спочатку оновлює віртуальну модель DOM, а потім синхронізує ці зміни з реальним DOM, мінімізуючи кількість необхідних маніпуляцій з реальним DOM.

Перевагою React є також ефективне управління станом. Завдяки таким інструментам, як Hooks і Context API, React надає змогу розробникам гнучко керувати локальними і глобальними станами. Hooks забезпечують простоту і зручність у використанні функціональних компонентів, а Context API сприяє ефективній передачі даних між компонентами без потреби у використанні пропсів через всі їх рівні вкладеності.

Завдяки цим перевагам – компонентному підходу, Virtual DOM, зручному управлінню станом за допомогою Hooks і Context API, а також гнучкості – React є оптимальним вибором для сучасної веброзробки, що дозволяє створювати швидкі, масштабовані та ефективні застосунки.

Список використаних джерел та літератури

1. Офіційна документація React. URL: https://reactjs.org/docs/getting-started.html (дата звернення: 6.11.2024р.)

2. Babel Documentation URL: https://babeljs.io/docs/en/ (дата звернення: 7.11.2024р.)

3. Redux Documentation. URL: https://redux.js.org/ (дата звернення: 6.11.2024р.)

4. useContext Documentation URL: https://react.dev/reference/react/ useContext (дата звернення: 7.11.2024р.)

5. Webpack Documentation URL: https://webpack.js.org/concepts/ (дата звернення: 7.11.2024р.)