ПРИРОДНИЧІ ТА ТЕХНІЧНІ НАУКИ

УДК 004.4:(04.416.3: 004.451.84):(004.451:004.057)

ВПРОВАДЖЕННЯ АДАПТИВНОГО ДИЗАЙНУ З REACT: ТЕХНІКИ ДЛЯ КРОСПЛАТФОРМНОЇ СУМІСНОСТІ

Д. С. Афанасьєва, О. В. Зелінська

Анотація. У дослідженні розглядається використання React у поєднанні з різними техніками адаптивного дизайну для забезпечення кросплатформної сумісності. Описані різні підходи до адаптивності і їх переваги та недоліки. Детально розглянуто використання бібліотеки React-Socks для реалізації адаптивного дизайну та його можливості. Проаналізовано використання CSS Grid Layout та Flexbox в контексті адаптивності вебінтерфейсів. Результати дослідження вказують на значний потенціал React для створення адаптивних і гнучких інтерфейсів, які забезпечують комфортний користувацький досвід на різних пристроях та екранах.

Ключові слова: React, адаптивний дизайн, CSS, FlexBox, React-Socks.

Вступ. Наслідком стрімкого розвитку технологій сучасного світу стало виникнення величезної кількості різноманітних пристроїв та платформ. Саме тому перед фахівцями з галузей веброзробки та дизайну постало складне завдання з реалізації інструментів, що дадуть змогу будувати інтерфейси різної складності з можливістю адаптації їх елементів до девайсів з відмінними один від одного розмірами.

З проведеного аналізу джерел [1–5] видно, що розвиток адаптивного інтерфейсу в React ведеться активною спільнотою фахівців. У них розглядають використання CSS з медіазапитами для гнучкого керування стилями елементів, інші звертають увагу на гнучкий макет FlexBox, а також на використання CSS Grid Layout для складних макетів. Згідно з аналізом, використання спеціалізованих бібліотек для React дає змогу ефективно керувати точками переривання для різних розмірів екранів. Проте варто враховувати, що деякі з цих бібліотек можуть вимагати більше зусиль для налаштування. Загалом різні підходи та інструменти можуть бути застосовані для досягнення адаптивності в React-додатках, і вибір конкретного методу повинен базуватися на вимогах проєкту.

Виклад матеріалу. Адаптивний дизайн та гнучкий дизайн є новими поняттями у сфері веброзробки, вони відображають важливі аспекти забезпечення оптимального користувацького досвіду на різних пристроях і екранах. Адаптивний дизайн визначається як підхід до створення інтерфейсів, де розміри елементів, їх розташування та вирівнювання адаптуються залежно від конкретних роздільних здатностей екранів [1]. Гнучкий дизайн передбачає прописування стилів елементів так, щоб їх вміст автоматично адаптувався до роздільної здатності пристрою, на якому відображається сайт [1]. У цьому випадку контент не має чітко визначених розмірів, а замість цього всі елементи підлаштовуються під різні типи екранів. Гнучкий дизайн використовується для створення більш універсальних інтерфейсів, які легко адаптуються до різних пристроїв без необхідності встановлення конкретних розмірів для кожного елементу. Доволі часто принципи адаптивного та гнучкого дизайну комбінують, аби досягти найкращого результату.

React – JavaScript-фреймворк з відкритим кодом, розроблений компанією Meta для швидкого та ефективного створення користувацьких інтерфейсів (UI) різної складності [2]. Основним структурним елементом React є компоненти – окремі частини коду у вигляді функцій (у більш старих вебсайтах використовувались класи). Кожен компонент містить певні елементи, стилі та логіку. Використовувати один і той самий компонент у проєкті можна багато разів.

Додавати адаптивність вебсайта в React можна великою кількістю способів. Першим варіантом є використання правильно відформатованого CSS у поєднанні з написанням ReactJS-коду, що можна легко адаптувати. Принцип написання такого коду називається «reactive» (той, що реагує), він базується на побудові логіки на основі потоків даних, що здатні змінюватись [3].

Адаптивність з використанням CSS досягається багатьма способами. Першим варіантом є застосування медіазапитів. Додаються вони в кінці файла і починаються з ключового слова @media. Після цього прописуються розміри екранів, до яких застосовуються стилі, що описа-

ні у запиті. Наприклад, запис (min-width: 480px) and (max-width: 1024px) означає, що стилі цього запиту будуть додані до елементів, якщо ширина екрана девайсу буде в межах 480–1024 px (рис. 1) [4].

```
@media only screen and (max-width: 480px) {
  h1 {
    color: white;
  }
}

@media only screen and (min-width: 480px) and (max-width: 1024px) {
  h1 {
  color: red;
  }
}
```

Рис. 1. Медіазапити для зміни кольору заголовків залежно від розміру екрана

Наступним варіантом використання CSS є додавання FlexBox Layout до елементів інтерфейсу. Основною концепцією гнучкого макета є можливість контейнера змінювати ширину / висоту своїх елементів (і їх порядок), щоб найбільш оптимально заповнити доступний простір (переважно для адаптації до різних пристроїв та розмірів екранів) [5]. Розглянемо приклад, де його було використано для розміщення трьох елементів у горизонтальному рядку (рис. 2).

```
const FlexboxExample = () => {
  return (
    <div style={containerStyle}>
      <div style={itemStyle}>Елемент 1</div>
      <div style={itemStyle}>Елемент 2</div>
    </div>
  );
}

const containerStyle = {
  display: 'flex',
};

const itemStyle = {
  flex: 1,
  padding: '10px',
};
```

Рис. 2. Використання FlexBox для адаптивності

Змінна «containerStyle» встановлює батьківському елементу властивість «display: flex», щоб усі дочірні елементи відображалися як гнучкий контейнер. Клас «itemStyle» встановлює усім нащадкам однакову гнучкість, розподіляючи доступний простір рівномірно між ними. Отже, під час зменшення розміру екрана елементи з класом «itemStyle» будуть підлаштовувати свій розмір під його ширину.

Наступним прикладом додавання адаптивності є CSS Grid Layout. Це потужний модуль CSS, який надає зручний та гнучкий спосіб створення сіток для організації контенту на вебсторінках. CSS Grid дає змогу створювати складні макети з різними розмірами та пропорціями елементів, що допомагає створювати різноманітні дизайни без необхідності використання складних класів або фреймворків [4]. На рис. 3 показано приклад використання сітки з двома колонками, кожна з яких займає однакову частину вільного простору, а між елементами встановлено відступ у 10 пікселів.

Додавання адаптивності та гнучкості в інтерфейс React-додатків може бути досягнуте за допомогою використання сторонніх бібліотек. Одна з таких бібліотек – React-Socks. Вона пропонує потужні інструменти для реалізації адаптивного дизайну, даючи змогу компонентам реагувати на зміни розміру екрана та адаптуватися до них. Основна перевага React-Socks по-

лягає у його здатності спростити процес взаємодії з медіазапитами та розширити можливості адаптивного дизайну. Ця бібліотека надає зручний і ефективний спосіб визначати точки переривання для різних розмірів екрана, а також контролювати відображення компонентів на кожній з цих точок. Під час використання React-Socks ви можете легко визначати стандартні точки переривання, як-от small, medium та large, що спрощує процес адаптації додатка до різних пристроїв і розмірів екрана. До того ж бібліотека дає змогу визначати власні точки переривання, що допомагає забезпечити більшу гнучкість і контроль над адаптивним дизайном.

```
const GridExample = () => {
  return (
    <div style={{
      display: 'grid',
      gridTemplateColumns: '1fr 1fr',
      gap: '10px' }}>
      <div>Елемент 1</div>
      <div>Елемент 2</div>
    </div>
  );
}
```

Рис. 3. Використання Grid Layout для адаптивності

Для використання React-Socks у проєкті потрібно буде використовувати компоненти «Breakpoint» та «BreakpointProvider». «Breakpoint» дає змогу визначати точки переривання для різних розмірів екрана, а «BreakpointProvider» забезпечує передачу інформації про розмір екрана всім компонентам у дереві відображення. Хоча використання React-Socks може забезпечити більшу самостійність компонентів і спростити роботу з адаптивним дизайном, варто пам’ятати, що іноді вона може вимагати більше зусиль для налаштування, особливо у складних випадках, порівняно з традиційними методами, як-от медіазапити [6].

На рис. 4 використовується компонент «Breakpoint» з бібліотеки React-Socks, щоб динамічно змінювати вміст залежно від розміру девайса. Наприклад, «Breakpoint small down» відображає контент лише на екранах із невеликим розміром, а «Breakpoint large up» – на екранах великого розміру [6].

```
const App = () => {
  return (
    <BreakpointProvider>
      <div>
        <Breakpoint small down>
          <p>Small screen content</p>
        </Breakpoint>
        <Breakpoint large up>
          <p>Large screen content</p>
        </Breakpoint>
      </div>
    </BreakpointProvider>
  );
}
```

Рис. 4. Використання бібліотеки React-Socks для адаптивності

На основі проведених досліджень можна виокремити конкретні переваги та недоліки кожного з методів та бібліотек для створення адаптивного інтерфейсу в React. Основною перевагою CSS із медіазапитами є простота використання та реалізації. Цей підхід дає змогу гнучко

керувати стилями для різних розмірів екрана. Однак зі збільшенням кількості медіазапитів можуть виникнути труднощі управління складними структурами CSS.

Далі розглянемо FlexBox Layout. Цей метод відзначається легкістю використання для створення простих гнучких макетів і простим розміщенням елементів у горизонтальному та вертикальному напрямках. Проте він може бути обмеженим для складних макетів, де потрібно більше контролю над розмірами та порядком елементів. CSS Grid Layout надає потужні інструменти для створення складних та різноманітних макетів. Він дає змогу гнучко керувати розмірами та порядком елементів у сітці. Однак для оволодіння цим методом може знадобитися додатковий час та зусилля. Бібліотека React-Socks є зручним та ефективним способом реалізації адаптивного дизайну в React-додатках. Вона дає змогу легко визначати точки переривання та контролювати відображення компонентів. Проте використання React-Socks може вимагати більше зусиль для налаштування, порівняно з традиційними методами, як-от медіазапити.

Висновки. Здійснивши глибокий аналіз інструментів для реалізації адаптивного дизайну у React-проєктах, було виявлено, що використання медіазапитів, Flexbox, CSS Grid Layout та React-Socks відображають різні підходи до створення адаптивних інтерфейсів. Проведений аналіз дає змогу краще зрозуміти переваги та недоліки кожного інструменту й обирати оптимальний для конкретного продукту. Проте результати показують, що використання CSS є вирішальним. Дослідження підтверджує, що медіазапити, Flexbox та CSS Grid Layout – найефективніші засоби для створення адаптивних інтерфейсів, адже вони забезпечують гнучкість і контроль над розміщенням елементів на сторінці, а також дають змогу легко адаптувати їх до різних пристроїв. Правильне використання цих CSS-технологій сприяє поліпшенню користувацького досвіду та забезпечує кращу кросплатформну сумісність.

Abstract. The study examines the usage of React in conjunction with various adaptive design techniques to ensure cross-platform compatibility. It describes different approaches to adaptivity, discussing their pros and cons. Detailed exploration of utilizing the React-Socks library for adaptive design and its capabilities is provided. The usage of CSS Grid Layout and Flexbox in the context of web interface adaptivity is analyzed. Research findings indicate significant potential of React in creating adaptive and flexible interfaces, ensuring a comfortable user experience across different devices and screens.

Keywords: React, adaptive design, CSS, FlexBox, React-Socks.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Adaptive?! Responsive? Reactive! *Medium.* URL: https://medium.com/react-camp/adaptive-responsive-reactive-62fb938d6191 (дата звернення: 15.02.2024).

2. Як зробити сайт адаптивним: React JS. *SUNDRIES.* URL: https://sundries.ua/iak-zrobyty-sait-adaptyvnym-react-js/ (дата звернення: 16.02.2024).

3. Reactive Programming With React and RxJs. *Medium.* URL: https://betterprogramming.pub/reactive-programming-with-react-and-rxjs-88d2789e408a (дата звернення: 16.02.2024).

4. How to make React App Responsive using react-responsive? *BrowserStack*. URL: https://www.browserstack.com/guide/how-to-make-react-app-responsive (дата звернення: 17.02.2024).

5. A Complete Guide to Flexbox. *CSS-Tricks.* URL: https://css-tricks.com/snippets/css/a-guide-to-flexbox/ (дата звернення: 17.02.2024).

6. 3 Ways To Implement Responsive Design In Your React App. *Medium*. URL: https://itnext.io/3-ways-to-implement-responsive-design-in-your-react-app-bcb6ee7eb424 (дата звернення: 17.02.2024).

7. Федоренко Є. О., Зелінська О. В. Інструменти розробки онлайн-сервісів. *Прикладні інформаційні технології: матеріали III Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених.* (м. Вінниця, 22 квітня 2022 р.). Вінниця: ДонНУ імені Василя Стуса, 2022. С. 130–132. URL: https://jait.donnu.edu.ua/article/view/12286