МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПЗВО «МІЖНАРОДНИЙ КЛАСИЧНИЙ УНІВЕРСИТЕТ
імені ПИЛИПА ОРЛИКА»
Економіко-технологічний факультет
Кафедра інженерних технологій

Кваліфікаційна робота
на здобуття освітнього ступеня магістра
за освітньою програмою «Комп'ютерна інженерія»
зі спеціальності 123 «Комп'ютерна інженерія»

на тему: «РОЗРОБКА ТА ОПТИМІЗАЦІЯ УНІВЕРСАЛЬНИХ ВЕБ-ІНТЕРФЕЙСІВ З ВИКОРИСТАННЯ БІБЛІОТЕКИ REACH»

Виконав:
здобувач ІІ курсу, групи КІ -20-24
Волошин Владислав Ігорович

Керівник:
к.т.н., доцент кафедри інженерних технологій
Гайша Олександр Олександрович

Миколаїв – 2024

АНОТАЦІЯ

В роботі виконано дослідження питання побудови універсальних інтерфейсів для веб-додатків на основі використання бібліотеки React. Докладно проаналізовано наявні науково-технічні джерела з даної тематики, виконано системний аналіз проблеми. На основі дослідження типових задач користувача при роботі з веб-додатками зроблено моделювання системи та проектування базових алгоритмів її роботи. Сформовано загальні вимоги до інтерфейсів користувача веб-додатків та розглянуто їх особливості стосовно до реалізації на бібліотеці React; на основі цього розроблено конкретний програмний продукт, властивості якого досліджувалися в роботі. Проведено документаційне забезпечення розробленого програмного продукту та його тестування. Робота має практичну направленість і може впроваджуватися у реальні веб-додатки та інформаційні системи.

ABSTRACT

This paper investigates the issue of building universal UI for web applications based on the React library using. Existing scientific and technical sources on this topic are analyzed in detail, system analysis of the problem is carried out. Based on the study of typical user tasks when working with web applications, the system modeling and design of basic algorithms of its work were done. The general requirements for user interfaces of web applications have been developed and their peculiarities in relation to implementation on the React library have been considered; based on this, a specific software product was developed whose properties were investigated in the work. Documentation of the developed software product and its testing is carried out. The work is practical and can be implemented in real web applications and information systems.

Зміст

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ……..5
ВСТУП……6
РОЗДІЛ 1. Загальна характеристика бібліотеки React, її призначення та особливості використання……9
1.1. Аналіз особливостей бібліотеки React мови JavaScript для створення інтерфейсів користувача веб-додатків……9
1.2. Постановка задачі дослідження……..11
РОЗДІЛ 2. Аналітичний огляд існуючих джерел, присвячених розробці інтерфейсів користувача з використанням React…….13
2.1.Огляд літературних друкованих джерел, присвячених бібліотеці React13
2.2. Аналіз електронних публікацій про бібліотеку React…….16
РОЗДІЛ 3. Моделювання процесів розробки та оптимізації універсальних веб-інтерфейсів…….22
3.1. Моделювання проблемної галузі……..22
3.2. Вибір методів та засобів для розробки та оптимізації універсальних веб-інтерфейсів з використанням бібліотеки React……28
3.3. Моделювання інформаційних потоків системи та розробка її схематичного представлення…….33
3.4. Розробка та візуалізація алгоритмів розробки та оптимізації універсальних веб-інтерфейсів……..34
РОЗДІЛ 4. Особливості програмної реалізації створених алгоритмів розробки та оптимізації універсальних веб-інтерфейсів…….37
4.1. Опис загальної структури програмного рішення……37
4.2. Конкретизація особливостей інтерфейсу користувача системи……45
4.3. Особливості реалізації у висхідних кодах спроектованих алгоритмів47
РОЗДІЛ 5. Експериментальне дослідження спроектованої системи……50
5.1. Інструкція адміністратора системи……...50
5.2.Розробка керівництва користувача створеного програмного продукту50
5.3. Опис вимог до апаратного забезпечення……..51

5.4. Тестування системи та опис результатів її роботи……..52
РОЗДІЛ 6. Економічне обґрунтування ефективності розробленого рішення.
ВИСНОВКИ……..……..59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ……..61
ДОДАТКИ……63

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

AJAX	Asynchronous Javascript And Xml
ASP	Active Server Pages
CSS	Cascading Style Sheet
CSV	Comma Separated Values
DoS	Denial of Service
DDoS	Distributed Denial of Service
GUI	Graphics User Interface
HTML	HyperText Markup Language
JSP	Java Server Pages
PC	Personal Computer
PHP	Personal Home Page, **P**HP **H**ypertext **P**reprocessor
SQL	Structured Queries Language
WWW	World Wide Web
XML	eXtensible Markup Language
БД	База даних
ІТ	Інформаційні технології
ОО	Об’єктно-орієнтований (-на, -не)
ООП	Об’єктно-орієнтоване програмування
ПЗ	Програмне забезпечення
ПК	Персональний комп’ютер
СОІ	Система обробки інформації
СУБД	Система управління базами даних

ВСТУП

Людське суспільство за час свого існування пройшло через певні характерні фази, якими були сільськогосподарське (аграрне) суспільство, на зміну якому прийшло індустріальне, що охарактеризувалося бурхливим розвитком техніки. В кінці XX ст. акценти від розробки технічних засобів почали зміщуватися у бік інформаційних технологій. Інформаційною називається технологія, направлена на отримання, обробку та збереження інформації. Розвиток комп'ютерної техніки та супутніх їм інформаційно-комунікаційних технологій призвів до виникнення небувалих раніше можливостей по отриманню інформації самого різноманітного плану: миттєво доступними є як звичайні довідкові відомості, так і вузькоспеціалізовані, поточні дані про параметри самих різних систем (технічних, економічних, кліматичних, природних, і т.д., і т.п.).

Очевидно, що для успішного розвитку сучасного інформаційного суспільства необхідно слід удосконалювати (або розробляти нові) технології отримання інформації, переробки, збереження та відображення користувачеві. Останнє питання, з технічної точки зору може і є менш важливим за три попередніх, однак все одно вимагає значної уваги, оскільки зручне представлення даних може в рази підвищувати продуктивність праці оператора у відповідній системі обробки інформації (далі – СОІ). Таким чином, постає класична задача розробки ефективних підходів до побудови інтерфейсу користувача для прикладного програмного забезпечення (далі – ПЗ).

На сьогоднішній день велика кількість ПЗ розробляється у вигляді веб-додатків. Ярким прикладом є всесвітньо відомий пакет Microsoft Office, який, ймовірно готується до поступового повного переведення на веб-версію (пакет Office 365 особливо ні в чому не уступає настільній версії, а крім того, містить чимало переваг, пов'язаних із мережним характером продукту). Таким чином, основна направленість такої галузі ІТ, як розробка інтерфейсу користувача, буде орієнтована на веб-додатки, і відповідні технології не

стоять на місці. За останні 10 років надзвичайно сильно підвищилася популярність мови JavaScript, яка тепер застосовується у всіх можливих місцях та застосунках (від традиційної браузерної мови, до обслуговування серверних бек-енд задач в складі Node.Js і навіть використання її нативної версії у вбудовуваних рішеннях). Зокрема, значна кількість досить потужних засобів, таких як цілі бібліотеки та навіть фреймворки для мови JavaScript розроблені саме для створення інтерфейсу користувача веб-додатків. Серед них можна назвати React, Vue.Js, Ember.Js, Backbone.Js, Dojo, частково Angular та багато-багато інших більш-менш обширних засобів розробки. Серед усіх них React займає визначне місце і досить широко використовується на практиці у готових рішеннях. Таким чином, дослідження питань розробки та оптимізації універсальних веб-інтерфейсів з використанням бібліотеки React є надзвичайно **актуальною задачею** сучасної ІТ-галузі.

Відповідно, **метою роботи** можна назвати створення методики розробки та оптимізації універсальних веб-інтерфейсів на основі бібліотеки React.

Для досягнення поставленої мети слід вирішити наступні **задачі дослідження**:

- проаналізувати наявний стан вирішення проблеми створення інтерфейсів користувача для веб-додатків;
- виділити основні (універсальні) вимоги до інтерфейсів користувача веб-додатків;
- розробити алгоритмічні рішення для задоволення виявлених вимог до інтерфейсів веб-додатків;
- здійснити програмну реалізацію конкретного рішення на базі розроблених алгоритмів, методів та моделей;
- виконати тестування та аналіз результатів роботи створеної системи;
- зробити висновки по роботі та установити перспективи її розвитку.

Об’єктом дослідження тут виступає інтерфейс користувача сучасних веб-додатків.

Предметом дослідження є процес зведення універсальних інтерфейсів користувача для веб-додатків.

Наукові результати, що отримано при виконанні даного дослідження полягають у:

- інтегративному аналізі сучасних наукових джерел з питання зведення інтерфейсів користувача веб-додатків засобами JavaScript;

- створенні методики зведення універсальних інтерфейсів користувача веб-додатків на основі використання засобів, що надаються бібліотекою React.

Практична цінність отриманих результатів полягає у тому, що на основі розробленої методики (алгоритмів) можуть ефективно (швидко та у повній мірі) вибудовуватися інтерфейси користувача реальних веб-додатків, що продемонстровано на конкретному простому прикладі, але (зважаючи на універсальність розробки) може легко переноситися на інші програмні рішення.

В цілому робота є завершеним дослідженням з галузі інтерфейсу користувача і вносить, як деякі нові відомості аналітичного характеру, так і синтетичного; результати можуть використовуватися на практиці.

РОЗДІЛ 1. ЗАГАЛЬНА ХАРАКТЕРИСТИКА БІБЛІОТЕКИ REACT, ЇЇ ПРИЗНАЧЕННЯ ТА ОСОБЛИВОСТІ ВИКОРИСТАННЯ

1.1. Аналіз особливостей бібліотеки React мови JavaScript для створення інтерфейсів користувача веб-додатків

Як уже згадувалося вище, для мови JavaScript створено чимало різноманітних бібліотек та цілих фреймворків. Нажаль серед спільноти розробників немає чіткої єдиної думки по позиціонуванню продукту React і за одними оцінками його можна назвати тільки бібліотекою. Така думка обумовлена тим, що використання React дозволяє працювати лише на рівні Представлення (або Виду, View), тобто лише однієї із трьох нерозривних частин шаблону проектування MVC (Model-View-Controller), в той час, як «справжні» фреймворки для мови JavaScript типу Angular дозволяють будувати свої веб-додатки, розробляючи усі три необхідні складові: зовнішній вигляд (тобто Представлення, або Вид), реакцію на дії користувача (тобто Контролер) та реалізацію бізнес-логіки (Модель).

Деякі розробники заявляють про досить зручне поєднання React із іншим засобом для JavaScript розробки – Redux. Ця система призначена для контролю та управління станом веб-додатку, отже у зв’язці з React утворює комплекс, який фактично надає можливість розробки веб-додатків, побудованих по такій популярній на сьогоднішній день технології MVC (а отже, така система може у деякій мірі вважатися фреймворком).

У даній роботі все ж таки будемо притримуватися більш традиційної термінології і називатимемо React бібліотекою (як, зокрема, і написано на сайті її розробників). Визначившись із термінологією, перейдемо до опису її можливостей.

В цілому React надає розробнику мову шаблонів та деякі функції оберненого виклику, які дозволяють продукувати необхідний HTML. Ділянки HTML-коду взагалі є основним результатом роботи React. Так, об’єднання частин HTML та JavaScript, які називають компонентами, можуть

знаходитися в одному із різних своїх допустимих станів, що зберігаються у пам’яті, але незалежно від цього, кінцевим результатом роботи усіх компонентів являється чистий HTML-код.

Далі для React можна виділити три важливих особливості цієї бібліотеки:

а) в першу чергу, розробник завжди може визначитися, як буде виглядати той, чи інший компонент, аналізуючи висхідний текст програми. Якщо відомим є стан – стає зрозумілим результат відрисовки. Не треба відстежувати хід виконання програми (що важливо при розробці складного додатку, тим більше при колективній роботі), достатньо лише знати відповідний стан.

б) у React повсюдно зустрічається «мікс» JavaScript та HTML, що, як мінімум, не зовсім відповідає стандартним загальноприйнятим правилам хорошого тону про розділення логіки та зовнішнього вигляду. Як би це не було дивно, програмістською спільнотою вважається допустимим і, навіть, доцільним об’єднання при використанні бібліотеки React представлення або зовнішнього вигляду, який по суті і визначається HTML-кодом, та функціональності продукту (що, звичайно, створюється активним агентом - мовою програмування JavaScript). Упаковування розмітки та пов’язаної із нею функціональності (поведінки цієї розмітки) у суцільний, окремий компонент виявляється досить прогресивною практикою при використанні React.

в) Обробка методу render(), який є фактично основним при використанні React, і який якраз і продукує згаданий вище HTML, може відбуватися не лише у браузері клієнта, а й виконуватися на сервері. Такий підхід несе як плюси, так і мінуси. При малій та середній інтенсивності запитів на сервер рендеринг саме на ньому сприяє підвищенню продуктивності всього веб-додатка, однак, при наближенні інтенсивності запитів до певного граничного значення (яке, очевидно, залежить від загальної продуктивності серверу) можливе блокування роботи усієї системи,

аналогічне розподіленій атаці на відмову в обслуговуванні (Distributed DoS – DDoS). Для забезпечення максимального ефекту, можливим є внесення динамічного режиму, в якому між рендерингом на стороні клієнту та на сервері відбувається динамічний вибір, залежно від поточного завантаження серверу, що може бути перспективним для оптимізації інтерфейсу і має бути розглянуто у подальшій роботі.

Ще однією цікавою особливістю бібліотеки React (яка уже була згадана непрямим чином) є можливість використання спеціальної мови JSX. Фактично, вона є надбудовою над JavaScript і, на думку розробників, полегшує зведення необхідного коду. В принципі, можливим є використання React і без JSX, що з одного боку логічно (адже не вимагає вивчення нового синтаксису), а з іншого дуже консервативно та може зменшувати загальну продуктивність програміста при користуванні традиційним, менш зручним варіантом запису висхідних кодів, замість запропонованого JSX.

В цілому, бібліотека є досить зручною та надає цікаві можливості по створенню інтерфейсів користувача сучасними, прогресивними засобами.

1.2. Постановка задачі дослідження

Зважаючи на корисні особливості розглядуваної бібліотеки, виникає питання, чи можливо якимсь чином зробити процес зведення інтерфейсу користувача та/або його оптимізацію більш уніфікованим? Відповідно можна сформулювати задачу дослідження:

Дослідити наявні можливості бібліотеки React мови JavaScript по зведенню інтерфейсів користувача для веб-додатків та виконати проектування методики побудови універсальних інтерфейсів веб-додатків, а також розглянути питання оптимізації таких універсальних інтерфейсів.

У якості вхідної інформації використовується:

- документація на бібліотеку React;
- науково-технічні відомості про стандарти зведення інтерфейсу користувача;

- інформація про деякі «середні» потреби сучасних веб-додатків у можливостях інтерфейсу користувача, необхідні для внесення універсальності у розробку.

Вихідною інформацією є:

- методика та опис особливостей зведення інтерфейсу користувача в загальному випадку для «середнього» веб-додатку;

- приклад програмної реалізації інтерфейсу на основі розроблених відомостей універсального плану.

На основі вих. відомостей можна переходити до наступного етапу, а саме критичного аналізу наявних науково-технічних джерел із питання, що розглядається.

РОЗДІЛ 2. АНАЛІТИЧНИЙ ОГЛЯД ІСНУЮЧИХ ДЖЕРЕЛ, ПРИСВЯЧЕНИХ РОЗРОБЦІ ІНТЕРФЕЙСІВ КОРИСТУВАЧА З ВИКОРИСТАННЯМ REACT

Бібліотека React з'явилася у 2013 р., що по міркам галузі технічних наук є досить невеликим терміном, але саме для галузі інформаційних технологій 6 років є величезною відстанню в часі. За даний період з'явилася значна кількість як друкованих, офіційно випущених джерел (в основному – книжок, матеріалів конференцій та форумів, і т.п.), так і «самвидаву» (блогів, особистих конспектів, статей, форумів). Укрупнено усі джерела можна розбити на два класи: друковані (фундаментальних книги із більш-менш повним висвітленням усіх можливостей та особливостей бібліотеки) та електронні (серед них також є більш-менш повні, але більшістю електронні джерела присвячені окремим питанням використання бібліотеки, маловідомим її особливостям, засобам взаємодії з іншими компонентами комплексу веб-розробки, і т.д.). Таким чином, розглянемо обидва типи джерел, розпочинаючи з друкованої літератури.

2.1. Огляд літературних друкованих джерел, присвячених бібліотеці React

Як уже зазначалося вище, незважаючи на молодість бібліотеки, їй присвячено чимало книжок. Так, найбільш повні джерела стосуються не лише бібліотеки React у чистому вигляді, а використання її разом із платформою Redux.

Так, автори чимало уваги приділяють не просто можливостям React, а у досить великій мірі наступним відомостям:

- новим особливостям стандартів ECMAScript (які, як відомо, надзвичайно активно оновлюються після 2015 року, оскільки автори з організації ECMA вирішили видавати пакети оновлення щорічно);

- основам функціонального програмування (зважаючи на досить широке висвітлення даної тематики в рамках книги, до її вивчення можна переходити навіть із мінімальним уявленням у чому воно полягає);

- обробці великих обсягів даних без виконання перезавантаження сторінки.

Звичайно тут також розкриваються широкі можливості і самої бібліотеки. Ключове поняття, на якому базується її використання є «компонент». Під ним мається на увазі окремий будівельний блок, з яких може порівняно легко та швидко вибудовуватися весь інтерфейс веб-додатку в цілому. Компоненти можуть включати інші, простіші компоненти, а також можуть бути частиною компонентів більш високого рівня. Така багаторівнева структура, якраз і дозволяє за допомогою React використовувати переваги швидкої компонентної розробки, аналогічної до технологій типу Windows Forms (для C#) або до візуального інтерфейсу середовища розробки Delphi.

Широта можливостей таких компонентів UI робить можливим за допомогою React впровадження усіх основних принципів побудови ефективного інтерфейсу користувача (і які будуть докладніше розглядатися пізніше).

Особливої уваги дана книга заслуговує через докладне висвітлення питання взаємодії бібліотеки React та іншої сильно пов’язаної із нею бібліотеки Redux. Існують думки, що в комплексі вони можуть розглядатися як повноцінний фреймворк. Дійсно, як відомо, React працює на рівні «Представлення» або «Виду», англ. «View») і не може інкапсулювати у рішення, що розробляється, «бізнес-логіку» (рівень «Моделі»), або функції «Контролеру». Використання інших, сторонніх бібліотек заповнює цей пробіл, про що докладно описується у згаданій книзі.

Робота також розглядає React у комплексі з іншими рішеннями, зокрема із вищезгаданою бібліотекою Redux. Значна увага приділяється JSX, що є засобом, який полегшує генерацію коду, пов’язану із видачею у браузер

великих частин HTML. В цілому, через усю книгу проходить постійне слідування трьом важливим тезам, які стосуються інтерфейсу користувача системи:

- він має бути цікавим (оригінальним) і гарно виглядати (вимога до дизайну);
- швидкість реакції системи на дії користувача має бути якомога більшою;
- інтерфейс має бути гнучким (з контекстними інструментами, редагуванням по місцю, і т.д.).

Також у працях дослідників висвітлюються питання можливостей бібліотеки React у зв'язці з іншими програмними засобами розробки. Зокрема, знову значна увага приділяється взаємодії з бібліотекою Redux, причому інформація викладається на достатньо глибокому рівні проробки. Чимала увага приділена темі тестування Jest, Enzyme, що особливо важливо, оскільки в інших джерелах ця тема майже завжди оминається, не зважаючи на те, що тестування є обов'язковим етапом розробки будь-якого програмного забезпечення, а не лише елементів інтерфейсу користувача.

На елементарному рівні піднімається тема розробки мобільних додатків за допомогою React Native (функціональність рішення, що розглядається у якості прикладу, хоча і є сильно обмеженою, але дозволяє отримати загальне враження про процедуру зведення мобільного додатку у цьому фреймворку).

Ще однією особливістю цієї роботи є включення інформації про SSR (Server Side Rendering), що є досить просунутою можливістю, але потребує використання та знання принципів роботи з Node.js.

В цілому, можна із впевненістю сказати, що, незважаючи на «молодість» бібліотеки React, у широкому продажу доступна значна кількість якісної друкованої літератури по ній.

2.2. Аналіз електронних публікацій про бібліотеку React

Вказані друковані джерела є типовими з точки зору структури та опису особливостей роботи з бібліотекою, однак загальна їх чисельність є на порядки меншою, ніж об’єми публікацій в електронному вигляді, до аналізу яких зараз переходимо.

В першу чергу, існують сторінки-агрегатори, що виводять зведену інформацію про різноманітні джерела, присвячені об’єкту дослідження – рис. 2.1.

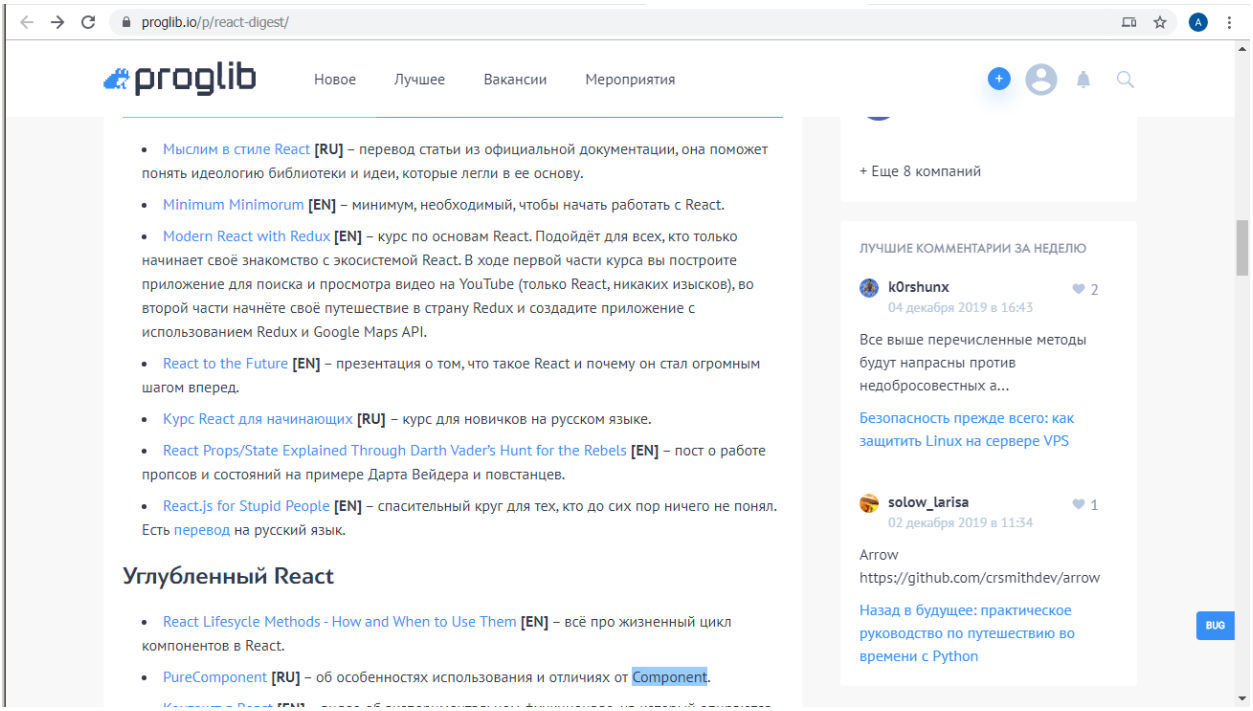


Рис. 2.1. Приклад веб-сторінки зі зведеними ресурсами по вивченню React.

Також уваги заслуговує ресурс, створений на React, із відповідною назвою, та який підійде як повним початківцям, так і розробникам із певним досвідом використання цієї бібліотеки – рис. 2.2.

На цьому ресурсі доступними для вивчення є розділи «Вступ» та «Швидкий старт» (для повних новачків), «Підручник» (для середнього рівня), «Просунутий» (для досвідчених розробників).

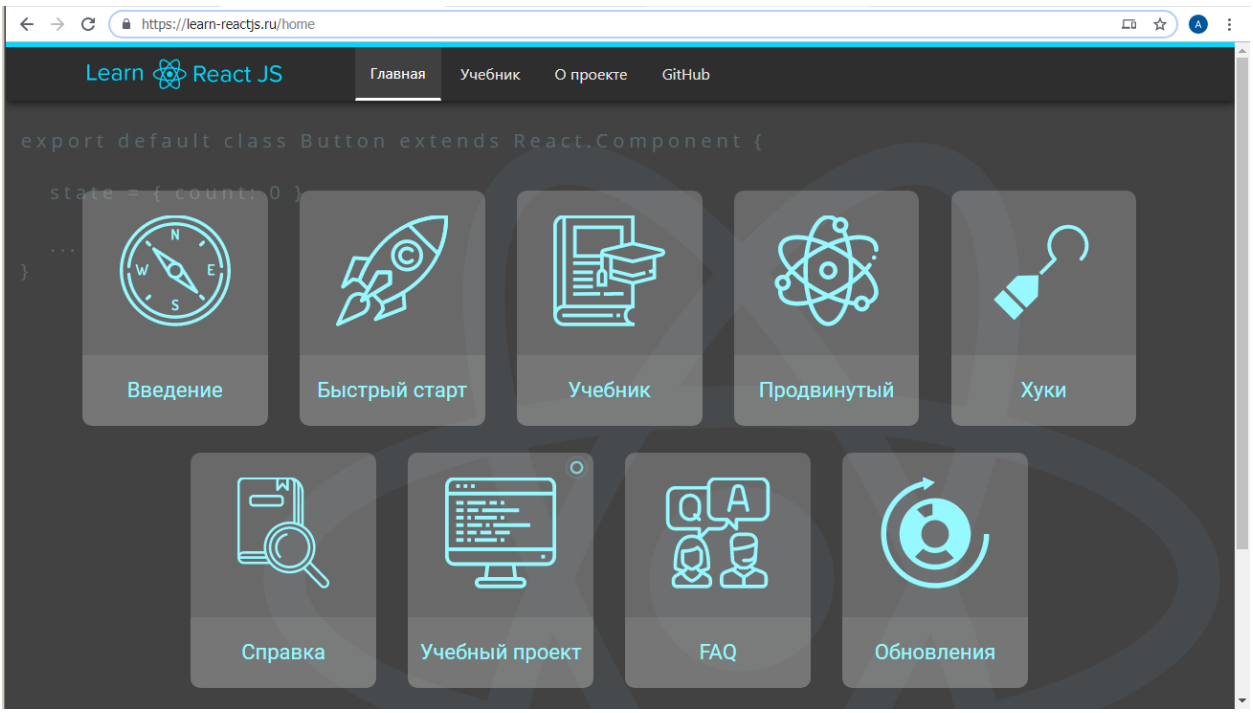


Рис. 2.2. Якісний сайт, присвячений вивченню React і написаний на ньому.

Також тут описується низка доволі специфічних питань, як наприклад «Хуки». Ще пропонується довідкова інформація та приклад виконання повного проекту на React.

Окремі ресурси виділяється тим, що надають будь-якому користувачеві мережі Інтернет відкритого доступу до обширної колекції відео файлів, які в цілому утворюють завершений курс по вивченню бібліотеки React. Доступними тут є теми:

- поняття про компонентний підхід;
- налаштування оточення;
- JSX-синтаксис для роботи в React;
- передача даних до React компонентів;
- поняття стану компонентів;
- Virtual DOM – основний алгоритм React.js;
- відображення масивів, атрибут key;
- оформлення компонентів, робота із CSS;
- життєвий цикл компонентів React;

- оптимізація React-додатків;

- відлагоджування React-додатків;

- зворотний потік даних, підйом стану.

Розширений перелік тем тут наводиться ще й тому, що вони включають у себе усі основні етапи розробки звичайного React-додатку.

Відмітимо, що, зважаючи на дуже вузьку специфіку тематики створення React-додатків, в мережі можна знайти не так багато джерел на дану тематику українською мовою, але усі – з одного серверу.

Окрім більш-менш докладної інформації про систему React, в мережі Інтернет можна також знайти чимало напівпрофесійних статей «для починаючих» про особливості цієї бібліотеки, її можливості та галузь застосування. Цілісної картини вони не надають, але можуть виступати допоміжним засобом на початку вивчення React, або підтягуванні окремих проблемних питань.

Також в Інтернет присутні окремі ресурси, присвячені певним новим особливим функціям бібліотеки (які, можливо, з'явилися поступово, через що їм і присвячувалися саме окремі статті, а не розділи у більш загальних посібниках).

Так, у публікаціях обговорюється питання використання хуків системи React, які були впроваджені у неї лише у 2018 р. (версія 16.8.0). Цей механізм дозволяє реалізовувати функціональні компоненти (які усіма розробниками вважаються більш простими, ніж класи), що мають доступ до стану (state), контексту (context), методам життєвого циклу, посиланням (ref), і т.д. без створення відповідних класів. Використовувати хуки слід лише з останніми версіями React, аби не викликати неочікуваних помилок у висхідному тексті програми.

Наявні у мережі і ресурси, присвячені оптимізації додатків, інтерфейс яких створений за допомогою React. Наприклад, описується підвищення швидкості завантаження ресурсу майже у 5 разів (!), тому вказані у статті підходи безперечно заслуговують на увагу при розробці усіх React проектів.

Деякі електронні ресурси по React мають допоміжний, пізнавальний характер, викладається історія розробки самої бібліотеки, що також є цікавим для розуміння циклу створення програмних продуктів. Так, зазначається, що початком розроблення бібліотеки React став момент, коли фахівцями з Facebook була розроблена технологія Virtual DOM, яка дозволила робити рендеринг непомітним для користувача. Ще однією цінною властивістю React стала ізоморфність, яка, дозволила виконувати один і той же код як на клієнті, так і на сервері, що вирішило проблеми серверного рендеринга.

Компонентний підхід спростив підтримку, тестування і переносимість коду; React набув величезної популярності, з'явилося багато допоміжних бібліотек, велика і доброзичлива спільнота розробників. Завдяки всьому цьому React став одним із лідерів на ринку бібліотек для створення односторінкових додатків, активно розвивається і затребуваною технологією і є одним з кращих рішень для створення інтерфейсів користувача веб-додатків.

Значна кількість публікацій, наявних у мережі, із очевидних причин, виконані англійською мовою, причому деякі з них перекладів не мають, тому також представляють інтерес для осіб, що вивчають можливості бібліотеки, наприклад.

Так, цікавим є підхід, при якому пропонується використання бібліотеки React без однієї з її найважливіших складових – JSX – що являє собою правила запису видачі коду HTML безпосередньо у командах мови JavaScript. Така технологія є доступною тільки за умови попередньої обробки висхідного тексту програми за допомогою програми-транспілятора (або транспайлера), яким, наприклад, є Babel.

Babel дозволяє собі на вхід подавати не лише валідний JavaScript, а і такі висхідні тексти, що містять серйозні відходження від стандартів, точніше від специфікації, яку підтримують більшість браузерів. За своїми внутрішніми алгоритмами Babel перетворює висхідний текст «з

відхиленнями», зокрема із наявним синтаксисом JSX, у повністю робочий і записаний по правилам, що розуміють браузери, JavaScript.

Для прикладу, можна навести два варіанти запису однієї і тієї ж функції: перший варіант із JSX, а другий – на чистому JavaScript:

```
function Button (props) {
  return <button type="submit">{props.label}</button>;
}
ReactDOM.render(<Button label="Save" />, mountNode);
```

```
function Button (props) {
  return React.createElement("button",{type:"submit"},props.label);
}
ReactDOM.render(React.createElement(Button,{label:"Save"}),
mountNode);
```

Очевидно, що нижній варіант набагато менш зручний і значно важче сприймається. Таким чином, використання JSX є повністю обґрунтованим, що однозначно і визнається усією програмістською спільнотою. Більше того, для записів на JSX усіма розробниками робиться величезне допущення (чого майже ніколи не буває в інших випадках), що змішування даних та їх оформлення у даному випадку (при роботі з JSX) є допустимим (!!!). Усе це показує, що синтаксис JSX дійсно є надзвичайно вдалим, тому немає жодного сенсу працювати з React без JSX (ймовірно, значна частка успіху самого React забезпечена тим, що його розробники узяли за основу використання мови JSX).

Слід відмітити, що транспайлер Babel має досить широкі можливості [26] (підтримка обробки JSX є лише однією із багатьох корисних його опцій), направлені в першу чергу на те, щоби код, написаний на останніх версіях мови JavaScript можна було «перекласти» на більш універсальні версії (типу ES5), які розуміє більша кількість браузерів (особливо, не самих популярних, таких, що рідко оновлюються).

Таким чином, як і у попередньому підрозділі, слід констатувати, що існує значна кількість електронних ресурсів, присвячених темі створення

веб-інтерфейсів за допомогою бібліотеки React; браку змістовної інформації не відчувається (за винятком певної недостачі джерел, написаних українською мовою).

РОЗДІЛ 3. МОДЕЛЮВАННЯ ПРОЦЕСІВ РОЗРОБКИ ТА ОПТИМІЗАЦІЇ УНІВЕРСАЛЬНИХ ВЕБ-ІНТЕРФЕЙСІВ

3.1. Моделювання проблемної галузі.

На сьогоднішній день процеси розробки веб-додатків розвиваються семимильними кроками, за місяць з'являються їх сотні і навіть тисячі. Всі вони мають більш, чи менш розвинутий інтерфейс користувача, який, звичайно, залежить від функціональності того, чи іншого додатку. Тим не менше, існують певні загальні правила, яким підкоряється будь-який розробник веб-інтерфейсів, розглянемо їх – рис. 3.1.

Рис. 3.1. Універсальні особливості інтерфейсу користувача сучасних веб-додатків.

Розшифруємо кожен пункт з рис. 3.1 докладніше.

1) **Робота з веб-додатком має бути легкою, інтуїтивною. Контекстні інструменти**.

В першу чергу, робота з будь-яким веб-додатком, як і з настільним додатком, має бути простою, інтуїтивно зрозумілою. В жодному разі процес

звичайного використання веб-додатку не повинен потребувати якихось додаткових знань, чи умінь, окрім тих, що необхідні для усвідомлення його функціональності. Елементи управління програмою мають відповідати одній із двох наступних альтернатив:

- бути традиційними, добре знайомими пересічному користувачеві по роботі з іншими веб-додатками (і дійсно, в реальності вже склалися певні «правила доброго тону» для поведінки окремих елементів управління);

- бути новими (оригінальними, інноваційними або удосконаленими), але із настільки очевидними покращеними властивостями та підходами до оперування ними, що «середня» людина без спеціальної підготовки могла би ефективно працювати із цими засобами інтерфейсу користувача.

Домогтися найбільшої простоти в роботі з додатком можна, наприклад, використовуючи контекстні інструменти, тобто такі, що з’являються, або підлаштовуються залежно від того, що саме зараз робить користувач в системі. Деякі із таких контекстних інструментів стали стандартом де-факто. Так, простими прикладами контекстної поведінки курсору миші, що допомагають користувачеві швидко розпізнавати ті, чи інші особливості контенту, вже давно стали наступні дії:

- при наведенні миші на гіперпосилання, курсор стає рукою із витягнутим вперед вказівним пальцем (CSS властивість cursor:pointer), а сам текст дещо змінює колір (при цьому, зазвичай, задіяні сині, фіолетові, червоні відтінки);

- при наведенні миші на текст, який за бажанням користувача підлягає редагуванню, курсор має із звичайної стрілки перетворюватися на текстову форму (на зразок широкої латинської літери «I», CSS властивість cursor:text);

- при знаходженні миші над елементом, що можна перетаскувати по сторінці, він перетворюється на чотири стрілки, направлених у всі боки (CSS властивість cursor:move);

- тощо.

2) **Прямі дії. Редагування по місцю, перетаскування, безпосереднє виділення**.

Другою властивістю хорошого інтерфейсу користувача є наявність можливості виконання так званих «прямих дій». В першу чергу, це стосується даних (числових, текстових), які повинні мати можливість змінюватися безпосередньо у місці їх відображення на веб-сторінці (пряме редагування або редагування «по місцю»). Також дуже важливою функцією, про яку часто забувають розробники веб-інтерфейсів користувача є перетаскування. Цей надзвичайно простий механізм може бути в той же час достатньо ефективним, якщо:

- дозволити користувачеві його використання;

- зробити це використання широким і ефективним;

- донести до користувача, що в нього є можливість використання перетаскування (можливо, наочно надати приклад такого перетаскування, або зробити це іншим способом).

Нарешті останнім природним елементом прямого маніпулювання є можливість безпосереднього виділення об’єктів: в сучасних веб-інтерфейсах користувач може виділяти все більше і більше об’єктів. Така «свобода» дозволяє наблизити поведінку веб-додатку до стилю поведінки настільного програмного забезпечення. Методами виділення, які можна і треба передбачити на власних веб-сторінках, є:

- виділення за допомогою перемикачів (галочок, позначок);

- виділення простим (однократним) клацанням мишею;

- виділення подвійним клацанням мишею;

- інші, екзотичні способи виділення.

3) **Односторінковий інтерфейс. Усі дії виконуються в межах сторінки.**

Із самого початку використання WWW суттєвим недоліком самого принципу роботи із Всесвітнім павутинням була необхідність постійного перезавантаження веб-сторінок при внесенні до них користувачем бодай

маленької зміни та/або необхідності отримати із сервера як завгодно просту відповідь, хоча би у форматі «так/ні» (тобто перезавантаження сторінки відбувалося навіть при необхідності отримування із серверу одного лише біту інформації!).

Такими діями постійно переривався процес взаємодії користувача із веб-системою, і з кожним завантаженням нова (чи та ж сама, але перезавантажена) сторінка мала знову і знову заволікати користувача до активної взаємодії. Поява технології AJAX (Asynchronous Javascript And Xml) докорінним чином змінила ситуацію, а саме зробила можливим обмін інформацією із сервером уже після завантаження у браузер веб-сторінки, причому це є можливим через будь-який (навіть і дуже довгий) час після завантаження сторінки (кожен AJAX-запит по суті є повноцінним HTTP-запитом, з єдиною різницею, що відповідь сервера браузер вміє тепер направляти в одну і ту ж раніше вже повністю завантажену веб-сторінку).

Із впровадженням технології AJAX задачею програміста стало формування замкненої моделі інформаційних потоків, що не мають виходити за межі веб-сторінки, а єдиним способом її взаємодії із зовнішнім програмним оточенням мають бути AJAX-запити.

Виконання всього веб-додатку у вигляді однієї насиченої сторінки наближає його за своїми властивостями до настільного програмного забезпечення, і може називатися по-різному. Один із варіантів назви для таких додатків – Single Page Application або SPA.

4) Очевидність запрошення (помітність, легкість знаходження опцій додатку).

Часто-густо програмне забезпечення (і не лише веб-версії, а й настільне, мобільне, системне і т.д.) має деяку частину функціональності, що є погано висвітленою в документації та/або погано видимою в інтерфейсі самого ПЗ. Відповідно, досить часто корисні функції програм простоюють, а гроші на розробку цих функцій виявляються витраченими даремно.

Ця ситуація особливо гостро відчувається і конкретно для веб-додатків, адже на веб-сторінці в цілому складніше організувати комплексні елементи управління, яких вистачає для настільних додатків. Програміст повинен реалізувати усі особливості інтерфейсу за допомогою більш-менш стандартних (простих) елементів управління. І тут якраз у нагоді стає бібліотека React, для якої уже існує чимало сторонніх елементів управління досить цікавого виду, але, найголовніше, практично будь-хто може створити власні, як елементи управління, так і цілі компоненти з оригінальним зовнішнім видом та поведінкою.

Таким чином, важливою вимогою до будь-якого компоненту є легкість знаходження усі можливих варіантів роботи із ним, його опцій. Якщо, наприклад, компонент допускає перетаскування, користувачеві про це має відображуватися відповідна інформація, наприклад, курсор миші може змінюватися на 4 стрілки, направлених у всі боки, або руку із витягнутим пальцем. Доцільною є організація спливаючих підказок із пояснювальним текстом, розміщення невеличких міток поруч із елементами для перетаскування, і т.д.

5) Застосування плавних переходів для дій користувача (анімації, затримки, і т.д.).

В процесі досліджень в галузі побудови інтерфейсу користувача встановлено, що при ініціюванні якихось змін у стані (зокрема, якщо це відображується на зовнішньому вигляді) програми, не слід вносити потрібну зміну одномоментно, а треба здійснювати невеликі поступові переходи між станами із візуалізацією цих переходів.

Якщо робити зміни зовнішнього вигляду за одну мить, користувач буде постійно сумніватися, чи були внесені потрібні йому зміни до стану програми; він буде вимушений приглядатися до окремих елементів, що підлягали модифікації та намагатися зрозуміти, чи дійсно із ними відбулися якісь потрібні йому зміни. Така ситуація є неприпустимою, адже введення всього лише коротенького, але помітного оком переходу (якогось спалаху у

комірці з даними, що змінилися, або швидкого переміщення об’єкту за межі екрану, або на якусь нову позицію) дозволяє позбутися усіх сумнівів, а вказаний перехід (підказка) буде сприйматися користувачем як постійне підтвердження своїх дій при виконанні операцій із даними багато разів.

Для підтвердження можуть використовуватися наступні дії:

- короткочасна зміна кольору заднього фону та повернення до початкового, що створює ефект короткого спалаху;

- короткочасне збільшення розмірів об’єкта (наприклад, шрифту);

- короткочасне виділення області додатковою (жирною, кольоровою, пунктирною, і т.п.) рамкою;

- короткочасний зсув об’єкта редагування на декілька пікселів, наприклад, вправо і вгору (ефект вдавлення, натиснення як на звичайну кнопку);

- реальна зміна старого значення на нове, яке тільки що було введено із певною часовою затримкою (реалізація ефекту, що на виконання змін даних потрібні затрати часу, хоча і невеликі);

- інші зміни об’єкта, або параметрів його оточення.

Вказані переходи є обов’язковими для виконання, оскільки у протилежному випадку користувачеві буде незручно працювати із додатком, і, за умови наявної конкуренції (тобто якщо контент, що надається, не є унікальним), великою буде імовірність його уходу із такої веб-сторінки.

6) Миттєва реакція (пошук в реальному часі, авто заповнення, динамічні підказки, і т.д.).

Під миттєвою тут мається на увазі реакція системи лише на саму зміну, без очікування якихось підтверджувальних дій з боку користувача. При цьому реакція має бути видимою, відповідно до умов попереднього пункту, але розпочинатися повинна зразу після внесення користувачем достатніх для хоча б якої-небудь реакції змін у документ.

Наприклад, при здійсненні користувачем пошуку і введенні чергового символу у поле для введення пошукового рядка не слід чекати натиснення на

клавішу Enter, або на кнопку «Знайти», а краще видавати результати пошуку у динамічному режимі – при натисненні чергової клавіші одразу надавати перелік найкращих результатів із можливістю вибору одного з них.

Таким чином, використовуючи вказані правила, можна досягти значно кращого рівня ефективності інтерфейсу користувача веб-додатку. Отже, відповідні правила підлягають впровадженню з використанням бібліотеки React, що буде реалізовано у наступних розділах.

3.2. Вибір методів та засобів для розробки та оптимізації універсальних веб-інтерфейсів з використанням бібліотеки React

Сама бібліотека React написана мовою JavaScript, відповідно, вона і є основним інструментом, що використовуватиметься під час розробки.

Коротко розберемо характеристики цієї мови програмування.

Це спеціалізована мова програмування, яка традиційно використовується для управління об’єктною моделлю документа у браузері під час перегляду сторінок мережі Інтернет (існують продукти, що перетворюють JavaScript на мову загального призначення, команди якої виконуються на сервері, наприклад, Node.JS; але такий підхід сильно протирічить початковій концепції використання цієї мови програмування, і, за умови наявності значної кількості традиційних засобів back-end-розробки перетворення на серверну мову ще й JavaScript значна кількість програмістів вважає абсолютно недоцільною, тому про цей варіант використання говорити більше не будемо).

Особливістю JavaScript є те, що його висхідні програмні коди інтерпретуються, що є досить гнучким, але повільним рішенням.

Оператори у цій, в цілому С-подібній мові, розділяються крапкою з комою. Мова чутлива до регістру, що часто випускають з виду початківці, ймовірно тому, що HTML, застосовуваний зазвичай з JavaScript спільно, не

залежить від регістру (імена тегів і атрибутів HTML можна писати як малими, так і великими літерами).

Однорядковий коментар виділяється символом //, а багаторядковий - парою символів / * і * /, в чому JavaScript знову повторює С.

До даних застосовується слабкий (динамічний) контроль типів. В операторах з різнотипними даними останні автоматично приводяться до необхідного типу. Типи даних можуть бути примітивними і складеними. Примітивні типи містять прості однорідні значення, такі дані можна передавати функціям як параметри за значенням, а не за посиланням. Складені типи містять різнорідні дані (в тому числі і складені), їх передають у функції тільки за посиланням.

Мова JavaScript об’єктно-орієнтована, проте заснована на прототипах, а не на класах. Є чотири типи об’єктів: вбудовані об’єкти, об’єкти браузера, об’єкти документа і об’єкти користувача (програміста).

Введення-виведення в основному обмежене взаємодією з документами і користувачами. За умовчанням передбачається, що доступ до локальної файлової системи заборонений. Однак браузери можуть надавати спеціальні об’єкти, за допомогою яких забезпечується робота з файловою системою користувача, хоча і з видачею попереджень про небезпеку виконання файлових операцій.

Сценарії JavaScript активно взаємодіють з об’єктами, вбудованими в Web-сторінку. Для цього вони, власне, і створюються. Але перш, ніж ця взаємодія стане можливою, слід впровадити код сценарію в текст HTML-документа. Існує кілька способів зв'язати HTML-документ з конкретним сценарієм (скриптом), але зазвичай їх просто розміщують всередині контейнерного тега <SCRIPT>, тобто між дескрипторами <script> і </script>.

Контейнер <SCRIPT> в цьому випадку буде перебувати безпосередньо в HTML-документі, причому у довільному його місці. Програмний код пишуть прямо в HTML-документі або в спеціальних текстових файлах, які можна викликати з головного HTML-документа. Для початку розглянемо

перший варіант. Перш за все браузер знаходить тег <script> в тілі веб-документа, і весь наступний текст намагається обробити як скриптовий код. І так до тих пір, поки не зустріне закриваючий тег </script>. Після цього всі наступні символи будуть вважатися HTML-текстом. Будь-який HTML-документ може містити довільне число «скриптових включень», але кожне має відкриватися і завершуватися відповідним тегом. Від їх розташування у тілі HTML-документа іноді може залежати функціонування всієї Web-сторінки, але про це буде сказано пізніше.

Контейнерний тег <script> може містити атрибут SRC, який вказує ім'я або URL-адресу текстового файлу, що містить код сценарію. Цей атрибут необхідний в тому випадку, якщо сценарій розташований не безпосередньо в HTML-документі, а в окремому файлі. Розширення файлу зі сценарієм може бути яким завгодно, але зазвичай використовують js:

```
<SCRIPT SRC = «myscripts.js»> </ SCRIPT>.
```

Якщо сценарій розташовується в окремому файлі, то в ньому, зрозуміло, теги <SCRIPT> і </ SCRIPT> не пишуть. Сценарій, завантажений з зовнішнього файлу, можна уявити собі просто як його вставку в HTML-документ.

У браузерах, що потенційно підтримують сценарії, цю функцію користувач може відключити (зокрема, із міркувань підвищеної безпеки). Крім того, існують браузери, які принципово не підтримують сценарії. У даній ситуації бажано хоча б вивести повідомлення про те, що на сторінці був сценарій, але в даному конкретному випадку він не виконується. З цією метою в HTML-документі використовують контейнерний тег <noscript>, в якому розміщують текст, який з'явиться користувачеві за умови, що сценарій з тієї, чи іншої причини не виконуватиметься. Всі браузери, які підтримують сценарії, проігнорують вміст тегів <noscript> крім тих випадків, коли підтримка сценаріїв відключена. Браузери, що принципово не підтримують сценарії, навпаки, вміст тега <script> опустять (особливо, якщо він вкладений

у теги <!-- -->, які є HTML-коментарем), а тега <noscript> - відобразять. Приклад наведено нижче:

```
<HTML> <HEAD>
<TITLE> Приклад застосування тега NOSCRIPT </ TITLE>
</ HEAD>
<SCRIPT TYPE = "text / JavaScript»>
<! -
alert ( «Підтримка JavaScript включена»); // ->
</ SCRIPT>
<NOSCRIPT>
<H2> Ваш браузер не підтримує JavaScript або його підтримка відключена </ H2>
</ NOSCRIPT>
</ HTML>
```

Тут був використана функція alert (повідомлення) для виведення повідомлень в маленькому діалоговому вікні.

Як уже зазначалося, в окремих файлах зазвичай розміщують бібліотеки функцій (визначення функцій), а також сценарії, які використовуються в декількох HTML-документах одного або декількох сайтів. Сценарій можна також писати у вигляді рядка операторів, між якими ставиться крапка з комою. Такий рядок, взятий в лапки, служить у якості значення атрибута-події, наприклад:

```
<IMG SRC = "mypicture.gif” ONCLICK = “alert ('Привіт!')”.
```

Виклики функцій і їх визначення можуть слідувати в довільному порядку, але тільки якщо вони розташовані в одному і тому ж контейнері <script>. Якщо вони розміщуються у різних контейнерах <script>, то необхідно, щоб визначення функції передувало її виклику. Аналогічно, якщо визначення функцій знаходяться в окремому файлі, то його необхідно завантажити в HTML-документ раніше викликів цих функцій.

При спробі завантажити і виконати неправильний варіант HTML-коду з'явиться діалогове вікно з повідомленням «Помилка: Передбачається наявність об’єкта». Браузер інтерпретує теги HTML послідовно. Так, зустрівши вираз виклику функції myfunc() в одному контейнері <script>, браузер ще не має в пам'яті визначення цього об'єкта (функції), розташованого в іншому контейнері <script>. Якщо ж визначення функції і її

виклик знаходяться в одному і тому ж контейнері <script>, то його вміст спочатку завантажується в пам'ять, а потім аналізується. Коли інтерпретатор зустрічає виклик функції, то він шукає її визначення в пам'яті і в разі успіху виконує код цієї функції.

Якщо необхідно, щоб сценарій виявився в браузері перш, ніж буде завантажено елементи HTML-документа, то його слід розташувати у верхній частині HTML-коду, а ще краще в контейнері <head> в заголовку документа.

Такими є базові властивості мови JavaScript, що є основою для побудови універсальних інтерфейсів користувача за допомогою бібліотеки React, яка, власне, і написана мовою JavaScript.

Окрім мови програмування (інтерпретатор якої фактично вбудований у кожний браузер), необхідною умовою для роботи з самою бібліотекою є виконання правильного налаштування програмного оточення. Також для швидкої розробки окремих універсальних компонентів ефективних веб-інтерфейсів можна використати онлайн-редактори, серед яких є досить якісні рішення.

Наприклад, на сторінці https://codepen.io/pen/ – рис. 3.2 – на вільній основі пропонується доступ до налаштованого середовища, що пропонує користувачеві три вікна:

- для введення тіла веб-сторінки (на HTML);
- для введення елементів каскадних таблиць стилів (CSS);
- для розробки власне програмної складової мовою JavaScript.

Цей програмний засіб та інші аналогічні готові онлайн рішення є надзвичайно зручними з позиції швидкої розробки програмного забезпечення на JavaScript.

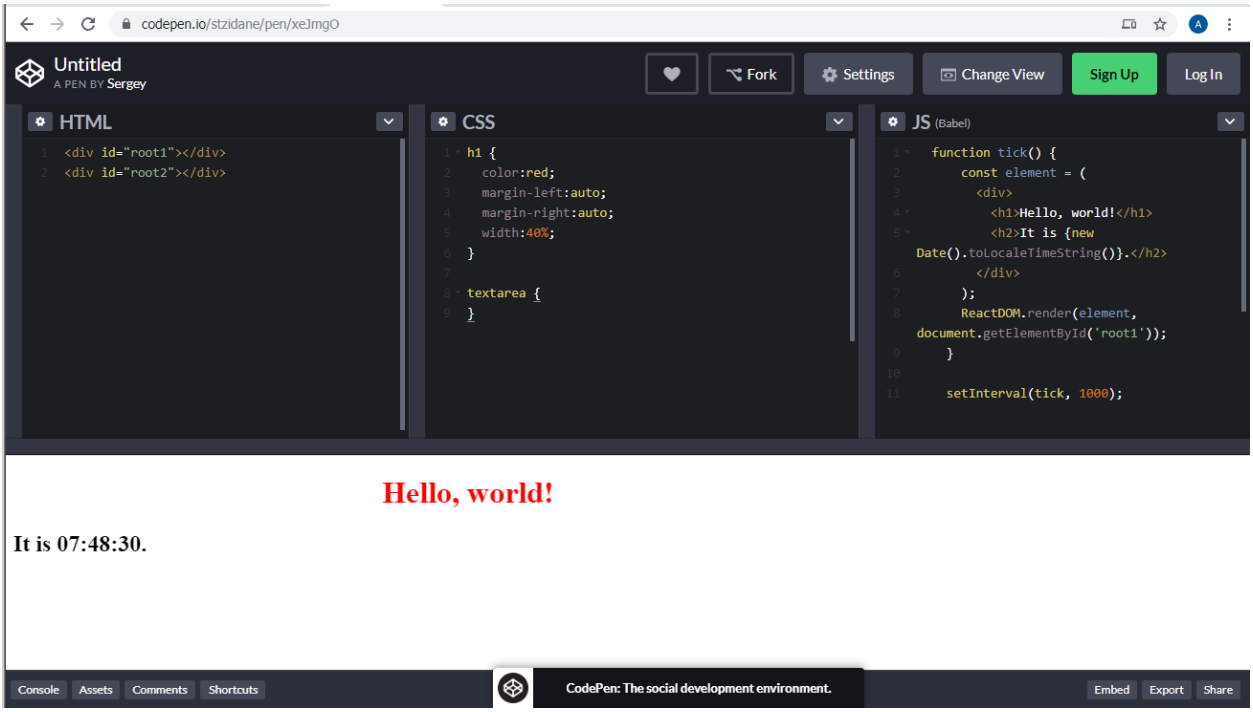


Рис. 3.2. Приклад вікна онлайн засобу CODEPEN, зручного для швидкої розробки програмних рішень на React (і взагалі – для Babel).

3.3. Моделювання інформаційних потоків системи та розробка її схематичного представлення

Інформаційні потоки системи можуть бути представлені схемою – як на рис. 3.3, а розшифровані у наступному переліку:

1.1. Формування веб-сторінки на основі прописаних у її тіло джерел даних (статично вбудованих у веб-сторінку, або зчитаних із бази даних, і т.д.).

1.2. Відображення користувачеві веб-сторінки, а точніше, наявної на ній інформації – за допомогою спеціально розроблених універсальних компонентів бібліотеки React. Також відображення користувачеві підказок (або запрошень) щодо можливості редагування показаної інформації безпосередньо в тому місці, де вона розміщена.

2.1. Ініціація користувачем режиму редагування одного із відображених полів, і отримання веб-сторінкою від користувача нового значення для даного параметру, що редагується.

Рис. 3.3. Схема інформаційних потоків системи, що проектується.

2.2. Виконання AJAX-запиту до серверу з оновленням відповідного значення у базі даних.

3.1. Інформація від серверу до веб-сторінки про успішність останньої операції зміни даних.

3.2. При неуспішному виконанні процесу збереження нового значення для параметру, що змінювався, веб-сторінка має проінформувати про це користувача.

Базуючись на цих інформаційних потоках, можна переходити до стадії проектування алгоритму роботи універсальних інтерфейсів, що виконується у наступному підрозділі.

3.4. Розробка та візуалізація алгоритмів розробки та оптимізації універсальних веб-інтерфейсів

Базуючись на схемі наявних інформаційних потоків, що можуть існувати у загальному випадку системи універсального веб-інтерфейсу, можна сформулювати наступний алгоритм роботи системи – рис. 3.4:

1) формування тіла веб-сторінки, яке включає велику кількість фактичних даних у текстовій (або числовій) формі, інкапсульованих в компоненти IntTextarea та видача її у браузер;

2) відображення сторінки користувачеві та початок її перегляду ним;

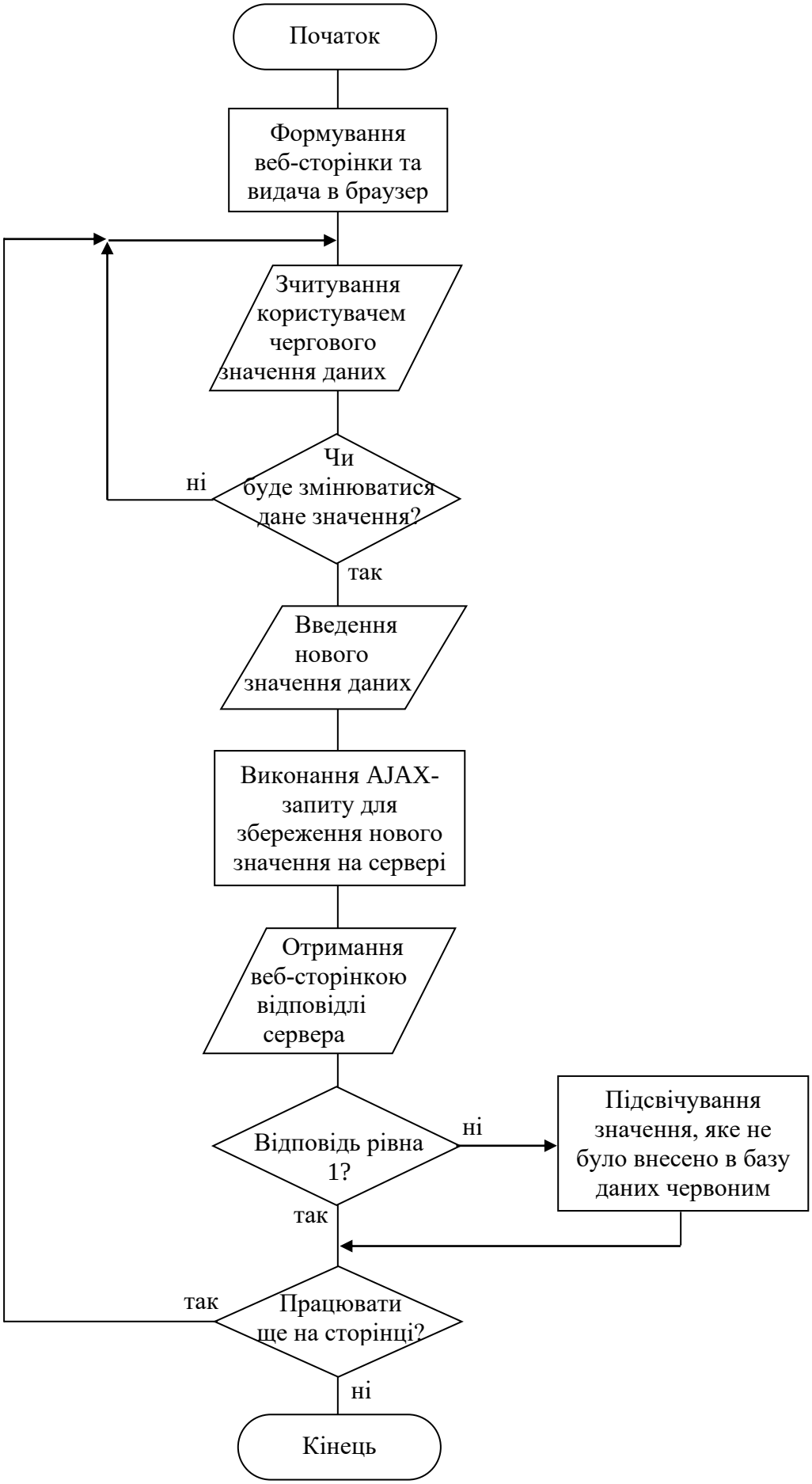


Рис. 3.4. Блок-схема алгоритму роботи спроектованої системи.

3) перегляд користувачем чергового фактичного значення, вбудованого у веб-сторінку;

4) якщо користувач вирішує змінити прочитане значення, він здійснює клікання мишею на відповідному компоненті IntTextarea бібліотеки React та переходить до режиму редагування, інакше переходить до пункту 3;

5) користувач вводить нове значення та завершує процес редагування кліканням мишею за межами текстового поля (саме поле при цьому втрачає фокус введення);

6) розпочинається AJAX-запит, у якому на сервер передається по протоколу HTTP (POST-запитом) два значення: ідентифікатор текстового поля, яке було змінено, та нове значення, введене користувачем;

7) якщо серверу вдається успішно зберегти дані, він передає у веб-сторінку 1, інакше передається 0;

8) якщо у відповідь на AJAX-запит було передано 0, відповідне поле підсвічується червоним (це означає, що дані не було збережено і пізніше слід повторити спробу), інакше реакції немає (нормальний хід процесу збереження);

9) якщо користувач ще буде працювати із веб-сторінкою, то перехід до пункту 3;

10) кінець.

На основі інформації з рис. 3.4 та наведеного текстового опису, можна переходити до програмної реалізації спроектованого рішення.

РОЗДІЛ 4. ОСОБЛИВОСТІ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СТВОРЕНИХ АЛГОРИТМІВ РОЗРОБКИ ТА ОПТИМІЗАЦІЇ УНІВЕРСАЛЬНИХ ВЕБ-ІНТЕРФЕЙСІВ

4.1. Опис загальної структури програмного рішення

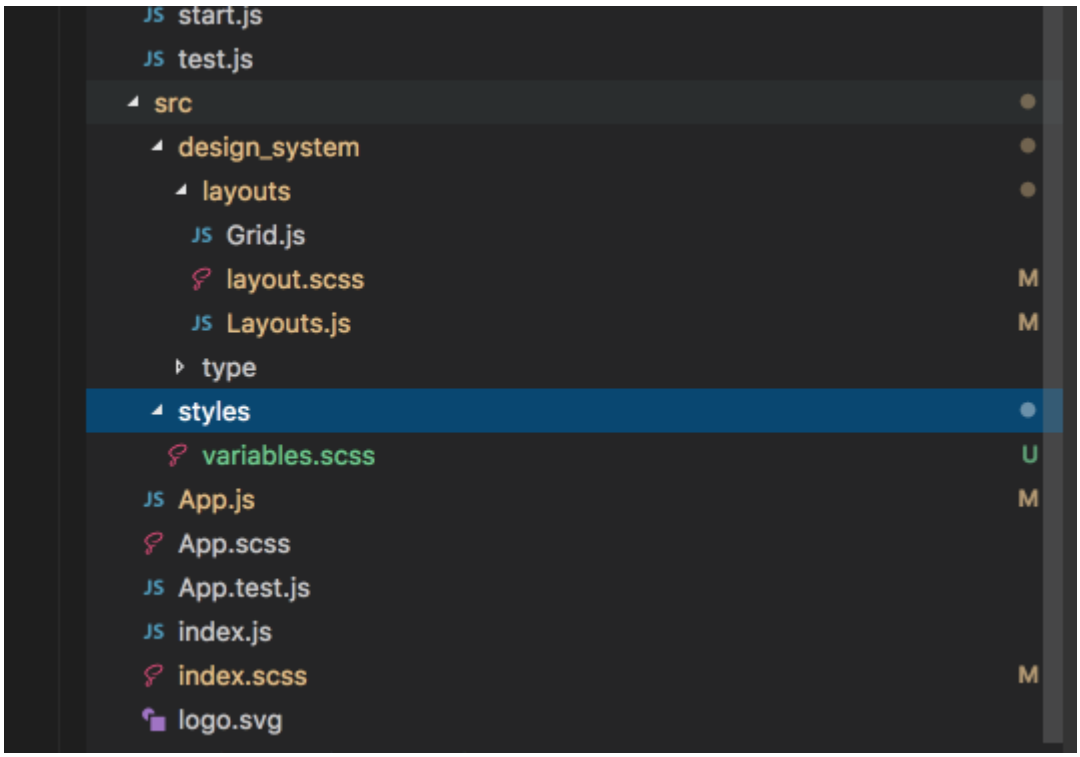
React зробив дуже багато для спрощення веб-розробки. Його компонентна архітектура принципово спрощує завдання розкладання і повторного використання коду. Однак, розробникам не завжди ясно, як використовувати одні і ті ж компоненти в різних проектах. Розглянемо тут як цю ситуацію можна виправити.

React спростив написання гарного, виразного коду. Однак без чітких шаблонів повторного використання компонентів, код з плином часу починає розгалужуватися і стає дуже важко його підтримувати. Існують проекти, де один і той же елемент призначеного для користувача інтерфейсу має до десяти різних реалізацій. Іншою проблемою є те, що, дуже часто розробники пов'язують призначений для користувача інтерфейс і бізнес-функціональність занадто щільно, що призводить до складнощів у майбутньому, коли інтерфейс змінюється.

У якості прикладу розглянемо, як можна створити спільно використовувані компоненти користувальницького інтерфейсу і як виробити єдину мову розробки для кожного кінцевого застосування.

По-перше, слід створити порожній проект React. Найшвидший спосіб його створити - це через команду create-react-app, але потрібно трохи постаратися, щоб налаштувати для цього Sass.

Всі візуальні компоненти будуть перебувати в папці design_system поряд з відповідними стилями. Всі глобальні стилі або змінні будуть розташовуватися в src / styles.



Однією з цілей створення бібліотеки призначеного для користувача інтерфейсу є поліпшення відносин між командами дизайну і розробки. Front-end розробники вже певний час співпрацюють з розробниками API і тепер здатні непогано виробляти API угоди. Але цей момент часто вислизає при координації з командою дизайнерів. Якщо задуматися, то існує тільки певне число станів, в яких може існувати елемент призначеного для користувача інтерфейсу. Якщо нам потрібно створити компонент заголовка, наприклад, то це буде елемент від h1 до h6, який може бути напівжирним, курсивним або підкресленим. Це повинно бути нескладно описати в коді.

Перше, що потрібно зробити, перш ніж приступати до створення будь-якого дизайн проекту - це визначитися зі структурою grid сітки. Для багатьох додатків вона носить випадковий характер. І це призводить до того, що немає чіткої системи розташування елементів, що викликає труднощі у розробників при визначенні яку систему використовувати. Таким чином слід вибрати конкретну систему. Дуже часто на практиці використовується система grid розмітки 4px - 8px, з якою легко спростити вирішення багатьох питань з оформленням.

Почнемо з того, що створимо базову систему grid розмітки в коді. Розпочнемо з компонента додатка, який визначає макет.

```
//src/App.js

import React, { Component } from 'react';
import logo from './logo.svg';
import './App.scss';
import { Flex, Page, Box, BoxStyle } from './design_system/layouts/Layouts';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Build a design system with React</h1>
        </header>
        <Page>
          <Flex lastElRight={true}>
            <Box boxStyle={BoxStyle.doubleSpace} >
              A simple flexbox
            </Box>
            <Box boxStyle={BoxStyle.doubleSpace} >Middle</Box>
            <Box fullWidth={false}>and this goes to the right</Box>
          </Flex>
        </Page>
      </div>
    );
  }
}

export default App;
```

Далі визначаємо кілька компонентів для стилів і блоків контенту.

```
//design-system/layouts/Layout.js
import React from 'react';
import './layout.scss';

export const BoxBorderStyle = {
    default: 'ds-box-border--default',
    light: 'ds-box-border--light',
    thick: 'ds-box-border--thick',
}

export const BoxStyle = {
    default: 'ds-box--default',
    doubleSpace: 'ds-box--double-space',
    noSpace: 'ds-box--no-space'
}

export const Page = ({children, fullWidth=true}) => {
    const classNames = `ds-page ${fullWidth ? 'ds-page--fullwidth' : ''}`;
    return (<div className={classNames}>
        {children}
    </div>);

};

export const Flex = ({ children, lastElRight}) => {
```

```
    const classNames = `flex ${lastElRight ? 'flex-align-right' : ''}`;
    return (<div className={classNames}>
        {children}
    </div>);
};

export const Box = ({
    children, borderStyle=BoxBorderStyle.default, boxStyle=BoxStyle.default,
fullWidth=true}) => {
    const classNames = `ds-box ${borderStyle} ${boxStyle} ${fullWidth ? 'ds-
box--fullwidth' : ''}` ;
    return (<div className={classNames}>
        {children}
    </div>);
};
```

Нарешті створюємо CSS стилі в SCSS.

```
/*design-system/layouts/layout.scss */
@import '../../styles/variables.scss';
$base-padding: $base-px * 2;

.flex {
    display: flex;
    &.flex-align-right > div:last-child {
        margin-left: auto;
    }
}

.ds-page {
    border: 0px solid #333;
    border-left-width: 1px;
    border-right-width: 1px;
    &:not(.ds-page--fullwidth){
        margin: 0 auto;
        max-width: 960px;
    }
    &.ds-page--fullwidth {
        max-width: 100%;
        margin: 0 $base-px * 10;
    }
}

.ds-box {
    border-color: #f9f9f9;
    border-style: solid;
    text-align: left;
    &.ds-box--fullwidth {
        width: 100%;
    }

    &.ds-box-border--light {
        border: 1px;
    }
    &.ds-box-border--thick {
        border-width: $base-px;
    }

    &.ds-box--default {
        padding: $base-padding;
    }
```

```
    &.ds-box--double-space {
        padding: $base-padding * 2;
    }

    &.ds-box--default--no-space {
        padding: 0;
    }
}
```

Тут variables.scss - це файл, в якому задаємо глобальні значення, такі як колір і настройка розмітки. Оскільки використовується сітка 4px-8px, базова величина буде 4px. Батьківський компонент - Page, і він контролює розмітку сторінки. Потім елемент нижнього рівня - це Box, який визначає, як вміст відображається на сторінці. Це просто div, який знає, як буде відображатися належним чином в певному контексті.

Тепер потрібен компонент Container, який об'єднує кілька divов. Обираємо flex-box, звідки витікає і назва компонента Flex.

Система типографії є найважливішим компонентом будь-якої програми. Зазвичай визначаються основні значення через глобальні стилі і перевизначаються, коли це необхідно. Це часто призводить до непослідовності в дизайні. Насправді можна легко вирішити цю проблему шляхом розширення бібліотеки дизайну.

По-перше слід визначити деякі стильові константи та створити клас оболонки.

```
// design-system/type/Type.js
import React, { Component } from 'react';
import './type.scss';

export const TextSize = {
    default: 'ds-text-size--default',
    sm: 'ds-text-size--sm',
    lg: 'ds-text-size--lg'
};

export const TextBold = {
    default: 'ds-text--default',
    semibold: 'ds-text--semibold',
    bold: 'ds-text--bold'
};

export const Type = ({tag='span', size=TextSize.default,
boldness=TextBold.default, children}) => {
    const Tag = `${tag}`;
    const classNames = `ds-text ${size} ${boldness}`;
    return <Tag className={classNames}>
        {children}
```

```
    </Tag>
};
```

Далі слід визначити стилі CSS, які будуть використовуватися для текстових елементів.

```
/* design-system/type/type.scss*/

@import '../../styles/variables.scss';
$base-font: $base-px * 4;

.ds-text {
    line-height: 1.8em;

    &.ds-text-size--default {
        font-size: $base-font;
    }
    &.ds-text-size--sm {
        font-size: $base-font - $base-px;
    }
    &.ds-text-size--lg {
        font-size: $base-font + $base-px;
    }
    &strong, &.ds-text--semibold {
        font-weight: 600;
    }
    &.ds-text--bold {
        font-weight: 700;
    }
}
```

Це простий компонент Text, що визначає різні UI стану, в яких може перебувати текст. Можливо в подальшому розширювати його, доповнюючи можливості обробляти мікро взаємодії, як наприклад відображення підказок при обрізанні тексту, або відтворення іншого елемента в особливих випадках, наприклад при виведенні електронної пошти, часу і т.д.

Поки що розглядалися тільки самі основні елементи, які можуть існувати в веб-додатку, але вони не використовуються самі по собі. Можна розширити даний приклад і створити просте модальне вікно.

По-перше визначається клас компонента для модального вікна.

```
// design-system/Portal.js
import React, {Component} from 'react';
import ReactDOM from 'react-dom';
import {Box, Flex} from './layouts/Layouts';
import { Type, TextSize, TextAlign} from './type/Type';
import './portal.scss';

export class Portal extends React.Component {
    constructor(props) {
        super(props);
```

```
        this.el = document.createElement('div');
    }

    componentDidMount() {
        this.props.root.appendChild(this.el);
    }

    componentWillUnmount() {
        this.props.root.removeChild(this.el);
    }

    render() {
        return ReactDOM.createPortal(
            this.props.children,
            this.el,
        );
    }
}


export const Modal = ({ children, root, closeModal, header}) => {
    return <Portal root={root} className="ds-modal">
        <div className="modal-wrapper">
        <Box>
            <Type tagName="h6" size={TextSize.lg}>{header}</Type>
            <Type className="close" onClick={closeModal}
align={TextAlign.right}>x</Type>
        </Box>
        <Box>
            {children}
        </Box>
        </div>
    </Portal>
}
```

Далі можна визначити стилі CSS для модального вікна.

```
#modal-root {
    .modal-wrapper {
        background-color: white;
        border-radius: 10px;
        max-height: calc(100% - 100px);
        max-width: 560px;
        width: 100%;
        top: 35%;
        left: 35%;
        right: auto;
        bottom: auto;
        z-index: 990;
        position: absolute;
    }
    > div {
        background-color: transparentize(black, .5);
        position: absolute;
        z-index: 980;
        top: 0;
        right: 0;
        left: 0;
        bottom: 0;
    }
    .close {
        cursor: pointer;
```

```
    }
}
```

Відмітимо, що createPortal дуже схожий на метод render, за винятком того, що він створює дочірні елементи в осередку, яка існує за межами DOM ієрархії батьківського компонента. Такий функціонал був доданий в React 16.

Тепер, коли визначено компонент, слід розглянути, як можна використовувати його в реальній ситуації.

```
//src/App.js

import React, { Component } from 'react';
//...
import { Type, TextBold, TextSize } from './design_system/type/Type';
import { Modal } from './design_system/Portal';

class App extends Component {
  constructor() {
    super();
    this.state = {showModal: false}
  }

  toggleModal() {
    this.setState({ showModal: !this.state.showModal });
  }

  render() {

          //...
          <button onClick={this.toggleModal.bind(this)}>
            Show Alert
          </button>
          {this.state.showModal &&
            <Modal root={document.getElementById("modal-root")} header="Test
Modal" closeModal={this.toggleModal.bind(this)}>
            Test rendering
          </Modal>}
            //....
    }
}
```

Можливо використовувати модальне вікно всюди і підтримувати його стан в об'єкті, що викликає. Але при такому підході виникає помилка, що не працює кнопка закриття. Це тому, що було створено всі компоненти у вигляді закритої системи. Він просто бере ті параметри, які йому потрібні, і ігнорує все інше. В даному контексті, компонент тексту ігнорує обробник події onClick. Вказану проблему порівняно легко можна виправити.

```
/ In  design-system/type/Type.js

 export const Type = ({ tag = 'span', size= TextSize.default, boldness =
TextBold.default, children, className='', align=TextAlign.default, ...rest})
=> {
    const Tag = `${tag}`;
    const classNames = `ds-text ${size} ${boldness} ${align} ${className}`;
    return <Tag className={classNames} {...rest}>
        {children}
    </Tag>
};
```

ES6 надає зручний спосіб для вилучення решти параметрів у вигляді масиву, що і слід застосувати, передаючи параметри компоненту.

Розроблена система є універсальною і може слугувати прикладом для побудови компонентів більш спеціалізованої дії, що мають більш точну, необхідну саме у даній, кожній конкретній роботі функціональність.

4.2. Конкретизація особливостей інтерфейсу користувача системи

Для рішення по створенню універсальних компонентів, що спроектоване у розділі 3, розглянемо особливості його інтерфейсу користувача, іншими словами – особливості зовнішнього вигляду.

Коли інформаційні дані відображуються користувачеві у звичайному режимі, вони виглядають як звичайний текст, але, відповідно до пункту 4 підрозділу 3.1 («очевидність запрошень») мають спеціальну позначку у вигляді двох коротких ліній, нахилених під кутом 45 градусів і розміщених справа внизу від значення, яке можна піддати редагуванню – рис. 4.1.

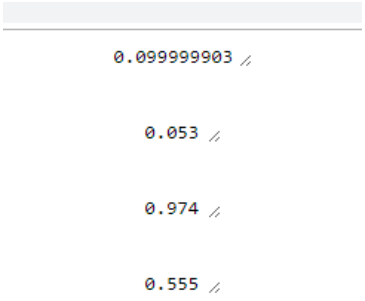


Рис. 4.1. Мітки справа-знизу від даних підказують, що їх можна редагувати.

Після однократного клацання мишею по числу, що має таку мітку, воно переходить у режим редагування – рис. 4.2.

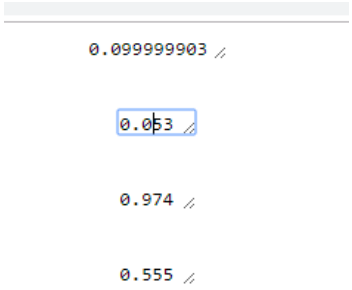


Рис. 4.2. Режим редагування інформативних значень «на місці».

Якщо кількість цифр змінюється, компонент автоматично змінює свій розмір, для того, щоб підказувальна мітка завжди знаходилася зразу справа від значення, яке може редагуватися - рис. 4.3.

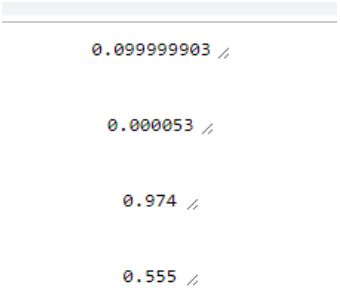


Рис. 4.3. Автоматична зміна розміру компонента при зміні кількості розрядів у введеному значенні (нове значення 0,000053 має на 3 розряди більше, ніж попереднє 0,053, тому ширина компоненту автоматично збільшилася на 3 символи).

Якщо після редагування нове значення було успішно внесено на сервері в базу даних, то відповідне поле (як і усі інші) відображається звичайним кольором (так показано на рис. 4.3 після редагування другого згори поля).

Якщо нове значення не було успішно збережено на сервері, то відповідне поле підсвічується червоним – рис. 4.4.

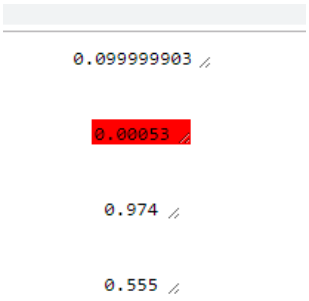


Рис. 4.4. Зовнішній вигляд компоненту у випадку, коли нові дані не вдалося зберегти на сервері.

Підсумовуючи, можна сказати, що спроектований компонент має зручний інтерфейс, захист від втрати даних, зручність знаходження можливості редагування, має можливість обміну даними із сервером без перезавантаження сторінки, одним словом, повністю відповідає вимогам до веб-інтерфейсів, сформованим у підрозділі 3.1.

4.3. Особливості реалізації у висхідних кодах спроектованих алгоритмів

Як уже було встановлено вище, уся програмна реалізація виконується мовою JavaScript. Традиційним підходом при цьому є створення обробника події повного завантаження для всього вікна браузера:

```
window.addEventListener('load',sethandlers,false);
```

Функція sethandlers являє собою підпрограму, у якій встановлюються обробники для усіх елементів сторінки:

```
function sethandlers()
{
for(let i=1;i<=N;i++)
   document.getElementById("text"+i).addEventListener('blur',
commit);
}
```

Для усіх спроектованих елементів встановлюється спільний обробник – функція commit(), яка і виконує усю змістовну роботу по забезпеченню логіки роботи компонента:

```
 function commit(e)
 {
    e.srcElement.cols=e.srcElement.value.length;
    const data=new FormData();
    data.append('changedobject',e.srcElement.id);
    data.append('newvalue',e.srcElement.value);
    const req=new XMLHttpRequest();
    req.addEventListener('load',getsaved,false);
    req.open("POST","processnewdata.php",true);
    req.send(data);
 }
```

У тілі цієї функції виконується підгонка розміру компонента під нові введені дані, а також виконання AJAX-запиту для збереження нових даних на сервері.

Відповідь від сервера на AJAX-запит обробляє функція `getsaved()`.

```
function getsaved(e)
{
    let resp=+e.target.responseText;
    if(resp<0)
       document.getElementById('text'+
Math.abs(resp)).style.backgroundColor="red";
    else
       document.getElementById('text'+
Math.abs(resp)).style.backgroundColor="white";
}
```

Головну ж частину програмної реалізації складають функціональні компоненти системи React:

а) базовий компонент

```
function IntTextarea(props)
{
    return <form>
```

```
    <textarea id={props.item_name} rows="1" cols="5">
{props.item_value}</textarea>
    </form>;
}
```

б) компонент, що агрегує потрібну кількість компонентів IntTextarea:

```
function App()
{
    return (
      <div>
       <IntTextarea item_name="text1" item_value="0.003" /><br/>
       <IntTextarea item_name="text2" item_value="0.653" /><br/>
       <IntTextarea item_name="text3" item_value="0.974" /><br/>
       <IntTextarea item_name="text4" item_value="0.555" /><br/>
       <IntTextarea item_name="text5" item_value="0.029" /><br/>
       <IntTextarea item_name="text6" item_value="0.005" /><br/>
       <IntTextarea item_name="text7" item_value="0.003" /><br/>
       <IntTextarea item_name="text8" item_value="0.323" /><br/>
       <IntTextarea item_name="text9" item_value="0.003" /><br/>
      <IntTextarea item_name="text10" item_value="0.303" /><br/>

      </div>
    );
  }
```

Крім коду на JavaScript проект також включає код HTML та CSS.

РОЗДІЛ 5. ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ СПРОЕКТОВАНОЇ СИСТЕМИ

5.1. Інструкція адміністратора системи

У попередніх розділах спроектовано та виконано програмну реалізацію компоненту `IntTextarea`, який призначений для відображення на веб-сторінці даних, що можуть редагуватися. Для використання компоненту у JSX-коді, що передається методу ReactDOM.render(), слід підключити його відповідним тегом:

```
<IntTextarea item_name="text1" item_value="0.003" /><br/>
```

Тег має два атрибути:

а) `item_name` – ідентифікатор даного текстового елементу, по цьому виразу інформація також ідентифікується і на сервері у базі даних;

б) `item_value` – початкове значення, якому відповідає ідентифікатор `item_name`.

Позиціонування цього компоненту зручно (як і для інших елементів веб-сторінки) виконувати, розміщуючи їх у шарі … , або інших блочних елементах.

5.2. Розробка керівництва користувача створеного програмного продукту

Користувачі, які працюватимуть із веб-додатками, що активно використовуватимуть розроблений компонент `IntTextarea`, повинні дотримуватися наступної інструкції:

а) При перегляді веб-сторінки у місцях розміщення фактичних даних (зокрема числових, хоча це і не є обмеженням, відомості можуть бути текстовими) потрібно слідкувати за наявністю спеціальної мітки справа внизу від усіх таких значень;

б) При наявності такої мітки, значення підлягає редагуванню, для чого слід клацнути один раз мишею на самих даних;

в) коли компонент перейде в режим редагування (з’являється рамка та курсор, що блимає), слід ввести нове значення для цього поля;

г) коли нове значення введено, слід вийти із режиму редагування, для чого натиснути мишею на вільному місці веб-сторінки;

д) впевнитися, що після втрати фокусу, число залишилося нормального кольору, і не стало червоним (тобто нове значення було успішно збережено на сервері);

е) якщо значення стало червоним, то слід через деякий час знову клацнути це поле, щоби воно отримало фокус, а після цього зразу клацнути десь в іншій частині сторінки, щоб фокус із текстового поля зник: якщо число стало нормального кольору проблема вирішена, а якщо залишилося червоним – повторити цей пункт через деякий час знову;

є) якщо потрібно змінювати інші поля, то перейти до пункту б).

5.3. Опис вимог до апаратного забезпечення

Розроблене програмне забезпечення може використовуватися на будь-якому ПК, оснащеному сучасним браузером, який підтримує хоча би стандарт мови JavaScript ES5. В іншому випадку слід оновити браузер, і, за необхідності, інше програмне забезпечення (типу операційної системи).

Коротко опишемо середні вимоги до апаратної частини ПК, на якому у браузері можуть комфортно запускатися веб-додатки із розробленим компонентом. Це має бути персональний комп'ютер, що відповідає наступним мінімальним системним вимогам:

- Процесор – 1,6 ГГц;
- ОЗУ – 1 Гб;
- відеоадаптер, сумісний з Directx 9.0;
- вільне місце на диску - 100 Мб;
- клавіатура;
- операційна система – Windows 7.

5.4. Тестування системи та опис результатів її роботи

По результатам виконання роботи проведено тестування компонентів, яке показало адекватність їх роботи спроектованому алгоритму (рис. 3.4).

Для тестування створено просте серверне програмне забезпечення (мовою PHP):

```
<?php
header('Content-Type: text/html; charset=UTF-8');
$hostname="localhost"; $username="root"; $password="";
$dbName="datafromreact";
$link=mysql_connect($hostname,$username,$password) or die("No co
nnect to DB!");
mysql_set_charset('utf8',$link);
mysql_select_db("$dbName",$link) or die("Cannot select DB!");

$query="UPDATE tablewithvalues SET item_value=\"".$_POST["newval
ue"]."\" WHERE item_name=\"".$_POST["changedobject"]."\";";
$result=mysql_query($query);
$num=+substr($_POST["changedobject"],4);
if(mysql_affected_rows($link)>0)
    print $num;
else
    print "-".$num;
?>
```

Головна мета програмного продукту – надання відповідей на AJAX-запити та внесення оновлених значень до бази даних.

На сервері створена проста база даних з однією таблицею, яка призначена виключно для тестування (в реальних проектах серверна частина може реалізовуватися різноманітними способами: мовами програмування, СУБД, і т.д.) – рис. 5.1.

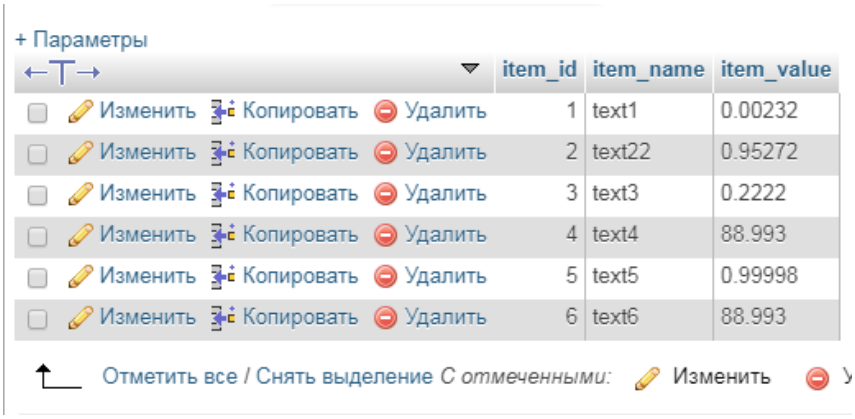


Рис. 5.1. Проста база даних із однієї таблиці, створена для тестування спроектованої системи.

Проведені тести показали повну адекватність роботи розроблених компонентів їх алгоритмам та технічному завданню.

РОЗДІЛ 6. ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО РІШЕННЯ

Будь-яка інженерна (науково-технічна) розробка має виконуватися тільки за умови її техніко-економічного обґрунтування (ефективності). Не виключенням є і розробка системи універсальних веб-інтерфейсів з використанням можливостей бібліотеки React.

Під універсальністю із самого початку мається на увазі властивість програмного коду, відповідно до якої його можна застосовувати не один раз в індивідуальному веб-додатку, а багато разів – в різних програмно-технічних рішеннях. Очевидною при цьому є економія робочого часу програмістів, а отже, і наявність певної економічної ефективності такої розробки, однак, для технічної роботи необхідним є виконання хоча б оціночних, але все ж таки чисельних розрахунків.

Для виконання такої оцінки задамося часами виконання різноманітних операцій, які необхідно виконати, аби довести систему до кінцевого програмного рішення.

Вся трудомісткість процесу доведення існуючого рішення до готового продукту може бути виражена формулою:

$$T = t_{роз} + t_{\alpha} + t_{кор.\alpha} + t_{\beta} + t_{кор.\beta} + t_{док},$$

де T – загальна трудомісткість;

$t_{роз}$ – час розробки;

t_{α} та t_{β} – затрати праці відповідно на альфа-тестування та бета-тестування;

$t_{кор.\alpha}$ та $t_{кор.\beta}$ – затрати праці на коригування знайдених відповідно при альфа-тестуванні та бета-тестуванні помилок;

$t_{док}$ – затрати праці на створення документації.

Стадія альфа-тестування передбачає, що весь програмний код написано, але в ньому існують очевидні помилки, що виникають в деяких особливих випадках виконання. Усі знайдені при альфа-тестуванні помилки

виправляються і продукт після закінчення цього етапу в цілому вважається завершеним. Після цього починається стадія бета-тестування, на якій виявляються приховані логічні або інші неочевидні невеликі помилки. Після виявлення усі помилки виправляються і розробка продукту є завершеною. Далі програмне забезпечення має бути документоване для його належного використання користувачем. Трудомісткість згаданих операцій для розроблюваної системи наведено в таблиці 6.1.

Табл.6.1. Трудомісткість основних операцій по створенню програмного продукту.

Назва процедури	Позначення	Трудомісткість, людино-годин	З них на комп'ютері, людино-годин
Розробка	$t_{роз}$	400	400
Альфа-тестування	t_{α}	160	160
Коригування помилок після альфа-тестування	$t_{кор.\alpha}$	80	72
Бета-тестування	t_{β}	80	80
Коригування помилок після бета-тестування	$t_{кор.\beta}$	40	32
Документування	$t_{док}$	32	24
Всього		792	768

Різні проектні процедури повинні виконувати різні люди – тестувати мають тестери, а виправляти помилки повинен розробник, але оскільки як розробник, так і тестери мають бути кваліфікованими спеціалістами в

комп’ютерних системах та мережах, їх рівень кваліфікації можна прийняти приблизно однаковим з відповідною однаковою оплатою праці.

Приймаємо вартість однієї людино-години кваліфікованого спеціаліста рівною 150 грн. Відповідно на оплату праці необхідно затратити:

ЗП = 150 грн/год ·792 год = 118800 грн.

Затрати МЧ на оплату машинного часу будемо спрощено рахувати як вартість затраченої електроенергії (відрахування на експлуатацію апаратних засобів не враховуємо, оскільки загальним часом їх використання при розробці продукту можна знехтувати, порівняно з загальним часом їх експлуатації):

МЧ = 768 год·(500 ват/1000)·1,69 грн/(кВт·год) = 649 грн.

Тут 500 ват – середня потужність сучасного комп’ютера, 1,69 грн/(кВт·год) – середня приблизна вартість однієї кВт*год електрики.

Розрахунок собівартості доведення проекту до серійного виробництва наведений в таблиці 6.2.

Табл.6.2. Розрахунок вартості розробки проектованої системи.

№	Найменування	Величина	Сума, грн.
1.	Основна заробітна плата		118800
2.	Премії	20% від п.1	23760
3.	Загальна заробітна плата	120% від п.1	142560
4.	Відрахування в пенсійний фонд	32% від п.3	45619
5.	Відрахування на соціальне страхування	2,9% від п.3	4134
6.	Відрахування у фонд зайнятості	1,9% від п.3	2709
7.	Фонд травматизму	0,2% від п.3	285
8.	Загальна сума відрахувань від заробітної плати	п.4+ п.5+ п.6+ п.7+	52747
9.	Фонд заробітної плати	п.3+ п.8	195307
10.	Вартість машинного часу		649
11.	СОБІВАРТІСТЬ	п.9+ п.10	195956

Розроблене рішення може окупатися за рахунок повторного використання коду (адже сама задача роботи звучала як розробка універсальної системи, тобто такої, що може застосовуватися у різних проектах).

Приймемо, що підприємство виконує 1 веб-додаток на місяць, вартість якого в цілому складає 6 тис. ум. од. ≈ 150 000 грн. За грубою оцінкою розробка інтерфейсу користувача займає біля 10% всього часу, що відводиться на створення програмного продукту (конкретна цифра може відрізнятися у кілька разів, зважаючи на тип веб-додатку, що реалізується, а 10% взято у якості середнього значення). Із цього часу розробки біля 50% може бути зекономлено через використання даного рішення.

Відповідно, строк окупаємості у місяцях може бути оцінений за наступною формулою:

$$N = \frac{195956}{150000 \cdot 10\% \cdot 50\%} = 26,$$

що є задовільним показником для технічної розробки і вкладається у прогнозовані розумні терміни використання бібліотеки React, як бази для зведення інтерфейсу користувача (бібліотека уже активно використовується біля 5 років і ще буде в експлуатації не менше 5).

Таким чином, дана робота є економічно обґрунтованою із орієнтовним терміном окупності біля 2 років (26 місяців). При розрахунку використовувалися дуже обережні оцінки (песимістичні, в гіршу сторону), тому реальні строки окупності можуть бути значно меншими (але не більше – за умови ефективної роботи IT-компанії, що реалізовує проекти).

Для підняття економічних показників даної розробки слід вести направлену маркетингову кампанію, яка у першу чергу полягає у більш активному впровадженні саме бібліотеки React у веб-додатки, які розробляє IT-компанія.

Ще одним способом підвищення економічних показників є зменшення затрат на розробку, які було оцінено із оплатою праці розробника на рівні

кваліфікованого програміста. Якщо ж врахувати, що розробка виконується в рамках випускної кваліфікаційної роботи, то затратами на оплату праці програміста можна взагалі знехтувати. При цьому окупність стає надзвичайно високою, і її терміни складають величину порядку місяця.

ВИСНОВКИ

У даній роботі досліджено питання зведення універсальних інтерфейсів користувача для веб-додатків з урахуванням специфіки їх використання на базі використання бібліотеки React.

Початково виконано аналітичний огляд існуючих науково-технічних джерел, присвячених темі використання React за призначенням, а саме, для зведення інтерфейсів користувача веб-додатків. Розглянуто як друковані, так і електронні джерела. Встановлено, що, незважаючи на те, що бібліотека з'явилася не так давно (2013 р.), на сьогоднішній день існує достатня кількість як якісних друкованих, так і різноманітних електронних джерел.

Після аналізу джерел виконано моделювання предметної галузі та встановлено вимоги до інтерфейсів користувача веб-додатків. Серед таких вимог можна назвати наступні основні: легкість та інтуїтивність роботи з системою, можливість прямого редагування даних, переважна робота в рамках однієї сторінки, очевидні запрошення до використання різноманітних опцій, застосування плавних переходів, а також миттєві реакції на дії користувача.

На основі аналізу цих вимог розроблено алгоритми побудови універсальних елементів інтерфейсів користувача (на базі бібліотеки React), а також запропоновано шляхи програмної реалізації цих елементів. Виконано тестування цих елементів, розроблено необхідну документацію.

В цілому, результати отримані в рамках виконання даного дослідження мають практичну направленість і можуть бути використані в реальних практичних роботах по зведенню ефективних інтерфейсів користувача веб-додатків з використанням сучасної бібліотеки React.

В перспективі запропоновані підходи по зведенню інтерфейсів для веб-додатків можуть бути адаптовані для мобільних систем (розширені чи модифіковані – для різних запропонованих концептів ситуація може відрізнятися, аж до повної неможливості застосування рішення через малі розміри екрану смартфону/планшету у порівнянні з монітором ноутбуку чи,

тим більше, настільного ПК. Таким чином, як і для всього світу IT перспективи даної роботи полягають у її переведенні на платформу React Native, яка, відповідно до її документації, являє собою повноцінний фреймворк і може застосовуватися для довільних мобільних операційних систем (платформ).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ:

1. Огляд бібліотек для форм в React [Електронний ресурс]. Codeguida, 2017–2019. URL: https://codeguida.com/post/1367.
2. React vs. Angular 2 [Електронний ресурс]. Codeguida, 2017–2019. URL: https://codeguida.com/post/662
3. Fullstack React/Rails: Знайомство [Електронний ресурс]. Codeguida, 2017–2019. URL: https://codeguida.com/post/371
4. Покращуємо компоненти багаторазового використання в React з шаблоном Overrides [Електронний ресурс]. Codeguida, 2017–2019. URL: https://codeguida.com/post/1588.
5. Rails та React: Справжнє використання (Частина 2) [Електронний ресурс]. Codeguida, 2017–2019. URL: https://codeguida.com/post/323
6. Rails та React: Золота середина (Частина 1) [Електронний ресурс]. Codeguida, 2017–2019. URL: https://codeguida.com/post/322
7. JavaScript’s History and How it Led To ReactJS [Електронний ресурс]. THENEWSTACK, 2019. URL: https://thenewstack.io/javascripts-history-and-how-it-led-to-reactjs/
8. React - A JavaScript library for building user interfaces [Електронний ресурс]. React, 2019. URL: https://reactjs.org/
9. React: Making faster, smoother UIs for data-driven Web apps [Електронний ресурс]. INFOWORLD, 2019. URL: https://www.infoworld.com/article/2608181/react--making-faster--smoother-uis-for-data-driven-web-apps.html
10. Facebook's React JavaScript User Interfaces Library Receives Mixed Reviews [Електронний ресурс]. INFOQ, 2006-2019. URL: https://www.infoq.com/news/2013/06/facebook-react/
11. React Without JSX [Електронний ресурс]. React, 2019. URL: https://reactjs.org/docs/react-without-jsx.html
12. Babel (составитель) - Babel (compiler) [Електронний ресурс]. Qw, 2019. URL: https://ru.qwertyu.wiki/wiki/Babel_(compiler)

13. Robbins J. N. Learning Web Design: A Beginner’s Guide to HTML, CSS, JavaScript, and Web Graphics. 5th ed. Sebastopol: O’Reilly Media, 2018.
14. Meyer E. A., Weyl E. CSS: The Definitive Guide: Web Layout and Presentation. 5th ed. Sebastopol: O’Reilly Media, 2023.
15. Flanagan D. JavaScript: The Definitive Guide: Master the World’s Most-Used Programming Language. 7th ed. Sebastopol: O’Reilly Media, 2020.
16. Stenberg D. HTTP/3 explained. 2nd rev. ed. 2020. URL: https://http3-explained.haxx.se/en
17. Biørn-Hansen A., Grønli T.-M., Majchrzak T. A Survey and Taxonomy of Core Concepts and Research Directions in Progressive Web Apps. ACM Computing Surveys. 2018. 51(5).108. DOI:10.1145/3241739.
18. Fauzan R., et al. A Systematic Literature Review on Progressive Web Application (PWA) Method Practices. Journal of Theoretical and Applied Information Technology. 2022. 100(19):6674–6697.
19. Aydos M., Kaya K., Gonen S. Security testing of web applications: A systematic mapping study. Journal of King Saud University – Computer and Information Sciences. 2022. 34(10):7652–7668.
20. Ehichoya O., et al. Evaluating security vulnerabilities in web-based applications: a qualitative assessment of scanners. arXiv. 2022. arXiv:2212.12308.

Додаток 1. Код мовою JavaScript, що є програмною реалізацією спроектованого рішення.

```
const N=10;

function IntTextarea(props)
{
    return <form>
    <textarea id={props.item_name} rows="1" cols="5">{props.item
_value}</textarea>
    </form>;
}

/*function makeNIntTextareas()
{
    let res="";
    for(let i=1;i<=N;i++)
        res+="<IntTextarea item_name=\"text"+i+"\" item_value=\"
"+0.003+"\" /><br/>";
    return res;
}*/

function App() {
    return (
      <div>
            <IntTextarea item_name="text1" item_value="0.003" />
<br/>
            <IntTextarea item_name="text2" item_value="0.653" />
<br/>
            <IntTextarea item_name="text3" item_value="0.974" />
<br/>
            <IntTextarea item_name="text4" item_value="0.555" />
<br/>
            <IntTextarea item_name="text5" item_value="0.029" />
<br/>
```

```
            <IntTextarea item_name="text6" item_value="0.005" />
<br/>
            <IntTextarea item_name="text7" item_value="0.003" />
<br/>
            <IntTextarea item_name="text8" item_value="0.323" />
<br/>
            <IntTextarea item_name="text9" item_value="0.003" />
<br/>
            <IntTextarea item_name="text10" item_value="0.303" /
><br/>

      </div>
    );
  }

ReactDOM.render(
    <App />,
    document.getElementById('root')
  );

  function getsaved(e)
  {
    //alert(e.target.responseText);
    let resp=+e.target.responseText;
    if(resp<0)
       document.getElementById('text'+Math.abs(resp)).style.back
groundColor="red";
    else
       document.getElementById('text'+Math.abs(resp)).style.back
groundColor="white";
  }

  function commit(e)
  {
    e.srcElement.cols=e.srcElement.value.length;
    const data=new FormData();
```

```
    data.append('changedobject',e.srcElement.id);
    data.append('newvalue',e.srcElement.value);
    const req=new XMLHttpRequest();
    req.addEventListener('load',getsaved,false);
    req.open("POST","processnewdata.php",true);
    req.send(data);
  }

  function sethandlers()
  {
      for(let i=1;i<=N;i++)
        document.getElementById("text"+i).addEventListener('blur
',commit);
  }

  window.addEventListener('load',sethandlers,false);
```

Додаток 2. Код мовою HTML, що є програмною реалізацією спроектованого рішення.

```
<?php header('Content-Type: text/html; charset=UTF-8');
?>
<html>
<head>
<title>
Універсальні інтерфейси на React
</title>
<script src="https://unpkg.com/react@16/umd/react.development.js
"></script>
<script src="https://unpkg.com/react-dom@16/umd/react-
dom.development.js"></script>
<script src="https://unpkg.com/@babel/standalone/babel.min.js"><
/script>
<script src="myscript.js"  type="text/babel"></script>
<link rel="stylesheet" href="mycss.css">
</head>
<body style="background-image: url(images/background.png)">
<div align="center" id="root"></div>
</body>
</html>
```

Додаток 3. Код CSS, що є програмною реалізацією спроектованого рішення.

```
textarea
{
    border:0;
}
```

Додаток 4. Код PHP, що використовується для тестування програмної реалізації спроектованого рішення.

```
<?php
header('Content-Type: text/html; charset=UTF-8');
$hostname="localhost"; $username="root"; $password="";
$dbName="datafromreact";
$link=mysql_connect($hostname,$username,$password) or die("No co
nnect to DB!");
mysql_set_charset('utf8',$link);
mysql_select_db("$dbName",$link) or die("Cannot select DB!");

$query="UPDATE tablewithvalues SET item_value=\"".$_POST["newval
ue"]."\" WHERE item_name=\"".$_POST["changedobject"]."\";";
$result=mysql_query($query);//print $result;
$num=+substr($_POST["changedobject"],4);
if(mysql_affected_rows($link)>0)
    print $num;
else
    print "-".$num;
?>
```