МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних
технологій

Кваліфікаційна робота бакалавра

на тему: Розробка універсального веб-інтерфейсу з використанням React

Виконав студент групи К-19
спеціальності 122 Комп'ютерні науки
Гутлиєв Батир

Керівник канд. техн. наук, доцент
Терещенко Т.М.

Рецензент док. техн. наук, проф.
Мещеряков В.І.

Одеса 2023

ЗМІСТ

СПИСОК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

DOM – Document Object Model.

SPA – Single-Page Application, односторінковий додаток.

UI – User Interface , інтерфейс користувача.

SaaS – Software as a Service, програмне забезпечення, як послуга.

CMS – Content Management System, система управління контентом.

ARIA – Accessible Rich Internet Applications, технологічний стандарт.

HTML – HyperText Markup Language, мова гіпертекстової розмітки.

КПІ – Конвенція про права людей з інвалідністю.

ПЗ – програмне забезпечення.

ВСТУП

Стрімкий розвиток інформаційних технологій став причиною їхнього активного використання в усіх сферах людського життя: економіка, виробництво, освіта тощо. Кожна комерційна організація або підприємство зацікавлено в розширенні своєї клієнтської бази та підвищенні прибутку. Некомерційні організації працюють над поширенням інформації про свою діяльність з метою залучення нових учасників. Всіх об'єднує те, що вони для цього використовують саме веб-технології, як засіб заявити про себе.

Сучасний ринок програмного забезпечення надає безліч послуг по створенню, просуванню, підтримці веб-ресурсів. Це може бути як прості SPA для представництва в мережі Інтернет, так і складні багаторівневі інформаційні системи управління підприємством з веб-інтерфейсом. Корпоративні веб-програми – це динамічні веб-сайти, які об'єднуються з програмуванням на стороні сервера. Вони допомагають компаніям автоматично керувати своїм бізнесом в Інтернеті та дозволяють їм використовувати додатковий час для інших корисних занять. Прикладами веб-додатків є онлайн-банкінг, соціальні мережі, онлайн-бронювання, програми електронної комерції/кошика для покупок, інтерактивні ігри тощо.

Використання таких веб-додатків надають підприємствам можливість ефективно керувати комерційною діяльністю та підвищувати рівень прибутків щорічно. Користувачі можуть отримувати доступ до веб-додатків незалежно від того, які операційні системи вони мають (Windows, Mac, Linux тощо). Для утримання існуючих клієнтів і залучення нових потрібна постійна взаємодія, яку забезпечує веб-додаток. Наприклад, веб-програму можна налаштувати для надсилання та отримання електронних листів. Використання веб-додатків не потребує встановлення додаткового ПЗ на жорстких дисках користувачів, що призводить до зниження пам'яті на комп'ютерах користувачів, яка постійно використовується. Це можна зробити безпосередньо на сервері, і всі оновлення можна ефективно розгорнути на комп'ютерах користувачів. Компанії

знижують свої витрати за рахунок використання веб-технологій, оскільки вони не потребують складного обладнання для підтримки та трудомістких витрат на оновлення.

Одним з важливих елементів такої системи є веб-інтерфейс, саме він виконує функцію зв'язку між користувачем та системою, являє собою "вітрину" підприємства. Він потребує постійного оновлення та удосконалення, щоб задовольняти вимогам користувачів і клієнтів.

Метою роботи є розробка універсального веб-інтерфейсу з використанням сучасних програмних інструментів.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Визначити основні структурні елементи веб-інтерфейсу, їхні функції та особливості. Розглянути принципи побудови універсального інтерфейсу, на основі яких сформулювати вимоги до проєкту.

2. Проаналізувати існуючі технології веб-розробки з метою вибору оптимальної для побудови універсального інтерфейсу. Обґрунтувати вибір програмних інструментів для розробки.

3. Обґрунтувати вибір бібліотека React для побудови універсального інтерфейсу, визначити основні компоненті та їхні параметри, які будуть використовуватися для розробки інтерфейсу.

4. Створити основні компоненти дизайну, які можуть бути використані при побудові універсального інтерфейсу.

5. Визначити основні інструменти для тестування та документування компонентів.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Структура та функції веб-інтерфейсу

Розробка дизайну інтерфейсу користувача (UI) починається з визначення елементів, які необхідні для того, щоб користувачі могли виконувати необхідні їм. При цьому доступ до цих елементів повинен бути легким та зрозумілим для будь-якого користувача, а також таким, що полегшує взаємодію користувач та системи. Інтерфейс користувача об'єднує концепції дизайну взаємодії, візуального дизайну та інформаційної архітектури (рис. 1).

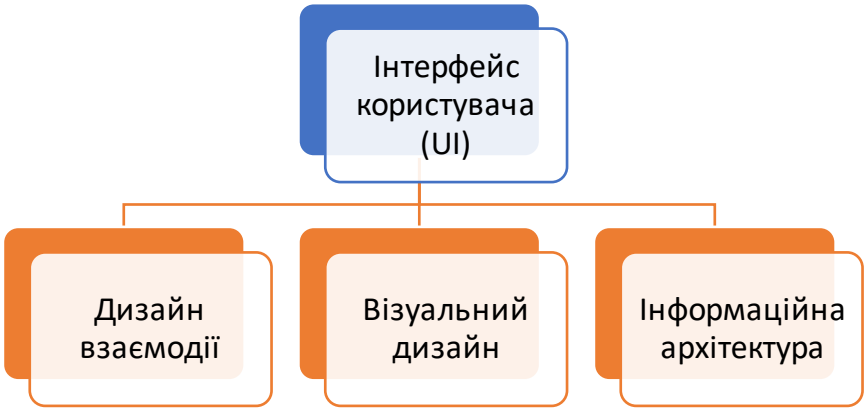

Рисунок 1 – Структура дизайну UI

Завдання вибору елементів інтерфейсу є першою і надважливою в процесі розробки та реалізації. Більшість користувачів вже мають досвід роботи з інтерфейсами та їхніми елементами, то основний принцип – це послідовність та передбачуваність у виборі елементів та їх компонуванні. Перш за все це стосується наступних елементів управління (рис. 2):

1. **Елементи керування введенням**: кнопки, текстові поля, прапорці, перемикачі, розкривні списки, поля зі списками, перемикачі, поля введення дати.

2. Навігаційні компоненти: навігація, повзунок, поле пошуку, розбиття на сторінки, повзунок, теги, піктограми.

3. Інформаційні компоненти: спливаючі підказки, піктограми, панель виконання, сповіщення, вікна повідомлень, модальні вікна.

4. Контейнери: вертикальне акордеоне меню.

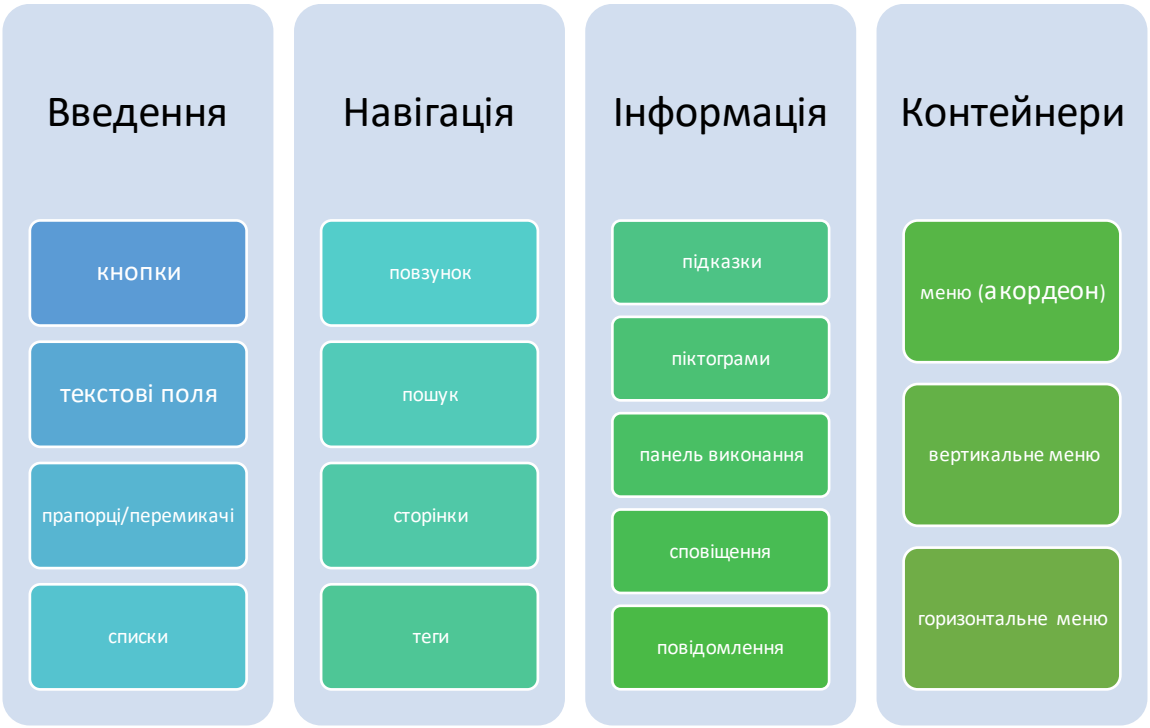

Рисунок 2 – Елементи управління в структурі дизайну UI

Бувають випадки, коли кілька елементів можуть підійти для відображення вмісту. Коли це трапляється, важливо враховувати компроміси. Наприклад, іноді елементи, які можуть допомогти розробнику заощадити простір, покладають на користувача більше психологічного навантаження, змушуючи його вгадувати, що знаходиться у списку або який це за елемент [1].

Для вибору найкращих методів розробки інтерфейсу слід проаналізувати деякі звички та здібності потенційних користувачів. Усе залежить від знань про користувачів, які включають розуміння цілей, навичок,

уподобань і схильностей. Після цього аналізу обов'язково слід врахувати наступні правила під час розробки інтерфейсу (рис. 3):

Рисунок 3 – Правила розробки дизайну UI

1. Забезпечення простоти інтерфейс. Найкращі інтерфейси практично непомітні для користувача. Вони уникають непотрібних елементів і чітко висловлюються на написах і в повідомленнях.

2. Наслідування послідовності і використання загальних елементів інтерфейсу користувача. Якщо використовувати загальні та відомі елементи в інтерфейсі, користувачі відчувають себе комфортніше та можуть швидше виконувати завдання. Також важливо створити шаблони в мові, макеті та дизайні на всьому сайті, щоб сприяти ефективності. Коли користувач навчиться щось робити, він зможе перенести цю навичку на інші частини сайту.

3. Забезпечення чіткої структури сторінки. Розгляньте просторові співвідношення між елементами на сторінці та структуруйте сторінку на

основі важливості. Ретельне розміщення елементів може допомогти привернути увагу до найважливіших фрагментів інформації та може сприяти скануванню та читабельності.

4. Вплив за допомогою кольору та текстури. Необхідно скеровувати увагу на предмети або відводити її від них за рахунок використання кольору, світла, контрасту і текстури.

5. Використання графіки для створення ієрархії та чіткості. Ретельно проаналізувати те, як використовується шрифт. Різні розміри, типи шрифтів та розташування тексту, щоб покращити сканування, розбірливість і читабельність інформації.

6. Забезпечення відповіді системи на дії користувача. Необхідне забезпечення інформування користувачів про місцезнаходження, дії, зміни стану чи помилки. Важливе використання різних елементів інтерфейсу користувача для передачі інформації про статус і, якщо необхідно, наступні дії. Це все може зменшити розчарування користувача.

7. Визначення значень за замовчуванням. Необхідно ретельно продумати та передбачити цілі користувачів, які люди приходять на сайт, створити стандартні параметри, які зменшать навантаження на користувача. Це стає особливо важливим, коли справа доходить до дизайну форми, де розробник може мати можливість попередньо вибрати або заповнити деякі поля.

1.2. Принципи побудови універсального веб-інтерфейсу

На початку роботи над дизайном веб-інтерфейсу розробники доволі часто забувають про важливий параметр – доступність. Часто це прихована потреба, про яку ніхто з команди не згадує до того моменту, поки щось не піде не так. Наприклад, скажімо, розробка перебуває в середині дизайн-проекту, і один із тестових користувачів виявляє, що не може прочитати текст на екрані.

Потім команда починає аналізувати те, що сталося, і виявляється, що цей користувач є одним із 8% чоловіків у світі, які мають дальтонізм, і він не може відрізнити зелений шрифт від червоного фону. Цей приклад свідчить про те, що саме доступність повинна стояти на першому місці в переліку принципів побудови веб-інтерфейсу.

Проблеми з доступністю можуть виникати протягом життєвого циклу проекту, а також, і після завершення проекту. Вирішення таких проблем потребує матеріальних та людських ресурсів. Крім того, вони можуть зашкодити іміджу компанії. У країнах із жорстким законодавством щодо доступності компанії можуть опинитися у дорогих судових позовах. Але окрім вирішення юридичних питань, доступність може принести користь користувачам, а також покращити бренд продукту. Все вище зазначене свідчить про те, що саме планування та проектування дизайну з урахуванням принципів доступності є однією з найважливіших задач у процесі розробки.

Вперше питання універсального дизайну було систематизовано колективом науковців на чолі з Роном Мейсом (USA) ще у далекому 1997 році. Ця група працювала над розробкою принципів у Центрі візуального дизайну одного з університету штату Північна Кароліна. Якщо розробники враховують ці принципи, вони отримують безліч переваг в процесі планування та керування розробкою веб-сайтів [2].

Принципи універсального дизайну є основою для дизайнерів, які збираються створювати продукти універсального дизайну. Принципи були створені, щоб керувати широким спектром дизайнерських дисциплін, включаючи середовища, продукти та комунікації. Розробники мають можливість застосувати ці принципи до будь-якого стилю чи напряму дизайну; вони поза часом і адаптуються. Розробник повинен завчасно займатися доступністю та врахувати ці принципи на початку проєкту [2].

Існує сім принципів, і всі вони включають вказівки з дієвими підходами для універсального дизайну (рис. 4). Кожен принцип містить ключову концепцію. Щоразу, коли розробник використовує вказівки для планування та

оцінки дизайну, важливо пам’ятати, що іноді лише кілька із семи принципів будуть актуальними для поточного дизайну. Розглянемо всі ці принципи та визначимо, які з них будуть актуальними для проєкту, що розробляється в рамках бакалаврської роботи.

Рисунок 4 – Принципи універсального дизайну

Перший принцип полягає у справедливості використання, переклад *«Дизайн корисний і продається людям з різними здібностями».* Справедливе використання є першочерговим принципом, оскільки це рушійна сила доступності. Принцип спонукає розробника думати про користувачів з різними здібностями. Коли використовується цей принцип, то треба враховувати всіх користувачів, а не лише цільову аудиторію. Коли розробники

враховують цей принцип, то вони не тільки покращують свій досвід у пошуку та орієнтації на цільову аудиторію, але і підвищують цінність самої компанії та бренду.

Основні рекомендації щодо справедливого використання:

1. Забезпечте однакові засоби використання для всіх користувачів: ідентичні, коли це можливо, еквівалентні, коли ні.

2. Уникайте сегрегації та стигматизації будь-яких користувачів.

3. Положення щодо конфіденційності та безпеки мають бути однаково доступними для всіх користувачів.

4. Зробіть дизайн привабливим для всіх користувачів.

Найчастіше у якості прикладу застосування цього принципу наводять використання контрасту кольорів у дизайні, що призводить до врахування інтересів користувачів, які мають дальтонізм. Орієнтація на таких користувачів обумовлена тим, що кольоровою сліпотою страждає приблизно 1 з 12 чоловіків (8%) і 1 з 200 жінок (0,5%) у світі. Розробник може уникнути сегрегації чи стигматизації своїх користувачів шляхом створення кольорової палітри з високим контрастом. Однак, є ті хто виступає проти доступності. У якості аргументу вони наводять те, що доступність знижує якість візуального дизайну в цілому. Але це помилкове твердження тому, що саме дизайн з високим контрастом може стати естетично привабливим для всіх користувачів.

Другий принцип полягає у гнучкості використання, переклад з англійської – *«Дизайн враховує широкий спектр індивідуальних уподобань і можливостей»*. Цей принцип зосереджується на тому, що жодна людина не схожа на іншу. Статичний і негнучкий дизайн ніколи не зможе вмістити всіх користувачів. Принцип гнучкості у використанні заохочує гнучкий, адаптований та/або настроюваний дизайн. Він враховує індивідуальні переваги та дозволяє користувачам обирати шляхи використання готового продукту. Надання вільного вибору користувачу робить веб-сайт більш контрольованим та привабливим.

Основні рекомендації щодо гнучкості використання:

1. Забезпечити вибір способів використання.
2. Можливість доступу та використання правою чи лівою рукою.
3. Сприяти точності дій користувача.
4. Забезпечити адаптацію до темпу користувача.

У якості прикладу такого дизайну доволі часто наводять функцію налаштування інформаційної панелі. Індивідуалізація – це техніка, що дозволяє врахувати широкий спектр індивідуальних уподобань і здібностей. Це дозволяє користувачам вибирати та організовувати те, що вони бачать на веб-сайті, і як вони будуть цим користуватися. Інформаційні панелі є хорошими прикладами для налаштування. Багато корпоративних систем і програм управління проектами мають функцію налаштування інформаційної панелі. Залежно від різноманітних робочих завдань і потреб користувачі можуть вибрати, що вони хочуть бачити на інформаційній панелі та як це використовувати. Один з найкращих прикладів – Trello – це веб-додаток для керування проєктами (рис. 5).

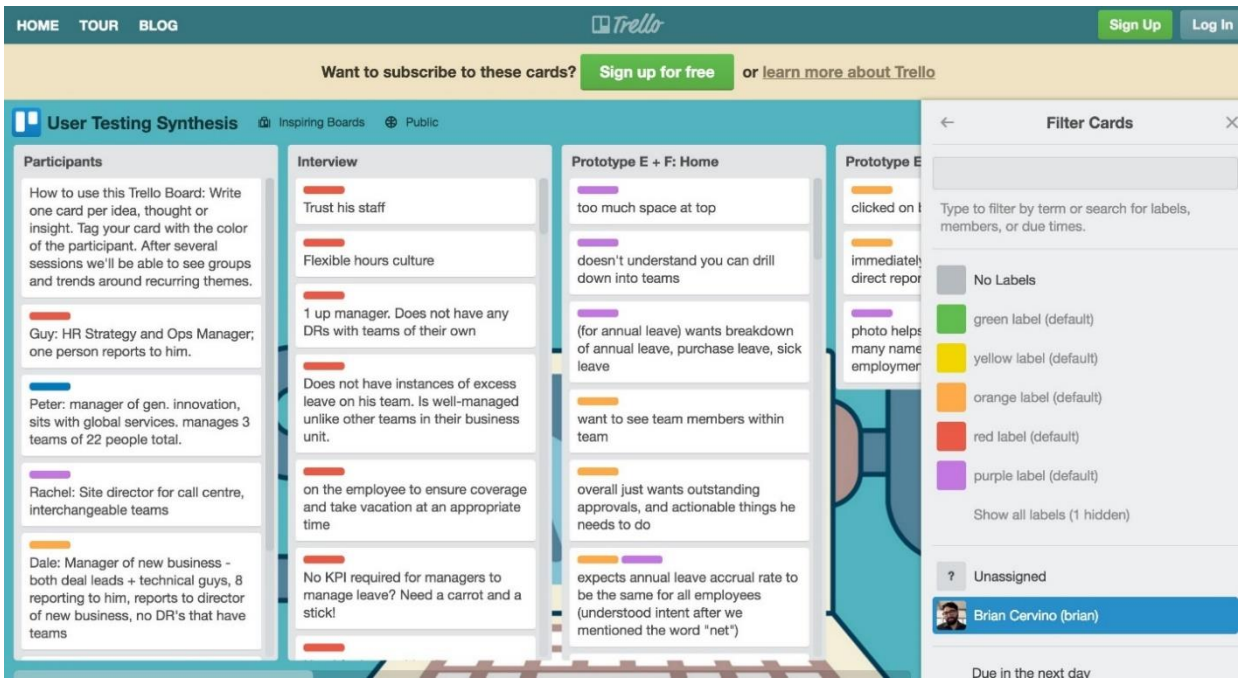

Рисунок 5 – Принципи універсального дизайну (джерело https://trello.com)

Він використовує дошки, списки та картки, щоб допомогти користувачам створити гнучку організацію середовища та розставляти пріоритети своїх проектів. Він пропонує ряд параметрів інформаційної панелі, які настроюються. Користувач може вибрати колір карток для відображення на дошці [3].

Третій принцип полягає у простоті та інтуїтивний зрозумілості використання, звучить – «Використання дизайну легко зрозуміти, незалежно від досвіду користувача, знань, мовних навичок або поточного рівня концентрації». Просте та інтуїтивно зрозуміле використання є однією з цілей дизайну взаємодії з користувачем. Це також один із універсальних принципів дизайну. Цей принцип спрямований на зменшення складності та розумового або когнітивного навантаження. Відповідно до теорії когнітивного навантаження , під час обробки інформації люди можуть обробляти лише 3–9 елементів за короткий проміжок часу. Щоб зменшити складність і зменшити когнітивне навантаження, розробники завжди повинні прагнути представити інформацію від 3 до 9 елементів.

Основні рекомендації щодо простого та інтуїтивно зрозумілого використання:

1. Усуньте непотрібну складність.
2. Будьте узгоджені з очікуваннями та інтуїцією користувачів.
3. Розраховуйте на широкий спектр грамотності та мовних навичок.
4. Розташуйте інформацію відповідно до її важливості.
5. Забезпечуйте ефективні підказки та зворотній зв’язок під час і після виконання завдання.

Четвертий принцип сконцентрований на забезпеченні відчутності інформації, переклад – «Дизайн ефективно передає необхідну інформацію користувачеві, незалежно від умов навколишнього середовища чи сенсорних здібностей користувача». Інформація є критичною для користувачів. Незалежно від того, передається це через текст, зображення, аудіо чи відео, розробник повинен подбати про те, щоб інформація була легкою для засвоєння

та доступу. Впроваджуючи цей принцип у дизайн треба зосередитися на роботі з користувачами. Розробник має з'ясувати, як найкраще представити інформацію, враховуючи користувачів з обмеженими можливостями, наприклад, із вадами зору чи слуху.

Рекомендації щодо відчутності інформації:

1. Використовуйте різні способи (зображувальний, вербальний, тактильний) для надлишкового подання важливої інформації.

2. Забезпечте відповідний контраст між важливою інформацією та її оточенням.

3. Збільште «розбірливість» важливої інформації.

4. Розрізняйте елементи способами, які можна описати (тобто, щоб було легко давати інструкції чи вказівки).

5. Забезпечити сумісність із різноманітними техніками чи пристроями, які використовують люди з сенсорними обмеженнями.

У якості прикладу слід навести використання транскрипцій або субтитри для відео-контенту, щоб користувач із вадами зору теж міг його переглядати.

П'ятий принцип – це толерантність до помилок, переклад – «Дизайн мінімізує небезпеки та несприятливі наслідки випадкових або ненавмисних дій». Помилки серед людей неминучі, тому існує прислів'я «людині властиво помилятися». Універсальний дизайн спрямований на проектування для всіх користувачів, а також на проектування з урахуванням різних середовищ і дій користувачів спонукає розробника думати не лише про екран, а й про те, як система й користувач взаємодіятимуть один з одним.

Основні рекомендації щодо толерантності до помилок:

1. Розташуйте елементи так, щоб звести до мінімуму небезпеки та помилки: найбільш використовувані елементи, найбільш доступні, небезпечні елементи вилучені, ізольовані або екрановані.

2. Надайте попередження про небезпеку та помилки.

3. Забезпечте безвідмовні функції.

4. Перешкоджайте несвідомим діям у завданнях, які вимагають пильності.

Простий та зрозумілий приклад цього принципу – перевірка заповнення форми на веб-сайті. Перевірка форми мінімізує та запобігає помилкам користувача. Існує три способи перевірки введених даних:

1. Перевірка формату введення – переконайтеся, що користувач використав правильний формат для поля введення. Наприклад, поле введення адреси електронної пошти має мати формат, що починається з рядка літер, за яким слідує символ «@» та ім'я домену електронної пошти.

2. Перевірка даних – перевірте, чи введені користувачем дані знаходяться в правильному контексті. Наприклад, багато систем бронювання готелів вимагають, щоб дати заїзду були не раніше «поточного» дня.

3. Перевірка сервера – перевірка формату введення та перевірка даних застосовуються до певного поля введення. Перевірка сервера надсилає всі дані форми на сервер і перевіряє правильний зв'язок даних. Наприклад, проста форма входу використовує перевірку сервера для перевірки правильності імені користувача та пароля.

Наступний принцип – це забезпечення низьких фізичних зусиль, переклад – «Цю конструкцію можна використовувати ефективно, комфортно та з мінімальною втомою». Багато людей використовують свої комп'ютери протягом восьми чи більше годин для виконання завдань на роботі. Час, який робітники проводять за комп'ютерами, завдає шкоди тілу. Насправді люди з обмеженими фізичними можливостями мають навіть більше труднощів з використанням Інтернету, ніж звичайні користувачі. Наприклад, тим, хто має проблеми з пересуванням, може бути важко перемістити мишу до потрібної цілі. Ось чому під час роботи важливо пам'ятати про розрахунок на низькі фізичні зусилля.

Рекомендації щодо низьких фізичних зусиль:

1. Дозвольте користувачеві зберігати нейтральне положення тіла.

2. Використовуйте розумні робочі сили.

3. Мінімізуйте повторювані дії.

4. Зведіть до мінімуму тривалі фізичні зусилля.

Прикладом використання цього принципу є мінімізація використання миші для дій на сторінці сайту за рахунок передбачених комбінацій клавіш. Вони зменшують потребу переходити з клавіатури на мишу для простих завдань. Більшість браузерів, наприклад, Chrome, Firefox і Safari, надають комбінації клавіш для виконання таких завдань, як копіювання (Ctrl + C) і вставлення (Ctrl + V). Розробник може створити чудовий інтерактивний досвід шляхом додавання відповідних комбінації клавіш. Вони покращать навігацію та спростять використання веб-сайтів для всіх користувачів.

Наприкінці цього списку стоїть принцип забезпечення розміру і простору для підходу та використання, переклад – «Забезпечується відповідний розмір і простір для підходу, досяжності, маніпуляцій і використання незалежно від розміру тіла, пози чи рухливості користувача».

Для дизайнерів продукту цей принцип зосереджено на таких факторах форми, як розмір і простір продукту. Для веб-дизайнерів – не на формах, як факторах, а більше на тому, що відображається на екрані. Це помилкове мислення, оскільки важливо думати поза екраном і також враховувати середовище наших користувачів, особливо тому, що користувачі переглядають веб-сайти не лише на настільних комп’ютерах, але й все частіше на мобільних пристроях. Більшість настанов щодо цього принципу стосуються більше продукту та дизайну середовища. Тим не менш, розробникам потрібно розглянути принцип і його вказівки, щоб створити веб-сайт універсального дизайну, як для настільних комп’ютерів, так і для мобільних пристроїв.

Рекомендації щодо розміру та простору для підходу та використання:

1. Забезпечте чітку лінію видимості важливих елементів для будь-якого користувача, який сидить або стоїть.

2. Зробіть доступ до всіх компонентів зручним для будь-якого користувача, який сидить або стоїть.

3. Пристосовуються до різних розмірів руки та ручки.

4. Забезпечте достатній простір для використання допоміжних пристроїв або особистої допомоги.

Для оцінки використання такого принципу слід розглянути цільову область веб-сайту, коли він розміщений на мобільних пристроях. На робочому столі користувач взаємодіє з веб-сайтом за допомогою маленького покажчика на екрані. На мобільному пристрої користувач взаємодіє з веб-сайтом за допомогою вказівного або великого пальця. Маленька цільова область може бути проблемою на мобільних пристроях, оскільки її складніше вибрати з точністю. Згідно з дослідженням MIT Touch Lab у 2003 році, середній розмір вказівного пальця дорослої людини становить від 1,6 до 2 см. Перетворивши це, маємо приблизно 60–76 пікселів на цифровому екрані. Отже, при проектуванні для мобільних пристроїв слід переконайтися, що цільові зони дотику враховують фізичні фактори людини.

1.3. Вимоги до проєкту

Найважливішою міжнародною роботою щодо доступності є Конвенція про права людей з інвалідністю (КПІ). Законодавці та спільнота людей з обмеженими можливостями називають це Конвенцією. Це перша велика угода про права людини 21 століття, яка захищає права та гідність людей з обмеженими можливостями. Вона закликає до усунення екологічних бар'єрів і бар'єрів у фізичному та цифровому просторі. Організація Об'єднаних Націй (ООН) прийняла КПІ у 2007 році. Конвенція перерахувала універсальний дизайн як одне із загальних зобов'язань щодо захисту прав людей з обмеженими можливостями. «Здійснювати або сприяти дослідженням і розробкам товарів, послуг, обладнання та засобів універсального призначення, як визначено в статті 2 цієї Конвенції, які повинні вимагати мінімально можливої адаптації та найменших витрат для задоволення конкретних потреб людини з обмеженими можливостями, сприяти їх доступності та

використанню, а також сприяти універсальному дизайну при розробці стандартів і настанов.

Створення доступних продуктів може бути складним завданням. Іноді дизайнери навіть не знають з чого почати. Універсальний дизайн – це надійний підхід до дизайну, оскільки сім принципів допомагають враховувати потреби всіх користувачів загалом. Універсальний дизайн приносить користь усім, а не лише старіючому населенню чи людям з обмеженими можливостями. Принципи універсального дизайну – це чудовий ресурс для розробників, коли вони мають на меті створити доступні веб-сайти, які обслуговують якомога більше користувачів. Ці принципи допоможуть керувати процесом проектування.

З урахуванням проведеного аналізу цілей та принципів побудови універсального дизайну, визначимо наступні вимоги до проєкту:

1. Забезпечити розробку дизайну з високою контрастність для урахування потреб людей з дальтонізмом, але при цьому, не втратити привабливості для аудиторії в цілому.

2. Передбачити функцію налаштування головної сторінки сайту під потреби окремо взятого користувача та з урахуванням його вподобань.

3. Знизити когнітивне навантаження на користувача за рахунок використання невеликої кількості елементів.

4. При розташуванні інформації на сторінці врахувати її важливість та знизити вимоги до мовних навичок користувача. А також, забезпечити виділення важливої інформації для спрощення її сприймання.

5. Врахувати при розташуванні елементів принцип толерантності до помилок, за рахунок розташування найбільш використовуваних елементів у найбільш доступних місцях. Всі форми на сайті зробити з функцією перевірки вмісту перед відправленням.

6. Забезпечити гнучкість дизайну для звичайної та мобільної версії.

2. ВИБІР ТЕХНОЛОГІЇ ВЕБ-РОЗРОБКИ

Після докладного аналізу принципів побудови універсального дизайну слід визначити інструменти та ресурси, необхідні для його реалізації. У веб-розробці це часто означає пошук найбільш підходящих фреймворків із функціями та перевагами, необхідними для прискорення та покращення зусиль розробників. Причини вибору однієї рамки замість іншої можуть відрізнятися від проекту до проекту. У деяких випадках найкращим рішенням може бути жорстка і сувора інструкція, тоді як для інших краще гнучкіший вибір. Для деяких проектів гнучкість і налаштування, які забезпечуються відсутністю фреймворку, можуть виявитися величезною перевагою. Це один із методів, який розробники можуть використовувати для створення сервісів, які виділяються з великої кількості веб-ресурсів. Завдяки досвідченим і кваліфікованим веб-розробникам це може бути дуже доречним варіантом.

2.1. Системи управління вмістом сайту (CMS)

Системи управління вмістом (CMS) – це комп’ютерні програми, які дозволяють користувачам організовувати свій цифровий вміст і керувати ним. Найпоширеніші види використання CMS включають керування веб-контентом і корпоративним вмістом. Системи керування веб-контентом зазвичай дозволяють декільком користувачам створювати та редагувати вміст на веб-сайтах, екстранетах або інтранетах. Вони покладаються на прості інтерфейси та шаблони, які допомагають нетехнічним користувачам (наприклад, редакторам і авторам) контролювати, як їхній вміст відображається в Інтернеті. Корпоративні системи керування контентом зазвичай об’єднують веб-контент, керування документами та записами, інструменти робочого процесу та співпраці тощо на одній платформі [4].

Веб-CMS забезпечують доступ до бази даних веб-сайту через простий графічний інтерфейс користувача, зазвичай через веб-браузер. У цьому інтерфейсі є доступ до ряда інструментів керування вмістом, які реалізують наступні функції:

- створення та публікація нових сторінок на веб-сайті;
- оновлення або видалення наявного вмісту і сторінки;
- використання попередньо встановлені категорії, теми або шаблони для організації макета ваших сторінок;
- забезпечення послідовного представлення вмісту на всьому веб-сайті;
- керування структурою та навігацією веб-сайту, включаючи меню та карти сайту;
- керування рівнями авторських дозволів і редакційними процесами;
- збереження та отримання різних типів вмісту (наприклад, текст, зображення, подкасти, відео) у базі даних.

Існує три типи програмного забезпечення CMS: з відкритим кодом, пропрієтарне та програмне забезпечення як послуга CMS, включаючи хмарні рішення.

CMS з відкритим кодом гарантують користувачу можливість безкоштовно завантажити програмне забезпечення CMS та використовувати його для створення власного веб-ресурсу. Немає жодної плати за ліцензію чи оновлення чи контрактів. Однак із CMS з відкритим кодом, можливо, доведеться заплатити за технічну допомогу під час встановлення та налаштування, налаштування для розширення програмного забезпечення за межі основної пропозиції, сумісні шаблони, доповнення та плагіни (хоча можуть бути доступні безкоштовні версії), навчання персоналу, підтримку,

включаючи регулярне оновлення програмного забезпечення. Приклади найпоширеніших платформ CMS з відкритим кодом наведені на рис. 6.

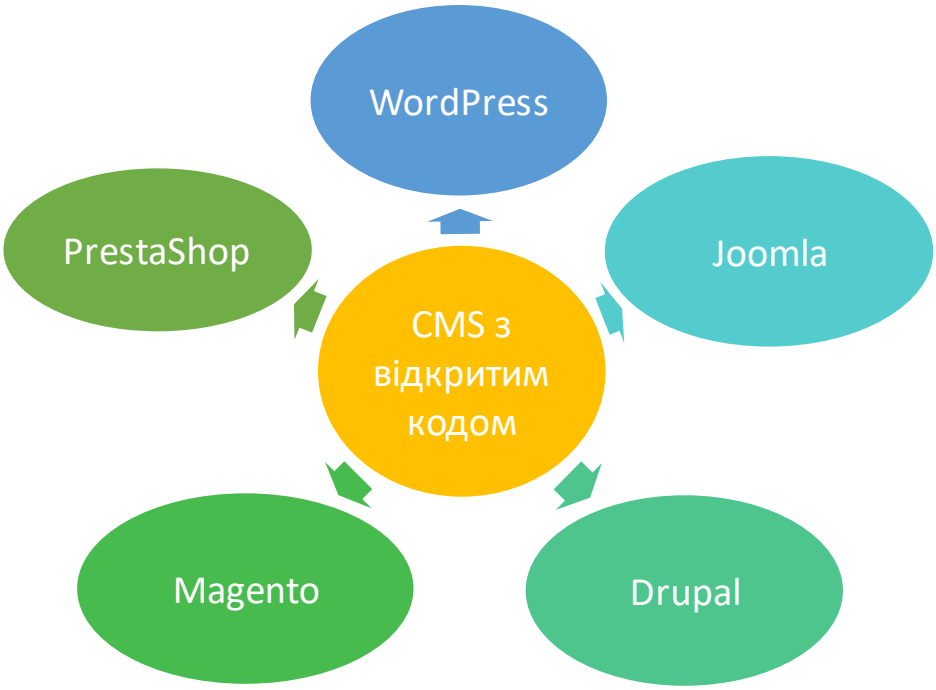

Рисунок 6 – Найпоширеніші платформи CMS з відкритим кодом

Власне або комерційне програмне забезпечення CMS створено та керується однією компанією. Використання такої CMS зазвичай передбачає наявність ліцензійної плати за використання програмного забезпечення та сплати щомісячної або річної плати за оновлення або підтримку. Можливі додаткові витрати за налаштування та оновлення, а також за навчання та постійну технічну підтримку або підтримку користувачів.

Приклади популярних рішень CMS:

- Kentico
- Microsoft SharePoint
- IBM Enterprise Content Management
- Імпульсна CMS
- Sitecore

- Shopify

Використання таких систем дозволяє налаштувати власну CMS із вбудованими функціями, хоча це може вимагати додаткову плату. Ефективніше використовувати рішення CMS, яке відповідає всім вимогам із коробки. Впровадження власної CMS з наявним веб-сайтом або серверною системою вимагає значної роботи з розробки. У випадку пошуку CMS для абсолютно нового веб-сайту, найкраще вибрати рішення, яке має всі важливі функції та функції для задоволення поточних і майбутніх бізнес-потреб.

Рішення SaaS CMS (програмне забезпечення як послуга) зазвичай включають програмне забезпечення для керування веб-контентом, веб-хостинг і технічну підтримку від одного постачальника. Це віртуальні рішення, розміщені в хмарі та засновані на моделі підписки, як правило, на основі кожного користувача або сайту. Ціна зазвичай включає обсяг передачі даних (тобто пропускна здатність до та з сайту), зберігання вмісту та даних, постійна підтримка.

Існує два типи хмарних систем керування контентом:

1. «Повністю хмарна» CMS часто постачається як частина пакета або послуги. Як правило, це запатентовані системи під контролем постачальника, тому не завжди можливо налаштувати або змінити їх функціональні можливості відповідно до власних потреб.

2. «Часткова хмара» CMS розташована на власному хмарному веб-сервері. Це забезпечує більшу гнучкість, оскільки власник має можливість змінювати функціональні можливості за допомогою додаткових модулів або змінюючи вихідний код.

До основних переваг CMS слід віднести:

- зручність використання;
- швидке розгортання;
- простота обслуговування, включаючи оновлення;

- економічна ефективність, особливо з готовими рішеннями, відкритим кодом або безкоштовним програмним забезпеченням;
- розширювана функціональність за допомогою великої кількості плагінів і розширень;
- зручні для SEO функції;
- підтримка розробників і спільноти.

Однією з головних переваг CMS є те, що вона дає змогу користувачам, які не мають технічного мислення, створювати функціональні сторінки або самостійно завантажувати та змінювати вміст, не передаючи цю роботу веб-розробнику чи розуміючи мови програмування, такі як HTML або PHP.

Незважаючи на численні переваги, є кілька загальних проблем, які слід розглянути перед тим, як вибрати CMS. До недоліків систем управління контентом CMS слід віднести:

- можуть бути приховані витрати (наприклад, на впровадження, налаштування, підтримку або навчання);
- значні серверні ресурси для певних типів CMS;
- регулярні оновлення та виправлення, щоб забезпечити безпеку програмного забезпечення;
- складне або дороге налаштування;
- складний експорт даних або перехід на іншу платформу;
- орієнтація лише на одного постачальника.

2.2. Фреймворки та бібліотеки

Фреймворки являють собою структуру, яка забезпечує основу для успішної побудови проєкту. Надання архітектури, абстракції та деякого початкового коду для досягнення поставлених цілей забезпечує заощадження

часу і розвиток проєкиу на кожному етапі розробки. З цієї потужної бази розробники мають можливість розширюватись і розвиватися в напрямку, який підтримує належну робочу практику, дотримуючись структур і шаблонів, передбачених структурою.

Одним з моментів плутанини, який часто виникає навіть серед розробників, є різниця між бібліотекою та фреймворком. Доволі часто ці терміни використовуються як синоніми для опису одного поняття. Визначене розмежування цих двох понять зустрічається не так часто. Але слід зазначити, що все таки існує відмінність між бібліотеками та фреймворками. Бібліотека повинна надавати одну або кілька колекцій функціональних можливостей і помічників для зв'язування проекту. Фреймворк, навпаки, повинен надавати основу коду, з якого розробник має можливість створювати власну програму.

Більшість фреймворків, ймовірно, включають одну або більше бібліотек, але вони також прописуватимуть або принаймні пропонуватимуть деякі архітектурні шаблони, які допоможуть розробити надійну програму. Хоча це робоче визначення також містить поняття, які далеко не чітко визначені, воно надає приблизне, але корисне правило для роботи під час обговорення цих технологій.

Для визначення технології веб-розробки, яку слід використовувати у проєкті, розглянемо переваги та недоліки використання фреймворку для розробки додатків.

До переваг слід віднести:

1. Спрощення багатьох завдань і викликів.

Для складних, високонавантажених або надзвичайно насичених систем і додатків інфраструктура може вважатися абсолютно необхідною. Закладання міцної архітектурної основи створює добре продуману програму, яку легко створювати та підтримувати без втрати продуктивності програми або контролю над її структурою. Для команд інженерів структура визначає мову проектування системи таким чином, щоб усі були миттєво знайомі з нею та могли легко спілкуватися та працювати над нею. Для невеликих систем

правильний фреймворк значно збільшує кількість рядків коду, які інженери повинні писати, тестувати та підтримувати вручну. Високоякісний фреймворк із великою кількістю функціональних можливостей і попередньої збірки системи може заощадити тижні часу на розробку та скоротити написання тисяч рядків коду.

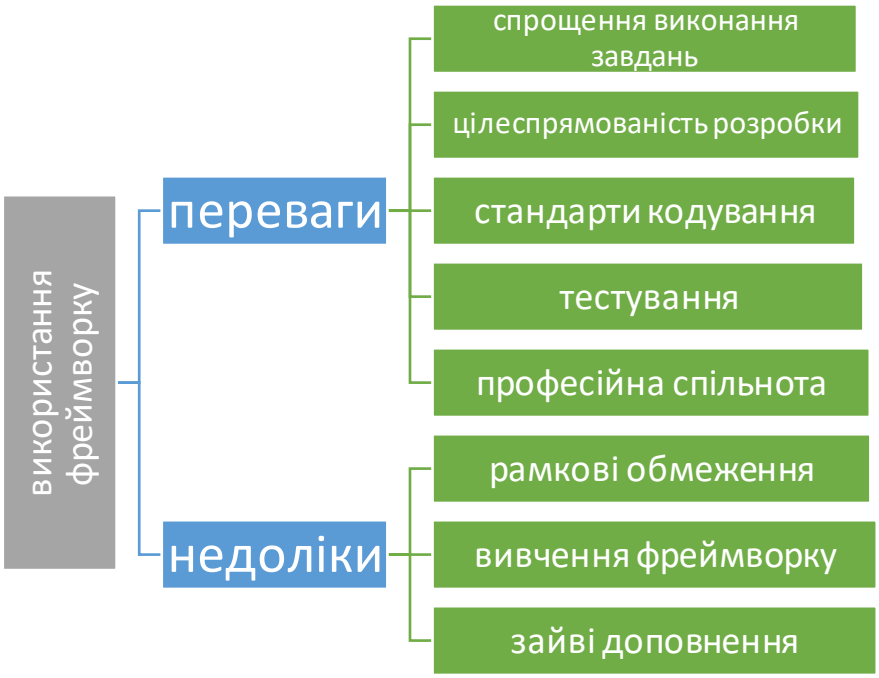

Рисунок 7 – Переваги та недоліки використання фреймворку

Прикладом реалізації цієї переваги є React.js. Це фреймворк із усім необхідним, попередньо запакованим і готовим до розгортання. Беручи до уваги базову роботу, фреймворк має звільнити розробників додатків зосередити свій час і увагу на проблемах вищого рівня.

2. Цілеспрямована розробка додатків

Фреймворк ефективно запускає процес розробки програми. Розробники можуть пришвидшити навіть найбільш налаштовані веб-програми від початкової ідеї до робочого прототипу за відносно короткий термін, використовуючи відповідну структуру. Шаблони, наприклад, є ідеальним інструментом для скорочення часу на написання шаблонного коду. Забезпечуючи прискорений початок ранніх стадій проекту, вони дозволяють

розробникам додатків зосередити свій час на реалізації творчих рішень більш нагальних проблем. Завдяки завантаженню базової функціональності, структури та архітектури фреймворки зберігають кінцеву програму в ідеальній формі для майбутнього розвитку.

3. Стандартизовані практики кодування

Одна з найбільших переваг, яку може принести ефективно обраний фреймворк, – це його здатність створювати чітко визначену структуру, якої слід дотримуватися протягом усього життя проекту. Сьогодні комунікація як між командами, так і між розробниками є однією з найактуальніших проблем, які вирішують команди розробників програмного забезпечення. Залучення нових розробників і передача проектної роботи значно спрощується, коли структуру проекту можна пояснити просто структурою, у якій він написаний. Переваги стандартизації приносять багаті дивіденди в проектуванні, розробці, налагодженні та розгортанні.

4. Тестування та налагодження

Переваги міцної та надійної структури програми особливо помітні на етапах тестування. Часто це найбільш трудомісткі та важкі завдання в життєвому циклі розробки, правильна структура для проекту може підтримати фазу тестування за допомогою відповідних інструментів і ресурсів. Програми, створені на основі жорсткішої архітектури, як правило, легше відокремити та перевірити окремо, ніж ті, що створені за допомогою спеціального рішення.

5. Наявність широкої та кваліфікованої спільноти

Мабуть, однією з найбільш недооцінених переваг розробки додатків із високопродуктивної системи є спільнота розробників, інженерів і користувачів, доступних для його підтримки. Сильні спільноти користувачів створюють велику базу знань зі статей, документації та обговорень для вирішення типових проблем, з якими стикаються розробники. Фреймворк із сильною спільнотою може легко висвітлити шляхи розвитку, які можуть

виявитися продуктивними та вартими. До фреймворків з особливо сильними спільнотами розробників можна віднести: ReactJS, VueJS і Django.

Для визначення можливості і необхідності використання фреймворку в процесі розробки веб-ресурсу необхідно відповісти на декілька запитань. По-перше, визначити наскільки обраний фреймворк для поставленого завдання. Наступне запитання стосується кваліфікації розробників, чи є вони новачками, чи мають вони достатній рівень практичних навичок. Далі слід розглянути функціональні можливості обраного фреймворку та визначити чи не перевантажений він зайвими функціями. Все це спростить вирішення проблем в процесі розробки, тестування та обслуговування у майбутньому.

Незважаючи на визначені переваги, фреймворки мають і недоліки використання. До них слід віднести наступні:

1. Рамкові обмеження

Фреймворк зазвичай легко налаштовується, але не може бути змінений фундаментально. Одне з найскладніших завдань, наприклад, адаптація для розміщення в середовищі, що призначено для легкої конструкції. Розробники змушені поважати обмеження та умовності, накладені його дизайном. Хоча це може бути перевагою, гарантуючи суворе дотримання конвенцій і належної практики, розробники повинні бути впевнені, що вони обирають структуру, яка відповідає їхнім потребам і вимогам. Вимушеність змінювати та перекручувати проєкт, щоб відповідати йому на наступних етапах, може спричинити стільки проблем, скільки і вирішить.

2. Необхідність вивчення фреймворку, а не мови програмування

Суттєвим недоліком для команд є те, що велика опора на структуру може затьмарити частину навчання, необхідного для кращого розуміння системи або самої мови. Проблеми та труднощі, з якими стикаються розробники, наприклад, у jQuery, відрізняються від тих, з якими стикаються ті, хто пише простий JavaScript. Розробники, знайомі з одним фреймворком, не обов’язково

знають інший. Оскільки фреймворк існує для того, щоб виконувати велику частину важкої роботи, розробники можуть залишити деякі глибші частини програми недоторканими. Навіть працюючи з кодом щодня, інженери можуть не знати тонкощів системи, коли взаємодіятиме лише з вищими рівнями фреймворку. Результатом надмірної залежності від фреймворку є менше шансів для програмістів вирішити складні проблеми та отримати детальне розуміння всієї програми.

3. Непотрібні доповнення

Фреймворк, за необхідності, постачається з усім необхідним для задоволення широкого спектру проектів і випадків використання. На практиці це означає, що фреймворк матиме функції та код, які взагалі не будуть використовуватися у майбутньому проєкті. Насправді основна частина коду, включена у велику структуру, цілком може бути нерелевантною. Особливо це стосується створення надзвичайно простих веб-додатків. Зайві файли та непотрібний код у проєкті можуть негативно вплинути на швидкість і продуктивність фреймворку та сайтів, які він підтримує. Легкий фреймворк, який реалізує мінімальні додаткові функції, може діяти як середина між стратегіями розробки без фреймворків і важкими фреймворками з багатьма функціями. Деякі особливо легкі фреймворки, які можуть накладати менше обмежень на проєкти, це – Express у JavaScript або Flask для розробки Python.

Існують випадки, коли не слід використовувати фреймворки взагалі. Умовами такого варіанту є наявність висококваліфікованої команди, яка пропонує різні рішення на довгострокову перспективу. Або, якщо однією з цілей розробки є поглиблене вивчення мови та її технології командою для подальшого використання цих навичок. Ще одна причина відмовитися від використання фреймворку обумовлена третім недоліком, а саме, коли потрібно рішення, яке є винятково легким і не має додаткового обладнання.

2.3. Обґрунтування вибору інструментів розробки

У попередніх розділах розглядалися готові системи керування контентом та бібліотеки/фреймворки, як інструменти для розробки універсального інтерфейсу. Аналіз показав, що використання CMS не є можливим, оскільки не забезпечує гнучкості процесу розробки інтерфейсу. І хоча, існує можливість створення власного шаблону та його інтеграції, все ж таки такий варіант не вирішує основного завдання. Використання фреймворків та бібліотек є ефективним засобом розробки універсального інтерфейсу. Вони дозволяють створювати власні шаблони з використанням готових елементів, що значно прискорює та спрощує процес розробки.

У якості інструменту розробки розглянемо React, як такий, що відповідає всім вимогам. React – це бібліотека JavaScript з відкритим кодом, розроблена командою з Facebook. Це бібліотека орієнтована перш за все саме на створення зручних інтерфейсів користувача. Вона стимулює створення компонентів інтерфейсу користувача, які представляють дані, що змінюються з часом і дозволяє їх повторно використовувати. Більшість інтерфейсних розробників JavaScript поєднують його з такими фреймворками, як Angular і Vue, для створення складних функцій.

React не використовує шаблони . Традиційно інтерфейси веб-додатків створюються за допомогою шаблонів або директив HTML. Ці шаблони зазвичай диктують повний набір абстракцій, які розробнику дозволено використовувати для створення інтерфейсу користувача. Підхід React до створення інтерфейсів інший – він розбиває їх на компоненти. React може обробляти шари перегляду веб- та мобільних додатків, а отже, підтримує розробку веб- та мобільних додатків. Ця гнучка структура також дає змогу створювати складні програми, якщо використовувати її разом з іншими допоміжними бібліотеками.

React має односторонню прив’язку даних, що означає, що його структура переходить від батьківського до дочірнього компонентів. Однак для

двостороннього зв’язування даних React пропонує LinkedStateMixin, який встановлює загальний шаблон циклу потоку даних. Оновлення React максимально прості. У традиційному потоці даних для кожного нового введення даних потрібно було перезавантажувати всю сторінку, щоб переглянути зміни. У React перезавантажувати не потрібно. Причина цього полягає в тому, що React не створює жодних додаткових об’єктних моделей документів (DOM), як це робили б традиційні потоки даних, приймаючи нові оновлення даних. Він використовує віртуальну копію реальної DOM, що значно спрощує процес розробки та підвищує швидкість функціонування додатків.

До важливих переваг використання React слід віднести наступні:

1. Легкість навчання. Простота навчання React є справді однією з його основних переваг. Бібліотеку порівняно легко освоїти та впровадити, тому підприємства можуть швидко розпочати роботу. Для React не потрібно володіти JavaScript на високому рівні.

2. Широка професійна спільнота. React безумовно має сильну підтримку спільноти, що можна пояснити його природою з відкритим кодом. Незалежно від того, чи зіткнулися розробники з проблемою, помилкою чи чимось іншим, вони впевнені, що спільнота завжди буде поруч, щоб надати вам будь-яку допомогу.

3. SEO-дружність. Бібліотека React є дружньою до SEO та зосереджена на швидкості рендерингу. Веб-сайти на основі JavaScript можуть мати деякі проблеми через повторне відтворення DOM. Тривалий час завантаження також може бути причиною проблем із самим веб-сайтом. Завдяки віртуальній DOM і відображенню на стороні сервера це більше не є проблемою, оскільки Google або будь-якій іншій пошуковій системі не потрібно використовувати JavaScript безпосередньо для відтворення вмісту.

4. Зменшене кодування. У React подібний код як для клієнтської, так і для серверної сторони програми. Ось чому будь-який веб-сайт із React має

переваги високої швидкості, що робить його привабливим для сканерів, користувачів і розробників.

5. Підтримка Facebook. Facebook сам по собі є величезною перевагою React. Не лише група окремих розробників, окремі спільноти, але й сам Facebook підтримує цю структуру.

6. Багаторазові компоненти. Вони є одними з факторів, які впливають на популярність React. Вони є однією з важливих функцій React, коли йдеться про розробку веб-додатків, та значно прискорюють весь процес розробки. Усі компоненти програми React є багаторазовими та відповідають за виведення невеликого багаторазового фрагмента HTML-коду.

7. Підвищення продуктивності. React керує віртуальною DOM. Завдяки цьому розробники можуть створювати швидкі програми, які відповідають сучасним стандартам. Віртуальний DOM копіює існуючий DOM і зберігає кеш-пам'ять, заощаджуючи зусилля, пов'язані з повторним відображенням дерев DOM при оновленні HTML-коду. При зміні стану компонента віртуальна DOM змінює лише цей конкретний об'єкт у справжньому DOM. Це забезпечує плавнішу та швидшу роботу.

8. Підтримка Handy Tools.

9. Чиста абстракція.

10. Кросплатформні функції.

11. Висока сумісність. React створений для безперебійної роботи з іншими бібліотеками JS і не обмежує можливості використання додаткових інструментів для створення сайтів. Це дозволяє вибрати будь-яке відповідне розширення React, яке за допомогою зовнішніх контролерів визначатиме рівень перегляду сторінки. Іноді достатньо попрацювати з React на простих сторінках. Однак чим складніша сторінка, тим більше бібліотек їй потрібно для належного функціонування.

12. Обсяг тестування кодів. Програми React надзвичайно легко тестувати. Він пропонує область, де розробник може тестувати та налагоджувати свої коди за допомогою рідних інструментів. Код React

стабільний, оскільки має низхідний потік даних. Будь-яка зміна дочірніх компонентів ніколи не впливає на материнські компоненти. Це допомагає розробникам легко налагоджувати помилки.

13. Масштабованість.

Незважаючи на те, що React має багато переваг, є деякі недоліки, які слід враховувати у процесі розробки. Найголовніші з них – швидкість розробки, погана документація та спеціалізація інтерфейсу. Однак, не зважаючи на наявність недоліків, React є потужним інструментом веб-розробки. Саме ця бібліотека буде використовуватися для створення універсального веб-інтерфейсу.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЄКТУ

3.1. Бібліотека React Aria

Для реалізації проєкту було обрано технологію, яку пропонує сучасна бібліотека компонентів React. Як було зазначено в попередніх розділах, основна відмінність React полягає у використанні принципу компонентів для побудови веб-додатків та веб-сайтів. Однією з головних задач розробника є створення такого компонента, який можливо використовувати в будь-якому місці коду. Саме цьому за останні роки React став одним із найпопулярніших інтерфейсних фреймворків завдяки своїй компонентній архітектурі та потужним можливостям рендерингу. Однак створення адаптивної та доступної бібліотеки інтерфейсу користувача може бути складним завданням. Але за допомогою правильних інструментів і методів це можливо.

У цій публікації [6] викладено використання бібліотеки React Aria для створення адаптивних і доступних компонентів інтерфейсу користувача в React. Саме ця бібліотека використовується в рамках проєкту. Можливості React Aria дозволяють створювати компоненти інтерфейсу користувача, які будуть доступні для всіх користувачів, у тому числі для людей з обмеженими можливостями. Це один з основних принципів створення універсального інтерфейсу, які були викладені у попередніх розділах.

Бібліотека React Aria являє собою набір React Hooks, розроблений, щоб допомогти розробникам створювати доступні компоненти інтерфейсу користувача за допомогою шаблонів ARIA (Accessible Rich Internet Applications). Вона підтримує різні режими введення, наприклад взаємодію з мишею, дотиком, клавіатурою та програмою зчитування з екрана, а також забезпечує керування фокусом, щоб гарантувати, що користувачі клавіатури та програми зчитування з екрана можуть переходити між компонентами інтерфейсу користувача доступним способом.

Визначимо основні функціональні можливості бібліотека React Aria (рис. 8):

1. Доступність: підтримує навігацію з клавіатури, програму зчитування з екрана та шаблони ARIA для створення компонентів інтерфейсу користувача, доступних усім користувачам.

2. Багатомовність: підтримує понад 30 мов, включаючи двонаправлений текст і локалізоване форматування дати і числа, для створення інтернаціоналізованих компонентів інтерфейсу користувача.

3. Повне налаштування: можна інтегрувати в будь-яку структуру дизайну чи стилю без нав'язування конкретного рендерингу, структури DOM або деталей дизайну.

4. Адаптивність: підтримує різні режими введення, що робить компоненти інтерфейсу користувача адаптованими до різних потреб користувачів і допоміжних технологій.

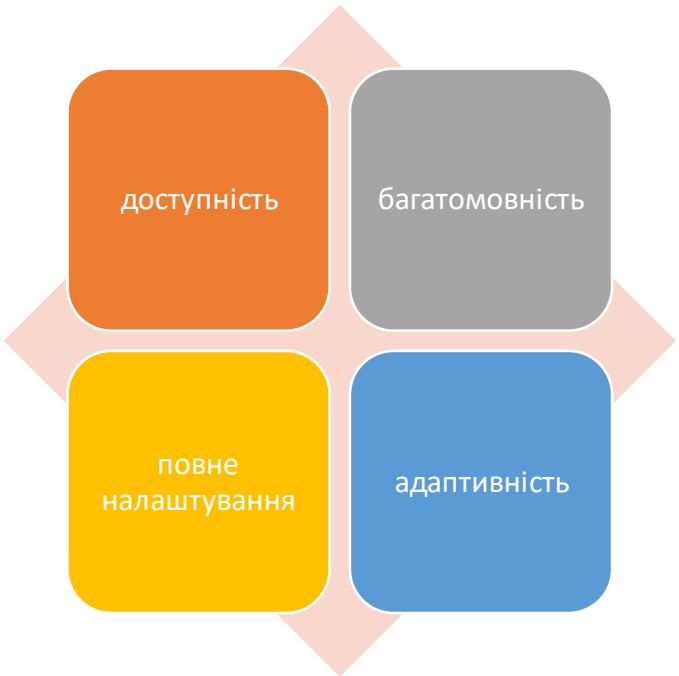

Рисунок 8 – Функціональні можливості React Aria

За допомогою React Aria розробники можуть створювати доступні компоненти інтерфейсу користувача, які використовуються для побудови універсальних інтерфейсів.

3.2. Створення основних елементів дизайну

Наступна бібліотека, яка використовується – це React Spectrum. Вона є бібліотекою компонентів інтерфейсу користувача и розроблена Adobe. Вона надає набір попередньо зібраних, доступних і настроюваних компонентів, створених на основі React Aria. Компоненти React Spectrum створені за допомогою хуків React Aria, що гарантує, що вони доступні за замовчуванням. React Spectrum пропонує широкий спектр компонентів інтерфейсу користувача, включаючи кнопки, прапорці, текстові поля тощо. Бібліотека створена за допомогою React і відповідає сучасним веб-стандартам, таким як доступність, адаптивний дизайн і інтернаціоналізація.

Завдяки використанню функцій доступності, наданих React Aria, компоненти React Spectrum розроблені таким чином, щоб бути доступними за замовчуванням, а його потужна система створення тем дозволяє розробникам легко налаштовувати зовнішній вигляд своїх веб-сторінок.

Щоб розпочати роботу з React Aria, необхідно встановити її за допомогою npm або yarn, командою:

npm install react-aria

або

yarn add react-aria

Після встановлення бібліотеки імпортуємо хуки, необхідні для проєкту. React Aria включає різноманітні хуки та компоненти, такі як, useButton, useCheckbox тощо. Ці хуки додають необхідні атрибути ARIA та події, для

того, щоб зробити компоненти доступними (useSlider useFocusRing useCombobox).

Хук useButton забезпечує доступність і підтримку взаємодії для елементів кнопки. Це гарантує, що кнопка може бути активована за допомогою подій клавіатури, таких як клавіші Enter або Space, а також подій миші та дотику:

```
import { useButton , mergeProps } from "react-aria" ;

function MyButton ( props ) {
let {children,onPress,...otherProps}=props;
let {buttonProps}=useButton({onPress},otherProps);
return
< button { ... mergeProps ( buttonProps , otherProps )}>
{ children }
</ button > ; }
```

Визначили функціональний компонент під назвою MyButton, який використовує useButton хук для створення доступної кнопки. Хук генерує властивості доступності та обробники подій для елемента кнопки. Ці атрибути гарантують, що кнопка доступна з клавіатури та працює з програмами зчитування з екрана (react-aria useButton aria-pressed onPresson KeyUp).

Функція утиліти mergeProps використовується для об’єднання властивостей доступності, згенерованих з будь-якими додатковими атрибутами. Отримані об’єднані атрибути потім поширюються на елемент кнопки за допомогою синтаксису JSX (react-aria useButton MyButton otherProps).

Щоб використовувати MyButton імпортуємо його та відображаємо з потрібними параметрами, включаючи children, onPress та будь-які інші атрибути, які сприймає звичайний елемент кнопки HTML:

```
import { useToggleButton, mergeProps } from "react-aria";
```

```
function ToggleButton(props) {
  const { children, isPressed, onPress, ...otherProps } = props;
  const { buttonProps } = useToggleButton({ isPressed, onPress }, otherProps);
  return (
    <button {...mergeProps(buttonProps, otherProps)}>
      {isPressed ? "ON" : "OFF"} - {children}
    </button>
  );
}

export default function App() {
  const [isPressed, setIsPressed] = useState(false);

  return (
    <ToggleButton
      isPressed={isPressed}
      onPress={() => setIsPressed(!isPressed)}
    >
      Toggle Me
    </ToggleButton>
  );
}
```

Використовуємо хук useToggleButton для керування станом і обробкою подій для кнопки. Передаємо реквізити isPressed і onPress хуку, щоб контролювати стан кнопки, а об’єкт buttonProps, повернутий хуком, поширюється на button-елемент, щоб забезпечити належні атрибути доступності та обробку подій.

Компонент ToggleButton також приймає children-атрибут, який використовується для рендерингу вмісту кнопки, і будь-які інші додаткові атрибути поширюються на елемент button, щоб mergeProps міг переконатися, що вони належним чином об'єднані з результатами buttonProps, які повертає хук. Далі використовуємо ToggleButton-компонент у App-компоненті, де керуємо станом кнопки за допомогою useStateHook.

Доступність є важливою умовою для будь-якої бібліотеки інтерфейсу користувача, оскільки вона гарантує, що всі користувачі можуть отримати доступ і використовувати надані компоненти [6]. Щоб створювати доступні компоненти за допомогою React Aria, необхідно дотримуватися таких наступних правил:

1. Всі компоненти повинні мати мітку або доступну назву.

2. Надання клавіатурної навігації та керування фокусом.

3. Всі компоненти відповідають стандартам WCAG щодо контрастності та кольору.

4. Всі компоненти доступні програмі зчитування з екрана.

React Aria надає кілька хуків і компонентів, які допомагають із доступністю, наприклад useFocusRing. Він забезпечує функцію доступності, відому як «кільце фокусування». Коли елемент отримує фокус, навколо нього відображається кільце фокусування, яке вказує користувачеві, що елемент зараз у фокусі. Використання useFocusRing для забезпечення навігації з клавіатури та керування фокусом у спеціальному компоненті кнопки:

```
import { useState } from "react";
import { useFocusRing } from "react-aria";

export default function CustomInput() {
  const [value, setValue] = useState("");
  const { isFocusVisible, focusProps } = useFocusRing();

  function handleChange(event) {
```

```
    setValue(event.target.value);
  }
  return (
    <div>
      <label>Type something:</label>
      <input
        {...focusProps}
        value={value}
        onChange={handleChange}
        style={{
          boxShadow: isFocusVisible ? "0 0 3px 3px #4D90FE" : "none",
          padding: "5px",
          fontSize: "16px"
        }
        }
      />
    </div>
  );
}
```

У цьому прикладі використовуємо useFocusRing, щоб визначити, коли введення сфокусовано, і застосувати спеціальний стиль фокусування. Використовуючи ці хуки, покращуємо доступність спеціального компонента введення:

```
import CustomInput from "./CustomInput";

function App() {
  return (
    <div>
      <label htmlFor="custom-input">Enter your name:</label>
```

```
      <CustomInput id="custom-input" />
    </div>
  );
}
export default App;
```

У цьому прикладі CustomInput-компонент імпортується з файлу під назвою ***./CustomInput*** та використовується всередині компонента. Елемент асоціюється з компонентом за допомогою власного атрибута та атрибута компонента (CustomInput.js App label CustomInput htmlFor id CustomInput). Це важливо для цілей доступності, оскільки це дозволяє програмам читати мітку з екрана, коли введення виділено (рис. 9).

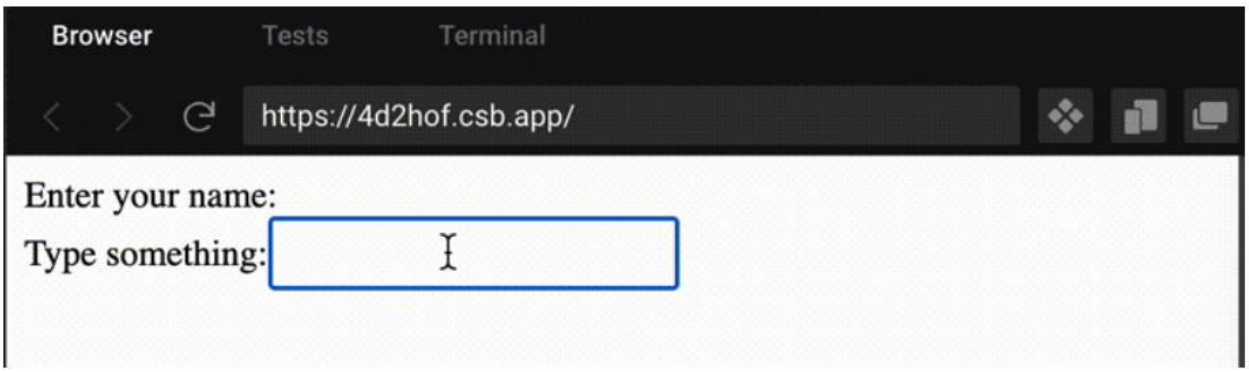

Рисунок 9 – Функціональні можливості React Aria

Адаптивний дизайн є важливим моментом для будь-якої бібліотеки інтерфейсу користувача, оскільки він гарантує, що компоненти добре працюють на різних розмірах екрана та типах пристроїв. Щоб створювати адаптивні компоненти за допомогою React Aria, необхідно дотримуватися наступних правил:

1. Використання адаптивного дизайну для налаштування компонування та поведінки компонентів залежно від розміру екрана.

2. Надання спеціальних можливостей, таких як режим високої контрастності та підтримка масштабування.

3. Використання медіа-запитів CSS для налаштування стилів компонентів на основі розміру екрана.

Налаштування за допомогою React Context дозволяє легко налаштувати поведінку та зовнішній вигляд компонентів всього додатку. React Context надає спосіб передавати дані через дерево компонентів без необхідності передавати атрибути вручну на кожному рівні.

Використовуємо React Context, щоб дозволити споживачам бібліотеки інтерфейсу користувача налаштовувати колірну схему спеціального компонента кнопки:

```
import { useButton, mergeProps } from "react-aria";
import { createContext, useContext } from "react";

let ButtonContext = createContext({ colorScheme: "light" });

export function MyButton(props) {
  let { children, onPress, ...otherProps } = props;
  let { colorScheme } = useContext(ButtonContext);
  let { buttonProps } = useButton({ onPress }, otherProps);
  return (
    <button
      {...mergeProps(buttonProps, otherProps)}
      style={{
        backgroundColor: colorScheme === "dark" ? "black" : "white",
        color: colorScheme === "dark" ? "white" : "black"
      }}
    >
      {children}
    </button>
```

```
  );
}


export function MyButtonProvider(props) {
  let { children, colorScheme } = props;
  return (
    <ButtonContext.Provider value={{ colorScheme }}>
      {children}
    </ButtonContext.Provider>
  );
}
```

Створюємо ButtonContext за допомогою createContext зі значенням за замовчуванням. Далі визначаємо компонент, який споживає за допомогою Hook (colorScheme: "light" MyButton ButtonContext useContext).

У цьому компоненті використовуємо useButton для створення доступної кнопки. Потім об'єднуємо їх за допомогою from. Нарешті, встановлюємо атрибут кнопки для зміни її (react-aria buttonProps buttonProps otherProps mergeProps; react-aria style backgroundColor color colorScheme ButtonContext).

Потім в App-компоненті загортаємо MyButton-компонент у MyButtonProvider-компонент із colorScheme темного кольору. Це змінює вигляд кнопки на темний колір фону та світлий колір тексту:

```
import { MyButtonProvider, MyButton } from "./MyButton";


function App() {
  return (
    <MyButtonProvider colorScheme="dark">
      <div>
        <MyButton onPress={() => console.log("Button pressed")}>
```

```
        Click me!
      </MyButton>
    </div>
  </MyButtonProvider>
 );
}
export default App;
```

Використовуючи React Context, легко налаштувати вигляд компонентів у всій програмі, просто змінивши значення в компоненті Provider (рис. 10).

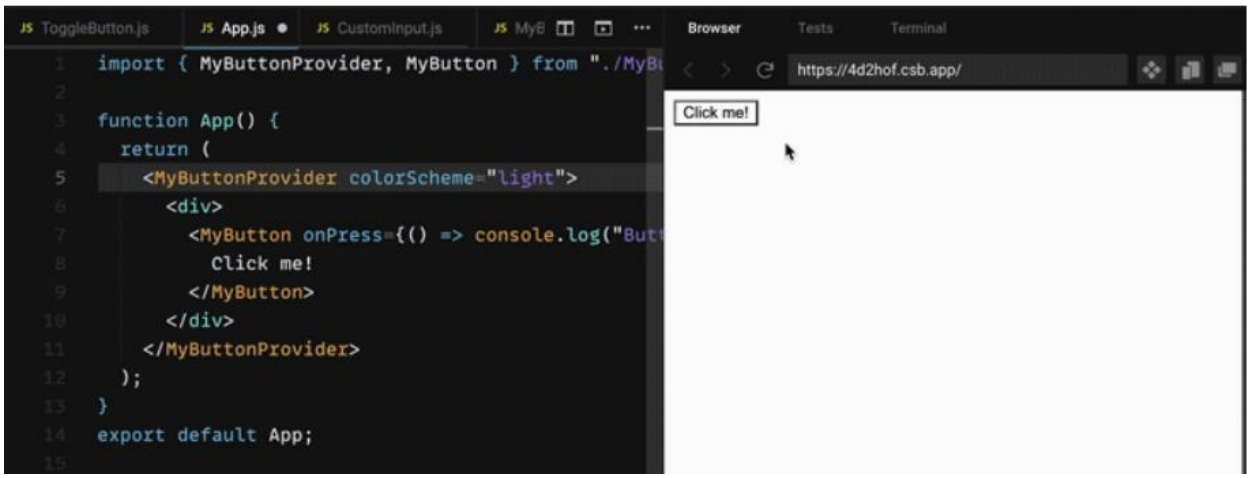

Рисунок 10 – Функціональні можливості React Aria

3.3. Інструменти тестування та документування компонентів

Тестування на доступність має вирішальне значення під час створення бібліотеки інтерфейсу користувача, оскільки воно гарантує, що компоненти, які надаються, доступні для всіх користувачів. Існує багато доступних інструментів тестування доступності, таких як axe-core і pa11y. Щоб протестувати компоненти, використовуємо ці інструменти в поєднанні з бібліотеками тестування React, такими як rest-testing-library, Jest або Enzyme.

Щоб зробити бібліотеку інтерфейсу користувача доступною для інших, важливо надати документацію та приклади. Використовуємо інструмент Storybook, щоб створити посібник зі стилю кодування, який документує компоненти та показує, як ними користуватися. Включаємо документацію та приклади на свій веб-сайт або у файл README свого сховища.

Приклад надання документації для власного спеціального компонента кнопки:

># MyButton

A customizable button component.

Props

| Prop | Type | Description | Default Value |

| ------ | --------- | ---------- | ---------- |

| onPress | function | Function to call when button is pressed | none |

| colorScheme | string | The color scheme for the button (either 'light' or 'dark') | 'light' |

Example

```jsx

```
import { MyButton } from "my-ui-library";

<MyButton onPress={() => console.log("Clicked!")}>Click me</MyButton>
```

ВИСНОВКИ

В роботі було розглянуто структурні елементи веб-інтерфейсу, визначені основні технології веб-розробки, обрана бібліотека для побудови універсального веб-інтерфейсу, представлені основні React-компоненти, які є основою структури дизайну.

На першому етапі була проаналізована предметна область, визначена структура та функції веб-інтерфейсу, описані його основні елементи та їх особливості. Докладно розглянуті принципи побудови універсального веб-інтерфейсу з метою визначення вимог до проєкту. Ці принципи були створені, щоб керувати широким спектром дизайнерських дисциплін, включаючи середовища, продукти та комунікації.

Після докладного аналізу принципів побудови універсального дизайну визначили інструменти та ресурси, необхідні для його реалізації. У веб-розробці пошук найбільш підходящих фреймворків із функціями та перевагами, необхідними для прискорення та покращення зусиль розробників, є важливим завданням. Тому спочатку були проаналізовані технології веб-розробки: використання систем управління контентом або використання фрейворків/бібліотек. Аналіз показав, що використання фреймоворків є ефективним рішенням для створення універсального веб-інтерфейсу. У якості бібліотеки було обрано React, як таку, що відповідає всім вимогам до поставлених завдань.

На третьому етапі описані функціональні можливості бібліотеки, яка використовується для розробки, а саме, React Aria. Представлені основні компоненти, які відповідають за функціонування інтерфейсу, компоненти тестування та документування.

Подальші дослідження можуть бути пов’язані з удосконаленням існуючих компонентів з метою реалізації всіх принципів універсального дизайну.

ПЕРЕЛІК ПОСИЛАНЬ

1. User Interface Design Basics / https://www.usability.gov/what-and-why/user-interface-design.html (дата звернення 12.04.2023).

2. Learn to Create Accessible Websites with the Principles of Universal Design / https://www.interaction-design.org/literature/article/learn-to-create-accessible-websites-with-the-principles-of-universal-design (дата звернення 12.04.2023).

3. Trello brings all your tasks, teammates, and tools together / https://trello.com (дата звернення 12.04.2023).

4. Content management systems / https://www.nibusinessinfo.co.uk/content/content-management-systems (дата звернення 28.04.2023).

5. UML Use Case Diagram Tutorial / https://www.lucidchart.com/pages/uml-use-case-diagram (дата звернення 28.04.2023).

6. Building an adaptive, accessible UI library with React Aria / https://blog.logrocket.com/building-adaptive-accessible-ui-library-react-aria/ (дата звернення 28.04.2023).