Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

Пояснювальна записка

до бакалаврської дипломної роботи на тему:

«Програмне забезпечення тестування студентів з можливістю самотестування»

Виконав: студент 2 курсу, групи 1КІ-20мс
спеціальності 123 — Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)
_____ Хмель С.А.
(прізвище та ініціали)
Керівник к.т.н., доцент
_____ Черняк О. І.
(прізвище та ініціали)
«17» 06 2022 р.
Рецензент зав. кафедри МБІС д.т.н. професор
_____ Яремчук Ю. Є.
(прізвище та ініціали)
«20» 06 2022 р.

Допущено до захисту
Зав. кафедри ОТ, д.т.н., проф. Азаров О. Д.
«21» 06 2022 р.

Вінниця ВНТУ — 2022 рік

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

Освітній рівень — перший (бакалаврський)

Спеціальність — 123 Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ, д.т.н, проф.
Азаров О.Д.
"09" 02 2022року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Хмель Сергію Анатолійовичу

1 Тема роботи «Програмне забезпечення тестування студентів з можливістю самотестування»

Керівник роботи Черняк Олександр Іванович , к.т.н., доц.,

затверджені наказом вищого навчального закладу від "24"березня 2022року №66

2 Термін подання студентом роботи 14.06.22.

3 Вихідні дані до роботи: технічний опис програмного застосунку, мова програмування JavaScript з використанням Framework React, веб додаток, сервер обробник даних Firebase.

4 Зміст розрахунково пояснювальної записки (перелік питань, які потрібно розробити): вступ, аналіз сучасних методів розробки, варіантний вибір засобів для розробки веб додатків на мові javaScript, розробка структури та алгоритмів роботи веб-додатку, програмна реалізація, тестування розробленої платформи.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): блок-схема алгоритму роботи сторінки авторизації.

6 Консультанти розділів роботи наведені в таблиці 1.

Таблиця 1 – Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-3	Черняк О.І., к.т.н., доц.		

7 Дата видачі завдання 24.03.22.

8 Календарний план наведено в таблиці2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів проекту	Підпис
1	Постановка задачі роботи	09.01.22	вик.
2	Пошук матеріалів по технологіям розробки веб-додатків на мові JavaScript	10.02.22 —25.02.22	вик.
3	Структурне проектування додатку тестування студентів з можливістю самотестування	26.02.22 — 28.02.22	вик.
4	Обгрунтування та вибір засобів реалізації веб додатку	29.02.22 — 04.03.22	вик.
5	Розробка головного головної сторінки, сторінки створення тесту, авторизації	05.03.22 —29.04.22	вик.
6	Підготовка матеріалів та розробка алгоритму збереження та обробки даних	30.03.22 —27.04.22	вик.
7	Оформлення пояснювальної записки та ілюстративного матеріалу	28.04.22 — 01.06.22	вик.
8	Перевірка якості виконання бакалаврської роботи та усунення недоліків	07.06.22	вик.

Студент Хмель Сергій Анатолійович

Керівник роботи Черняк Олександр Іванович

АНОТАЦІЯ

Дану бакалаврську дипломну роботу присвячено створенню веб додатка для тестування студентів за допомогою фреймворку React на мові програмування JavaScript. Також розглянуто питання веб дизайну та тестування додатку.

Результатом проведення досліджень є розроблений комплексний додаток, який містить авторизацію, сторінку тестування і форму створення тестів.

Дана розробка є доцільною для впровадження та використання навчальним закладом.

Ключові слова: React, Redax, JavaScript, App Tester, Quiz.

ABSTRACT

This bachelor's thesis is devoted to the creation of a Web – application for testing students using Framework React in the Java Script programming language. Web design issues are also discussed.

The result of the research is a comprehensive application that contains an authorization, a test page, and a form for creating tests.

This development is appropriate for implementation and use by educational institutions.

Keywords: React, Redax, JavaScript, App Tester, Quiz.

ЗМІСТ

ВСТУП

Для якісної освіти студентів потрібно найсучасніші сервіси для навчання. Якісне і продуктивне навчання потребує мотивування студентів до навчального процесу, а також потребує якісної перевірки знань. Одним із видів якісного навчального процесу є тестування, яке дуже важливо залучати на ранніх етапах підготовки студентів: на лекціях, контрольних роботах, а також на заняттях з лабораторних робіт. Рішенням є додаток «Tester App».

Додаток представляє собою крос браузерну веб платформу розроблену за допомогою сучасної мови програмування JavaScript за допомогою Framework React та Redax. Доступ до серверу надає платформи розробки мобільних та веб застосунків «Firebase».

Спочатку розробляється дизайн інтерфейсу та його зручність у використанні. Це потрібно, щоб додаток був красивим і в той же час зручним і зрозумілим користувачу. В процесі розробки був створений алгоритм веб додатку для тестування студентів з можливістю самотестування. Даний алгоритм полягає в тому, що викладач заходить на головну сторінку додатку, де потрібно авторизуватись, щоб потрапити в особистий кабінет. Після авторизації він автоматично перенаправляються на сторінку «список тестів» на якій відображаються всі тести створені ним раніше. Далі в боковому меню йому стає доступна вкладка «створення тестів», перейшовши на яку є можливість створення і публікації тестування для студентів. Далі тест стає доступним для всіх користувачів додатку. Дії студентів на сайті дуже прості: вони переходять на сторінку всіх тестів, обирають тест, і проходять його. В результаті пройденого тесту, на сторінці відображається кількість відповідей у відношенні до кількості питань, які для захисту інформації вказуються лише співвідношенням вірних відповідей до невірних. Результатом таких процесів є те, що студент може проходити тест для самоперевірки, а також для оцінювання викладача.

Базуючись на розглянутому, розробка яка полягає у необхідності вирішення проблеми швидкого та зручного доступу до тестування з можливостю самотестування, дозволяє отримати додаток для вирішення даної проблеми, на основі якого є можливість доступу до самотестування студента в зручний час є **актуальною задачею.**

Метою роботи є розробка додатку для тестування студентів з можливість самотестування на мові програмування JavaScript з використанням framework React та Redax, який дозволяє, авторизованому викладачу створювати тести, а студенту проходити їх.

Задачі дослідження бакалаврської роботи:

— проаналізувати методи та технології розробки додатків на JavaScript, React та Redax;

— проаналізувати існуючі аналоги та їх функціонал;

— проаналізувати та обрати засоби реалізації даного типу додатку;

— виконати проектування додатку;

— розробити додаток для тестування студентів з можливістю самотестування за допомогою фреймворку React і Redax для подальшого використання;

— створити систему для можливості авторизуватись;

— підключити сервер обробник даних;

— протестувати розроблений додаток.

Об’єкт дослідження — процес створення додатку для крос браузерної платформи з можливість тестування та самотестування студентів.

Предмет дослідження — додаток, що використовується в школах, навчальних закладах, та установах для можливості тестування і самотестування користувачів.

Методи дослідження бакалаврської роботи: у роботі проведено дослідження принципів програмування на мові JavaScript з використанням

фреймворку React і Redax для реалізації запропонованого підходу до реалізації таких додатків.

Апробація результатів бакалаврської роботи: зроблено доповідь на всеукраїнську науково-практичну інтернет-конференцію «Молодь в науці: дослідження, проблеми, перспективи».

Практичне значення отриманих результатів полягає в можливості використовування розробленої системи для створення тестів та тестування користувачів в різних установах.

1 АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБКИ ВЕБ–ДОДАТКІВ НА МОВІ ПРОГРАМУВАННЯ JAVASCRIPT

Головною ідеєю роботи є створення додатку для тестування студентів з можливісю самотестування.

1.1 Порівняння популярних фреймворків для створення веб–додатків на мові програмування JavaScript

Фреймворки JavaScript існують, щоб допомогти нам створювати програми зі схожою функціональністю, використовуючи загальний підхід. Існує багато фреймворків для реалізації інтерфейсу користувача, але найпопулярнішими серед них є React і Angular.

Розробка за допомогою фреймворку React — це опис того, що вам потрібно відобразити на сторінці, а не складання інструкцій для браузера, як це зазвичай робиться. Крім усього іншого, це означає значне зменшення кількості коду в шаблоні.

Angular, з іншого боку, має інструменти командного рядка для створення коду компонентів. Це трохи застаріло, ніж можна було б очікувати від сучасного інструменту розробки інтерфейсу. Насправді, Angular має стільки шаблонного коду, що навіть був створений спеціальний інструмент для його отримання.

У React, щоб почати розробку коду, просто потрібно розпочати кодувати. Немає коду компонента шаблону, який потрібно якимось чином згенерувати. Звичайно, перед розробкою необхідно провести певну підготовку. Тим не менш, при підході до розробки не потрібно вивчати принципово нові речі, щоб почати створювати код на React у вигляді чистих функцій.

Вибраний фреймворк повинен мати чіткий синтаксис. Це потрібно для того, щоб не вивчати щось нове, а базуючись на знаннях мови JavaScript можна зразу приступити до розробки. Зрештою фреймворки і придумали для того, щоб облегшити роботу. Чіткий синтаксис є важливим фактором, який слід враховувати при виборі UI фреймворка. У зв’язку з цим важливо зазначити, що React має

менше абстракцій, ніж Angular. Якщо ви знаєте JavaScript, ви ймовірно, зможете навчитися писати програми React за день. Звичайно, знадобиться деякий час, щоб навчитися робити це правильно, але ви можете почати вже зараз.

Приклад синтаксису фреймворку React наведено на рисунку 1.1.

```
const Greetings = ({ firstName }) => (
   <div>Привіт, {firstName}</div>
);
```

Рисунок 1.1 — Синтаксис фреймворку React

В фреймворків є особливості механізму прив'язки даних, мають певні розбіжності. Angular має двосторонню систему зв'язування даних. Це проявляється, наприклад, як зміни форми елемента, що призводять до автоматичного оновлення стану програми. Це ускладнює налагодження і є великим недоліком фреймворку. Використовуючи цей підхід, якщо щось піде не так, програміст не може точно знати, що викликало зміну стану програми.

React, з іншого боку, використовує одностороннє прив’язування даних. Це велика перевага цієї бібліотеки, оскільки програміст завжди знає, що саме викликало зміну стану програми.

У React функціональний підхід до розробки, тому одна з сильних сторін React це те, що ця бібліотека не змушує розробників використовувати класи. В Angular всі компоненти повинні бути реалізовані в класах. Це призводить до надмірного ускладнення коду без користі.

У React всі компоненти інтерфейсу можна представити як чисті набори функцій. Використання чистих функцій для формування UI можна порівняти з ковтком свіжого повітря.

Кожен інструмент хороший в чомусь своєму. Але, якщо заглиблюватися, між Angular і React то React є кращим вибором.

Як висновок можна відзначити, що сучасні фреймворки можуть однаково покрити завдання полегшення роботи, В сучасному світі є багато хороший рішень, але все із перерахованого вище демонструє, що Реакт виграє майже за всіма показниками.

1.2 Аналіз сучасних сервісів для тестування студентів

Перед створенням вимог до проектування програмного забезпечення для тесутвання студентів з можливістю самотестування у вигляді веб–додатку, слід проаналізувати існуючі програми, які реалізують можливість тестування студентів. На ринку в інтернеті дуже багато такий веб–додатків які здебільшого мають вигляд веб сайтів.

Лідерами на ринку України є — Google Classroom, Google Форми та Madtest.

Google Classroom — це віртуальний клас із можливістю додавання та вибору завдань (планування). За допомогою нього можна покращувати оцінки, вибирати літери, тестувати форми Google, публікувати документи. Інтерфейс Google Classroom простий та зрозумілий.

Про потреби навчання можна повідомити класу, використовуючи особистий код, який буде автоматично імпортований зі шкільного веб сайту.

Приклад інтерфейсу Google Classroom наведено на рисунку 1.2.

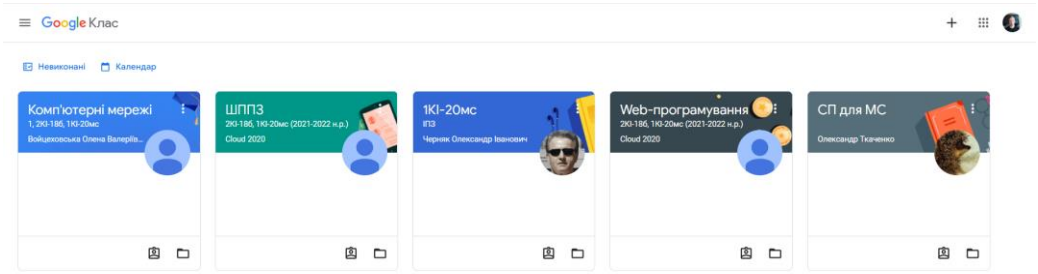

Рисунок 1.2 — Інтерфейс Google Classroom.

Google Forms — інструмент для створення тестів із можливістю визначення кількості балів за активність та коригування результатів. Є можливість автоматичного повторного сканування, але додатково, якщо потрібно повторне

сканування, можна виконати частину сканування вручну і потім побачити результат. У цьому режимі можна надіслати результати на вказану адресу поштою.

Щоб створити тест в Google Форми:

— відкрити форму в Google Forms;

— у верхній частині форми натиснути кнопку «Настройки»;

— увімкнути Make this a quiz;

— щоб зібрати адреси електронної пошти, поруч з пунктом "Відповіді" натисніть стрілку «вниз» вона включіть «Збір адрес» електронної пошти.

Приклад інтерфейсу створення тесту в гугл формі приведено на рисунку 1.3.

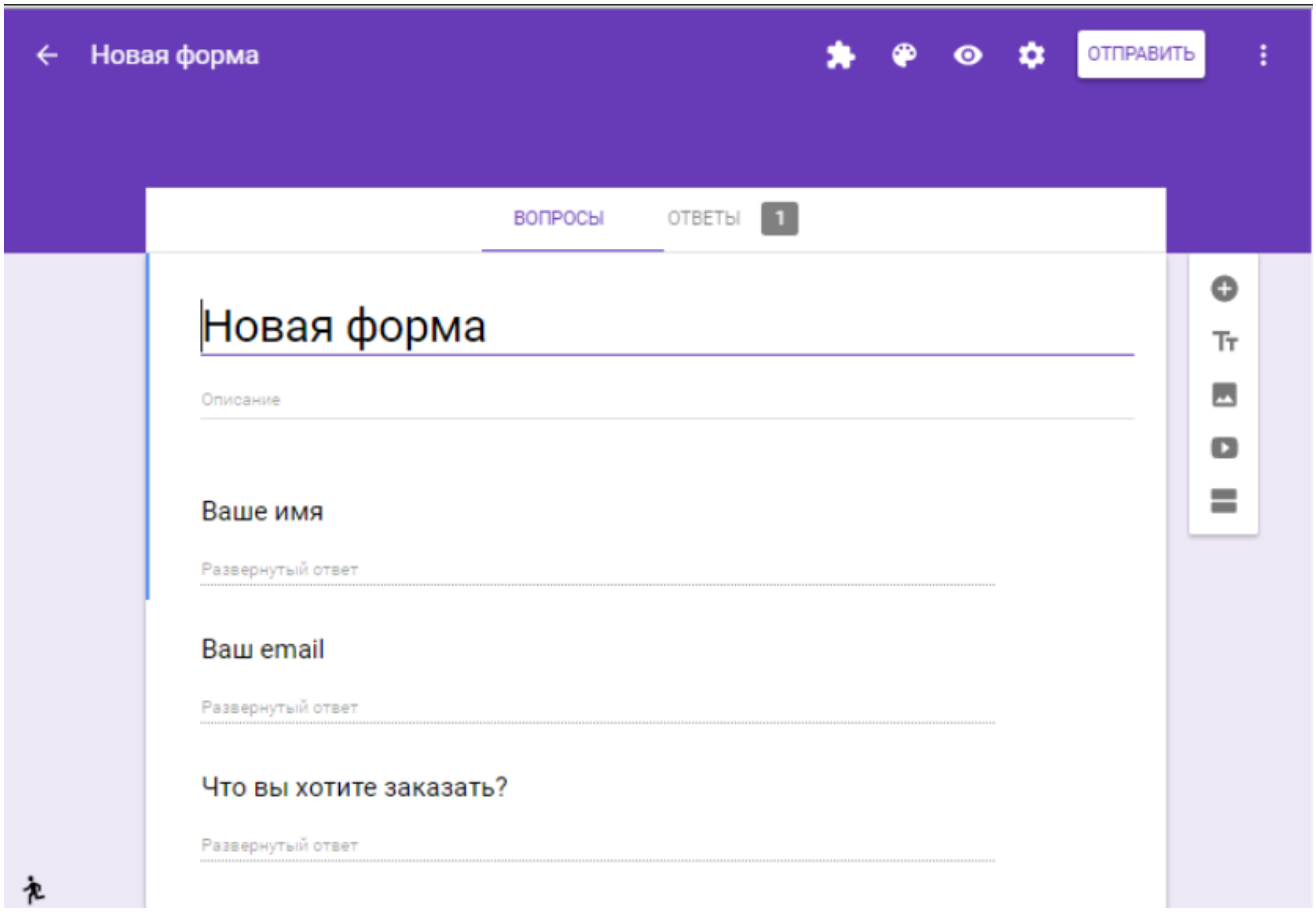

Рисунок 1.3 — Інтерфейс створення тесту в гугл формі

Madtest — веб сайт для створення тестів на вікторину та опитування. Є можливість створити тести на веб сайті Meduza.io, більше підходить для

тестування аудиторії для дорослих, для засобів масової інформації та тих, хто розміщує тести та опитування на своєму веб сайті.

Можливості:

— Кілька форматів - це тестова вікторина та особистий тест;

— Зручний і зрозумілий редактор;

— Персоналізовані результати;

— Великий вибір етапів на сторінках результатів, посилання на миттєвих месенджерів, вибір карт, вибір товарів;

— Можливість брендингу тестів;

— Різні типи відповідей;

— Коментарі до відповідей;

— Багато аналітики, яку послуга автоматично збирає;

— Створені тести можна проводити як за прямим посиланням, так і розміщувати на сайті.

Приклад багаторівневої архітектури приведено на рисунку 1.4.

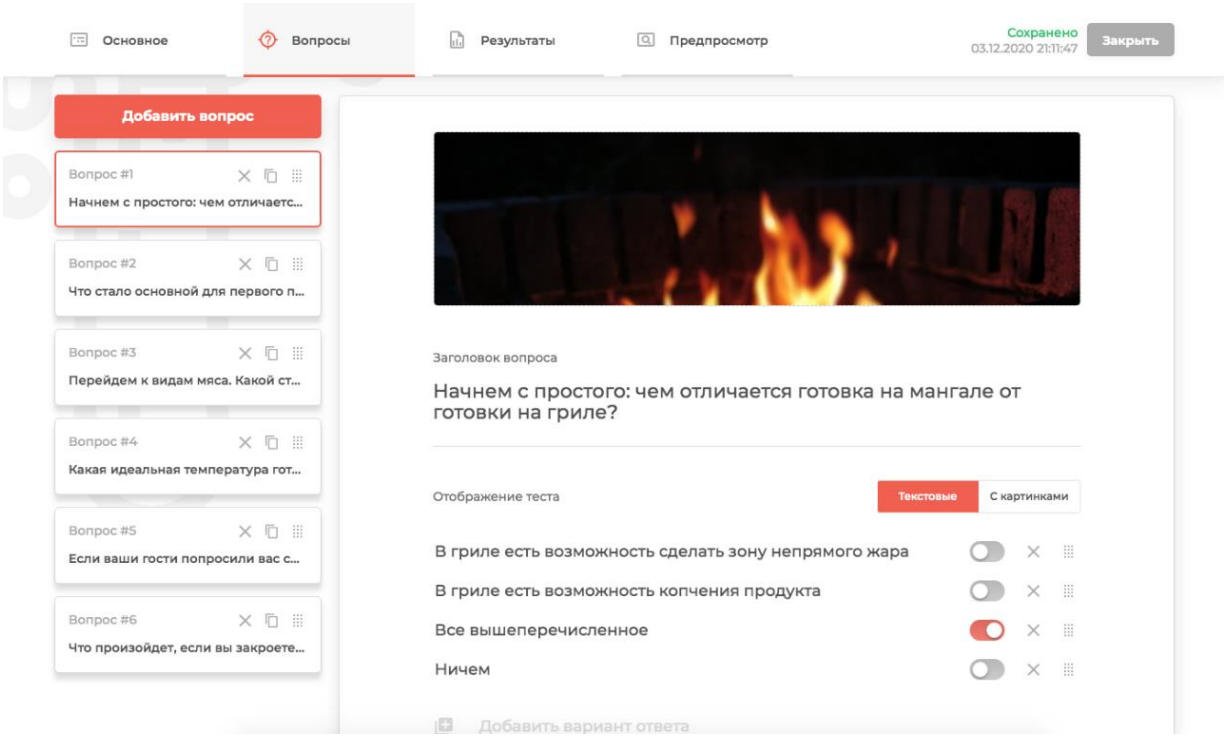

Рисунок — 1.4 Сторінка тестів

1.3 Аналіз мови програмування JavaScript

Використання JavaScript завжди дуже корисно для веб дизайну. Це дозволяє динамічно генерувати контент індивідуально для кожного відвідувача. Вміст завантажується автоматично за допомогою AJAX. Можна, також створювати анімацію, яка здатна на порядок підвищити ефективність вашого сайту.

JavaScript був розроблений, щоб додати до веб сторінок пеного живого наповнення. Зробити його більш рухливим та динамічним. На даній мові розробляють за допомогою скриптів. Дані скрипти підключаються до сторінки з індексом html та виконують написані дії при завантаженні сторінки.

Веб додаток чи сайт написаний на JavaScript — це по факту звичайний текст. Для читання коду не потрібно якогось особливого методу читання інформації. Це самий головний плюс даної мови, і це його відрізняє мід інших мов. Щоб зчитувати текст на JavaScript та запускати його, не потрібна спеціальна програма, для цього просто необхідно відкрити сторінку в браузері. Процес виконання скрипту називають «інтерпретацією».

Можливості використання JavaScript особливо цим не обмежені. Це мова програмування є повноцінною і самодостатньою, код можна запускати що на сервері, що на локальні, і навіть у пральній машинці, якщо вона використовує інтерпретатор, або хоча б в неї є браузер.

Можливості JavaScript необмежені, ця мова програмування є дуже легка і швидка для опанування мова програмування, призначення якої є пришвидшення і оптимізації автоматизації на сайті. По суті вона є мозком і нервовою системою всього сайту.

JavaScript не має рівних в роботі з браузерним середовищем, там його стихіями, там він вміє робити все, розпочинаючи з маніпуляції зі сторінкою, закінчуючи взаємодією з ДОМ елементами та сервером. Деякі сервери навіть написані на мові програмування JavaScript.

Перелік вмінь мови JavaScript:

— взаємодіяти з HTML — тегами, додавати та редагувати стилі, скривати елементи, видаляти та взаємодіяти з веб сторінками сайту, тощо;

— взаємодіяти з кукі файлами та локальним середовищем;

— обробляти натиснення, взаємодіяти з пікселями, прораховувати позицію скролу, та динамічно завантажувати сторінки;

— динамічно переходити між сторінками браузера, без перезавантаження сторінки;

— працювати з "AJAX" і посилати в вигляді Json коду запроси, отримувати відповіді, та форматувати отримані дані.

JavaScript в використанні необмежений. За допомогою фреймворків, він здатний потягатись за місце в трійці найкращих інструментів для розробки веб додатків.

Він є безпечним, і його скомпільований код не дає змоги прочитати дані користувачі. У браузері Firefox існує спосіб «підпис» скриптів з метою обходу частини обмежень, але він нестандартний і не крос браузерні.

Запуск JavaScript виконується не в браузері, а в хмарному середовищі, прикладом якого є сервер. JavaScript в браузері обмежений відкрити вікном чи сторінкою, але за допомогою АПІ він взаємодіє з сервером і запускається на ньому, що дає можливість асинхронних дій і запуск багатьох процесів одночасно.

JavaScript має певні обмеження, а інколи цілком не може читати чи записувати файли, так в нього немає змови запускати чи копіювати веб сайти чи програми з локального комп’ютера. Він не може достукатись до вузла операційної системи, та записати якісь дані в операційну пам'ять. Всі його дії обмежені браузерним середовищем.

Сучасні браузери можуть працювати з файлами, але ця можливість обмежена, спеціально виділеної директорією, яку ще називають пісочницею.

JavaScript, може працювати в рамках одної вкладки, доступу до зовнішніх вкладок в нього немає. І навіть щоб працювати в рамках вкладки, його потрібно підключити і запустити функцією. За винятком того, що JavaScript сам взяв і

запустив дане вікно, або інші декілька вкладок, так таке він вміє і може якщо в сторінок однаковий домен. Для того, щоб уникнути цього потрібно як мінімум згоду обох сторін. Просто так взяти і залізти в довільну вкладку з іншого домену не можливо.

JavaScript обробляє посилання посилати запитів на серверну частину веб сайту. Запит на домен не є зручним і безпечним.

Унікальність JavaScript полягає в тому, що є три чудових особливості JavaScript:

— Повна інтеграція з HTML / CSS;

— Прості речі робляться просто;

— Підтримується всіма браузерами;

— Включений за замовчуванням.

Цих трьох речей одночасно немає більше ні в одній браузерні технології. Тому JavaScript і є найпоширенішим засобом створення браузерних інтерфейсів.

Перед тим, як розпочати вивчати JavaScript, корисно ознайомитися з її розвитком і перспективами.

HTML 5 — еволюція стандарту базової розмітки, що дає можливість використовувати теги для платування планування таблиці, і дає можливість взаємодіяти з браузером.

Ось кілька прикладів:

— читання, редагування, та записування даних чи файлів на локальний диск чи хмарне середовище;

— можливість запуска кількох процесів одночасно, та можливість багато поточного запиту;

— можливість використовувати браузерне сховище та його локальну базу даних для збереження;

— відтворення відео та аудіо.

Сама мова JavaScript поліпшується. Сучасний стандарт EcmaScript 5 включає в себе нові можливості для розробки.

Сучасні браузери покращують свої движки, щоб збільшити швидкість виконання JavaScript, виправляють баги і намагаються слідувати стандартам.

Дуже важливо те, що нові стандарти HTML5 та ECMAScript зберігають максимальну сумісність з попередніми версіями. Це дозволяє уникнути неприємностей з вже існуючими додатками.

Втім, невелика проблема з HTML5 все ж є. Іноді браузери намагаються включити нові можливості, які ще не повністю описані в стандарті, але настільки цікаві, що розробники просто не можуть чекати.

Однак, з часом стандарт змінюється і браузерам доводиться підлаштовуватися до нього, що може призвести до помилок у вже написаному (старому) коді. Тому слід двічі подумати перед тим, як застосовувати на практиці такі «супер-нові» рішення.

При цьому всі браузери сходяться до стандарту, і відмінностей між ними вже набагато менше, ніж всього лише кілька років тому.

Серед недоліків JavaScript, найчастіше, недоліки підходів і технологій — це зворотна сторона їх корисності.

У JavaScript, однак, є цілком об'єктивні недоробки, пов'язані з тим, що мова, за висловом його автора (Brendan Eich) робилася «за 10 безсонних днів і ночей». Тому деякі моменти продумані погано, є і відверті помилки (які визнає той же Brendan).

В більшості випадків збільшення часу завантаження, потрібне для того, щоб javascript завантажився в браузері відвідувачів, спочатку він повинен завантажитися на їх комп'ютери. Це збільшує час, необхідний на завантаження вашої веб сторінки.

Але, якщо у вас досить швидкий інтернет-зв'язок, цей недолік може здатися незначним. Файли javascript в більшості своїй невеликі, і враховуючи можливості сучасного Інтернету, будуть завантажені менш, ніж за секунду.

Збільшення часу очікування, після того, як ваш сайт з елементами javascript завантажиться на комп'ютер вашого відвідувача, комп'ютеру знадобиться деякий

час для виконання вашого коду. Якщо користувач використовує потужний сучасний комп'ютер, то його браузер, ймовірно, може виконати javascript швидко і легко.

В CSS доступні різні способи реалізації. Так, як CSS реалізована на різних комп'ютерах і в кожному браузері по різному, javascript виконується в залежності від технічних можливостей, наявних у відвідувача сайту. Це може призвести до того, що ваш веб–сайт може виявитися для багатьох користувачів просто непрацездатним.

У нових версіях JavaScript (ECMAScript) ці недоліки поступово прибирають. Процес впровадження нешвидкий, в першу чергу через старих версій IE, але вони поступово вимирають. Сучасний IE в цьому відношенні незрівнянно краще

1.4 Аналіз розробки за допомогою фреймворка React та Redax

React — бібліотека JavaScript з доступним до коду для створення та розробки веб додатків, сайті, та додатків, використовуючи методологію та заготовлені скрипти. Розробка веб за допомогою компонентного підходу до розробки з такими фреймворками, як Next.js. Однак React робить лише управління станом і його прямим відправленням до DOM таким чином для такого підходу React потрібно використовувати додаткові бібліотеки, а також певних функцій які потрібно запустити зі сторони клієнтів.

Приклад використання React в HTML з JSX наведено на рисунку 1.5.

```
import React from "react";

const Greeting = () => {
  return (
    <div className="hello_world">
      <h1> Привіт, світ! </h1>
    </div>
  );
};
```

Рисунок 1.5 — Приклад використання React

Greeting — функція являє собою компонент React, який відображає знамениту ввідну фразу "Привіт, світ".

При відображенні в веб–браузері результатом буде рендеринг:

<h1>Привіт, світ!</h1>

React дотримується парадигми декларативного програмування. Розробники розробляють подання для кожного стану програми, а React відновлює і візуалізує компоненти при зміні даних. Це контрастує з імперативним програмуванням[9].

Код React складається з сутностей, званих компонентами. Ці компоненти є багаторазовими і повинні бути сформовані в папці SRC згідно з регістром Pascal в якості угоди про іменування (capitalize camelCase). Компоненти можуть бути візуалізовано для певного елемента в DOM за допомогою бібліотеки React DOM. При рендерінгу компонента можна передавати значення між компонентами через "props"

```
import React from "react";
import Tool from "./Tool";
const Example = () => {
return (<><div className="app">
    <Tool name="Gulshan" />
    </div></>); };
export default Example;
```

У наведеному вище прикладі «name» властивість зі значенням «Gulshan» передано від Examplекомпонента до Tool компоненту. Крім return того, розділ загорнутий в тег, тому що return функції є обмеження, вона може повертати тільки одне значення. Таким чином, всі елементи і компоненти JSX пов'язані в один тег.

Для оголошення компонентів в фреймворку React використовують функціональні компоненти які написані за допомогою класів.

Функціональні компоненти це компоненти функцій які оголошуються за допомогою функції, дана функція в відповідь відправляє JSX файл.

```
const Greeter = () => <div>Hello World</div>;
```

Компоненти які написані за допомогою класів оголошуються за допомогою класів. Клас ParentComponent розширює React.Component {

```
state = { color: 'green' };
render() {
  return (
    <ChildComponent color={this.state.color} />);}}
```

Там, де компоненти класу пов'язані з використанням класів і методів життєвого циклу, функціональні компоненти мають хуки для управління станом і інших проблем, що виникають при написанні коду в React.

Ще однією примітною особливістю є використання об'єктної моделі віртуального документа, або віртуального DOM. React створює копію локальних даних кешу, обробляє отримані дані стану, і потім вносить правки до DOM браузера в відповідності до наданих змін в коді. Це дозволяє писати код, щоб ніби при кожній зміні візуалізується вся сторінка, в той час як фреймворк React візуалізують тільки ті компоненти, які дійсно змінюються. Такий рендеринг дає змогу значно підвищити продуктивність. Це економить час та певні зусилля для перерахунку стилю в CSS, і рендеринга всієї сторінки сайту.

Є певні методи для забезпечення життєвого циклу:

— методи життєвого циклу для компонентів на основі класів використовують форму прив'язки, яка дозволяє запускати код в заданих точках протягом усього терміну служби компонента;

— shouldComponentUpdate допомагає не допустити додатковий рендеринг компонента повертаючи false, якщо не потрібно виконувати;

— componentDidMount запускається або викликається один раз після того як компонент уже скомпільований. Це пришвидшує завантаження сторінки та дає змогу усунути конфлікти при рендерингу;

— componentWillUnmount викликається перед тим, деякий компонент буде зірваний він використовується для очищення ресурсномістких залежність компонента, які не будуть просто видалені при ррозвиртанні компонента

(наприклад, видалення будь-яких setInterval()примірників, пов'язаних з компонентом, або "EventListener", встановленого в "документі" із-за присутності компонента);

Render — це найважливіший метод життєвого циклу і єдиний необхідний для використання в компонентах, який зазвичай викликається коли оновлюється стан компонента, що має бути відображено в інтерфейсі.

Для розширення JavaScript використовується JSX, він розширює синтаксис, змінює його, для того, щоб вмонтувати HTML, надає спосіб структурування компонентів з використанням синтаксису знайомого всім хто працює в сфері розробки веб технологій. Компоненти React зазвичай пишуться з використанням JSX, хоча це і не обов'язково (компоненти також можуть бути написані на чистому JavaScript).

Приклад коду JSX:

```
клас App розширює React.Component {
  render() { return (
     <div><p>Header</p><p>Content</p>
     <p>Footer</p></div>);}}
```

React дає змогу зручно та ефективно вести розробку. Незважаючи на те, що компоненти React мають концепцію state, він більше підходить для короткочасного зберігання.

React — це фреймворк, що надає змогу розробляти веб компоненти, а потім рендерити їх в Html файл.

React DOM — це асоційований фреймворк, який забезпечує підключення та редагування DOM.

Redux — це доповнення до React бібліотека, яка забезпечує зберігання даних. Призначений для управління і централізації додатків. Використовується з такими фреймворками, як React чи Angular для створення веб додатків чи веб сайтів.

Redux сам по собі невеликий та має просте простим API для використання, тому в ньому легко розібратись, легко використати, чим і набув популярності, використовується для збереження стану об’єктів і компонентів в одному місці. Надає швидкий доступ до стану, і не потрібно дублювати стан, що дає змогу позбутися мутації об’єктів чи банального загублення стану. За своєю структурою дає зручне підключення, так як всі дані в одному місці, і можна підключитись один раз і використовувати багато раз. Не потрібно дублювати код і на кожній сторінці підключати по багато раз. Працює на чистому React і використовує ті сам компоненти і функції для виклику подій. На події не потрібно кожен раз підключати стан, він підключається один раз і назавжди, до поки не буде прийняте рішення зміни або перепідключення.

Redux є універсальним засобом розробки та може бути використаний у зв'язці з різними бібліотеками та фреймворками. У цій же статті буде розглядатися використання Redux у програмах React.

Використання Redux, це запорука до загального стану сайту представленого об'єктом JavaScript.

Незмінний стейт доступний лише для читання інформації, не дає змогу редагувати, або мутувати дані в стейті.

Події (action) — це об'єкт в мові програмування JavaScript, який лаконічно описує суть зміни:

```
{  type: 'CLICKED_SIDEBAR'}
{  type: 'SELECTED_USER',userId: 232}
```

Єдина вимога до об'єкта дії — це наявність type.

У простому додатку тип дії задається рядком. У міру розростання функціональності додатка краще переходити на константи:

```
const ADD_ITEM = 'ADD_ITEM'
const action = { type: ADD_ITEM, title: 'Third item' }
```

і виносити дії в окремі файли. А потім імпортувати:

```
import { ADD_ITEM, REMOVE_ITEM } from './actions'
```

Генератори дій (actions creators) — це функції, що створюють дії.

```
function addItem(t) {
  return {
    type: ADD_ITEM,
    title: true}}
```

Зазвичай ініціюються разом з функцією надсилання дії:

```
dispatch(addItem('Milk'))
```

Або при визначенні цієї функції:

```
const dispatchAddItem = i => dispatch(addItem(i))
dispatchAddItem('Milk')
```

При запуску дії обов'язково щось відбувається і стан додатки змінюється. Це робота редукторів.

Редуктор (reducer) — це чиста функція, яка обчислює наступне стан дерева на підставі його попереднього стану і використаного дії.

```
(currentState, action) => newState
```

Чиста функція працює незалежно від стану програми і видає вихідне значення, беручи вхідна і не змінюючи нічого в ньому і в іншій програмі. Виходить, що редуктор повертає абсолютно новий об'єкт дерева станів, яким замінюється попередній.

Симулятор редуктора, або спрощено базову структуру Redux можна представити так:

```
const listManager = (state = {}, action) => {
  return {title: title(state.title, action),
   list: list(state.list, action)}
```

Сховище (store) — це об'єкт, який:

— містить стан додатка;

— відображає стан через getState();

— може оновлювати стан через dispatch();

— дозволяє реєструватися (або видаляти) в якості слухача зміни стану через subscribe().

Сховище в додатку завжди унікальне. Так створюється сховище для програми listManager:

```
import { createStore } from 'redux'
import listManager from './reducers'
let store = createStore(listManager)
```

Сховище можна ініціювати через серверні дані:

```
let store = createStore(listManager, preexistingState)
```

Функції сховища:

— Отримання стану store.getState()

— Оновлення стану store.dispatch(addItem('Something'))

Потік даних у Redux завжди однонаправленний.

1.5 Аналіз мов розмітки HTML і CSS.

Кожна сторінка в Інтернеті, яку ми переглядаємо, будується на виконанні окремих вказівок крок за кроком. Наш браузер відображає код, а код без браузера це просто текстовий файл. Коли ми відкриваємо веб–сторінку, браузер відображає HTML та інші мови програмування у максимально зрозумілому для нас форматі.

HTML і CSS насправді лише структура сторінки та інформація про стиль. Оскільки вони знаходяться на передній частині кожної веб сторінки та програми.

1.5.1 Аналіз мови HTML

HTML Hypertext Markup Language — мова розмітки гіпертексту, мова розмітки документів в браузерному середовищі.

HTML це також текстовий документ має текстовий формат представлення веб–документів.

Використовується з 1994 року по теперішній час розробка проводиться під егідою наддержавної організації World Wide Web Consortium. Інформацію "з

перших рук" про стандарти, рекомендації і перспективи розвитку не тільки мови HTML, але і цілого ряду інших web технологій, можна знайти в Інтернеті за адресою http://www.w3.org. Вся документація на сайті W3C представлена на англійській мові, проте є і посилання на переклади (зокрема, на українську мову).

Кожен стандарт, що знов приймається, надає в розпорядження веб майстра нові можливості, що дозволяють зробити HTML-документ ефективним і зовні привабливим.

HTML разом з CSS — являє собою основні технології побудови сайтів.

Охарактеризуємо основні елементи мови HTML. Для позначення меж HTML документу використовується подвійний тег < HTML >. Початковий тег < HTML >, у якого відсутні атрибути, розміщується на початку HTML файлу, а кінцевий тег < /HTML > є останнім тегом коду і позначає закінчення всього документу. Таким чином, елемент HTML є найповнішнім і не входить до складу інших елементів, тобто з елементом HTML не пов'язаний жоден з об'єктів, які відображаються у вікні браузера.

До складу контейнеру HTML входять два структурні елементи: HEAD (елемент заголовку) та BODY (основна частина або тіло документу). Таким чином, документ HTML має вигляд:

< HTML > ⋯

<! -- Тут розміщуються елементи заголовку і основної частини документу -- >

⋯ < /HTML >

Заголовок документу(елемент HEAD)

Заголовок HTML документу визначається елементом HEAD. Тег <HEAD> не має атрибутів. Елемент HEAD розміщується одразу після тегу < HTML >, перед основною частиною веб сторінки:

< HTML >

<HEAD>

<! -- Тут розміщується заголовок документу -- >

</HEAD> ⋯

< /HTML >

Елемент HEAD (як і елемент HTML) не відображається при перегляді веб сторінки, він надає браузеру загальну інформацію про HTML файл.

До заголовку документу входить обов'язковий елемент, який представлений контейнером <TITLE>. Все, що знаходиться між парою тегів <TITLE> і </TITLE>, інтерпретується браузером як назва веб сторінки.

Якщо в документі є гіперпосилання, то назва документу, на який вказано посилання, буде з'являтися у вигляді плаваючої підказки при на-веденні на посилання вказівника миші.

Елемент TITLE по відношенню до елемента HEAD є дочірнім,т обто вкладеним у контейнер <HEAD>:

< HTML >

<HEAD>

<TITLE >Назва Web сторінки</TITLE>…

</HEAD>

< /HTML >

Елемент BODY є наступним компонентом Web сторінки. Парні теги <BODY> і </BODY> вказують на початок і кінець тіла документа. Весь зміст документа міститься в елементі BODY.

Початкові і кінцеві теги елементу BODY є необов'язковими в структурі HTML документа. Проте контейнер BODY необхідний для того, щоб задати властивості всієї сторінки. Наявність в HTML документі елементу BODY є формальною ознакою того, що даний документ має звичайну структуру.

Тег <BODY> розміщується безпосередньо після елементу HEAD, а кінцевий тег </BODY> є передостаннім тегом документу:

< HTML >

<HEAD>

<TITLE >Назва Web-сторінки</TITLE>

</HEAD>

<BODY>

<!- - Зміст документу -->⋯

</BODY>

< /HTML >

Початковий тег <BODY> може доповнюватися декількома атрибутами, які визначають зовнішній вигляд документу в цілому.

HTML, дозволяє організовувати текст у логічні і зрозумілі форму, так як:

— стилі заголовків;

— фізичні стилі;

— логічні етилі;

— списки;

— спеціальні комбінації.

Тег управління абзацом

Фізичні стилі — це реальні стилі шрифту, призначенні, для стилізування тексту .

Тег <I>...</I>

<BIG >... </BIG>

<SMALL >...</ SMALL >

<SUB >...</ SUB >

<SUP >...</ SUP >

Значення:

— курсив (ITALIC);

— жирний шрифт (BOLD).

Телетайп Підкреслений

ПРИКЛАД файл HTML:

<HTML>

<HEAD>

<TITLE> Сторінка титульна <TITLE>

</HEAD>

<BODY>

<P ALIGN=CENTER>

Текст 1

Текст 2 </P>

</BODY>

</HTML>

Заголовки мають такі ознаки:

— Зміст, форма, склад головного заголовка;

— Місце виводу заголовків.

Мова HTML використовує ось такі стилі заголовків для виділення тексту.

Приклад форматі HTML:

<HTML> <HEAD>

<TITLE>Сторінка титульна<TITLE>

</HEAD>

<BODY>

<H1 ALIGN=CENTER>Текст 1</H1>

<H2 ALIGN=CENTER>Текст 2 </H2>

<H3 ALlGN=RIGHT>Текст 3</H3>

<H4> Текст 4</H4>

<H5 ALIGN=RIGHT>Текст 5</H5>

<H6>Текст 6</H6></BODY></HTML>

1.5.2 Каскадні таблиці стилів CSS

Каскадні таблиці стилів (CSS) — це мова таблиць стилів, яка використовується для опису подання документа, написаного в. CSS описує, як елементи мають відображатися на екрані, на папері, у мовленні або на інших носіях.

CSS є однією з основних мов відкритої мережі і стандартизований у всіх веб-браузерах відповідно до специфікацій W3C. Раніше розробка різних частин

специфікації CSS відбувалася синхронно, що дозволяло редагувати версії останніх рекомендацій.

Обсяг специфікації значно збільшився, і прогрес у різних модулях CSS почав настільки відрізнятися, що стало більш ефективним розробляти та випускати рекомендації окремо для кожного модуля. Замість версій специфікації CSS, W3C тепер періодично робить знімки останнього стабільного стану специфікації CSS.

Тег посилання — це спосіб включення файлу CSS. Це найкращий спосіб використання CSS за призначенням: один файл CSS включається на всі сторінки вашого сайту, а зміна одного рядка в цьому файлі впливає на представлення всіх сторінок сайту.

Щоб використовувати цей метод, додаємо тег посилання з атрибутом href, що вказує на файл CSS, який ви хочете включити.

<link rel="stylesheet" type="text/css" href="myfile.css">

Атрибути rel і type також необхідні, оскільки вони повідомляють браузеру, на який тип файлу ми посилаємось.

Замість того, щоб використовувати тег посилання для вказівки на окрему таблицю стилів, що містить наш CSS, ми можемо додати CSS безпосередньо в тег стилю. Ось синтаксис:

Використовуючи цей метод, ми можемо уникнути створення окремого файлу CSS.

Вбудовані стилі — це третій спосіб додати CSS на сторінку. Ми можемо додати атрибут style до будь-якого тегу HTML і додати до нього CSS.

...

...

Селектор дозволяє нам пов’язувати одне або кілька декларацій з одним або кількома елементами на сторінці.

Якщо в нас є елемент p на сторінці, і ми хочемо відобразити в ньому слова за допомогою жовтого кольору.

Ми можемо цей елемент, який націлений на весь елемент за допомогою тегу p на сторінці. Просте правило CSS для досягнення бажаного:

p {колір: жовтий;}

Елементи HTML мають 2 атрибути, які дуже часто використовуються в CSS для зв’язування стилю з певним елементом на сторінці: class та id.

Між цими двома є одна велика різниця: всередині HTML-документа ви можете повторювати одне й те саме значення класу для кількох елементів, але ідентифікатор можна використовувати лише один раз. Як наслідок, за допомогою класів ви можете вибрати елемент з 2 або більше конкретними іменами класів, що неможливо за допомогою ідентифікаторів.

Комбінування селекторів, як націлюватися на елемент, клас чи ідентифікатор. Давайте представимо більш потужні селектори.

Можна націлити на певний елемент, до якого приєднано клас або ідентифікатор.

Можна використовувати та комбінувати селектори, щоб застосувати одні й ті ж оголошення до кількох селекторів. Для цього потрібно відокремити їх комою.

Мою собаку звуть Роджеp,

Ви можете створити більш конкретний селектор, об’єднавши кілька елементів, щоб відповідати структурі дерева документа. Наприклад, якщо у вас є тег span, вкладений у тег p, ви можете націлити його, не застосовуючи стиль до тегу span, який не входить до тегу p.

Щоб зробити залежність суворою на першому рівні, ви можете використовувати символ > між двома токенами:

p > span { колір: жовтий}

У цьому випадку, якщо span не є першим дочірнім елементом p, до нього не буде застосовано новий колір.

Розрізняють декілька типів стилів, які можуть спільно застосовуватися до одного документа. Це стиль браузера, стиль автора і стиль користувача. Переваги стилів: - розмежування коду і оформлення. В ідеалі, веб-сторінка повинна містити

тільки теги логічного форматування, а вид елементів задається через стилі. При подібному розподілі робота над дизайном і верстkою сайту може вестися паралельно, різне оформлення для різних пристроїв. За допомогою стилів можна визначити вид веб сторінки для різних пристроїв виведення: монітора, принтера, смартфона, КПК і ін. Наприклад, на екрані монітора відображати сторінку в одному оформленні, а при її друці — в іншому. Ця можливість також дозволяє приховувати або показувати деякі елементи документа при відображенні на різних пристроях, розширені в порівнянні з HTML способи оформлення елементів. На відміну від HTML стилі мають набагато більше можливостей по оформленню елементів веб сторінок. Простими засобами можна змінити колір фону елемента, додати рамку, встановити шрифт, визначити розміри, положення та ін. Прискорення завантаження сайту. При зберіганні стилів в окремому файлі, він кешується і при повторному зверненні до нього витягується з кеша браузера. За рахунок кешування і того, що стилі зберігаються в окремому файлі, зменшується код веб сторінок і знижується час завантаження документів, єдине стильове оформлення великої кількості документів.

Підключення таблиці стилів браузер зчитує таблицю стилів, і форматує документ згідно цієї таблиці.

Існує три способи підключення таблиці стилів.

Перший спосіб підключення, це підключення внутрішньої таблиці стилів (глобальні стилі) застосовується тоді, коли один документ має унікальний стиль. Для визначення внутрішніх стилів використовують тег.

Другий спосіб підключення, це підключення зовнішньої таблиці стилів (зв’язані стилі) застосовується в ситуаціях, коли один стиль визначається для багатьох сторінок. Для підключення зовнішньої таблиці стилів призначений тег , який розміщається в розділі заголовка: Зовнішню таблицю стилів можна створити в довільному редакторі. Файл зовнішньої таблиці стилів не може містити ніяких тегів HTML. Наприклад: H1 {color: #000080; font-size: 200%; font-family: Arial;

text-align: center;} P {padding-left: 20px;} Розширення файла таблиці стилів — .css. Значення параметрів rel і type залишаються незмінними незалежно від коду.

Третій спосіб підключення, це за допомогою вбудованих стилів (внутрішні стилі) використовуються в тому випадку, коли необхідно застосувати стиль для одного елемента. Недолік: він змішує вміст документа з його представленням і втрачає багато переваг таблиць стилів.

Всі описані способи використання CSS можуть застосовуватися як окремо, так і сумісно один з одним. Але при цьому першим завжди застосовується внутрішній стиль, потім глобальний, а в останню чергу – зв’язаний.

2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ РОБОТИ ВЕБ-ДОДАТКУ ТЕСТУВАННЯ СТУДЕНТІВ З МОЖЛИВІСЮ САМОТЕСТУВАННЯ

2.1 Розробка структури додатку

Практично любий веб додаток є сукупністю сторінок та посилань, які відображають певний матеріал: текст, клавіші, зображення та інші об’єкти. Для комфортного використання користувачем додатку та для полегшеного пошуку потрібної інформації, додаток має мати чітку і продуману структуру.

Признаком хорошої навігація додатку є те, що користувачі точно знають, де знаходяться, де розташовані елементи сайту і як ними користуватися. Структура сайту розроблена таким чином, щоб користувач міг без проблем здійснювати перехід на вкладені сторінки, щоб легко було орієнтуватися на сайті та приємно перебувати на ньому.

Проаналізувавши всі види структур веб додатків, було прийнято рішення обрати комбіновану структуру для створення сайту циклової комісії інформатики та комп’ютерних технологій.

Комбінована структура — це поєднання структур додатку, що дає можливість можливість передбачення ієрархії додатку та в певному місці може містити виконання покрокових дій , тобто наслідувати лінійну структуру.

Вибір саме такої структури є найоптимальнішим рішенням, так як дана структура дозволяє створити зручне та інтуїтивне меню навігації, но при тому лінійне і зрозуміле тестування, виконуючи дію за дією, даючи користувачу вибір так чи ні. Дана структура є досить зручною для відвідувачів, надає легкий доступ до всіх сторінок, матеріалів тестування та необхідної інформації в додатку.

Веб додаток для тестування студентів з можливістю самотестування складається з таких компонентів:

— сторінка «авторизації» для авторизації та реєстрації з можливістю автоматичного входу в додатку при запуску;

— сторінка «створення тестів» на якій користувач створює тести;

— сторінка «список тестів» на якій виводиться список тестів з можливістю проходити тести;

— сторінка «тестування»;

— меню веб додатку;

— сервіс обробник даних Firebase.

Зручний веб додаток передбачає здійснення навігації, тобто має бути меню, доступ до списку тестування, можливості створення та проходження тестування, зручний та інтуїтивний інтерфейс, адаптивний дизайн, та блоки інформації — список гіперпосилань на його розділи

Також при розробці структури варто приділити особливу увагу дизайну, так як це головна складова додатку.

Дизайн — тобто зовнішній вигляд додатку, це дуже важливий знак якості розробки, адже іншими словами дизайн — це обличчя додатку. Від дизайну багато в чому залежить, чи розбереться користувач в діях чи покине його. Адже кожен і з нас перш за все бажає користуватись чимось красивим та зручним, а якщо це не так ми попросту покидаємо сайт, або бистренько робимо справи і більше не бажаємо повернутися. «Легкість» або «важкість» дизайну в деякій мірі впливає на час завантаження сторінок додатку.

Зручність користування — в деяких випадках більш важлива складова сайту навіть у порівнянні з контентом.

Зручність користування, це та міра комфорту, яку відчуває користувач, переміщаючись по сторінках додатку. Якщо сайт містить цінну інформацію, але до неї важко дістатися — не можна вважати такий додаток якісним. Якщо сайт надає корисну інформацію та матеріали, але потрібно прикласти багато зусиль, щоб нею скористатися — такий сайт теж навряд чи буде мати багато відвідувачів. Тому чітка і логічна структура розміщення інформації на сайті — передумова успішного перебування на ньому багатьох відвідувачів.

2.2 Розробка методів створення сторінки авторизації

В додатку розроблено сторінку «авторизації» яка дозволяє керувати рівнями та засобами доступу до додатку, в залежності від введеної пошти і пароля користувача або надання певних повноважень на виконання деяких дій у системі обробки даних.

Для реалізації автизизації підібрано зручний дизайн і реалізовано логіку роботи, щоб користувач при переході на цю сторінку інтуїтивно розумів, що йому потрібно натискати, як це працює і куди потрібно переходити, щоб успішно завершити авторизацію.

Використовуючи можливість фреймворку Реакт створено функціональний компонент Input і button який імпортовано на сторінку Auth. Компонет Input є полем вводу тексту, а button — кнопка. Вони потрібні для створення зручної форми для авторизації.

Форма авторизації складається з двох полів вводу пошти та пароля. Також на сторінці присутні дві кнопки «ввійти» та «авторизуватися».

Сценарій користувача:

— Якщо користувач на сайті вперше і він не зареєстрований, тоді йому потрібно ввести в поля вводу власну пошту та пароль і натиснути кнопку «зареєструватись», після чого дані відправляються на сервер і користувача перенаправить на сторінку «Список тестів» уже зареєстрованим і авторизованим;

— Якщо користувач уже має акаунт, йому потрібно ввести свої дані в поля вводу, та натиснути кнопку «Увійти», після чого його дані відправляються на сервер і користувач перенаправиться на сторінку «Список тестів» будучи авторизованим.

Сесія в додатку авторизованого користувача триває одну годину, після чого користувача повертає на головну сторінку, та не дає змоги робити дії адміна, він стає обичним користувачем.

На сторінці реалізовано перевірку на коректне введеня дананих. Користувачу не будуть доступні клавіші «Ввійти» та «Зареєструватись» поки він не введе коректну пошту та пароль.

Також реалізовано перевірку на мінімальну довжину пароля, який має складатись не менше чим шість символів. В випадку якщо дані введено вірно поля вводу підкреслюються зеленим кольором, не вірно – червоним.

Коли всі дані введено вірно, користувачам стають доступні клавіші «Ввійти» та «Зареєструватись» і в залежності від того, що бажає користувач зробити він тисне на клавіш.

Якщо користувач зареєстрований, но з певних причин натисне клавішу «Зареєструватись» — це не є помилкою, його впросто авторизує, так як коли він натиснувби клавішу «Ввійти». В випадку якщо киристувач введе свої дані, та бажатиме натиснути клавішу «Ввійти» але при цьому не буде авторизованим, додаток попередить його, що здійснити таку дію заборонено, та запропонує користувачу спочатку зареєструватись.

Загальну робота авторизації вказано блок-схемою на рисунку 2.1.

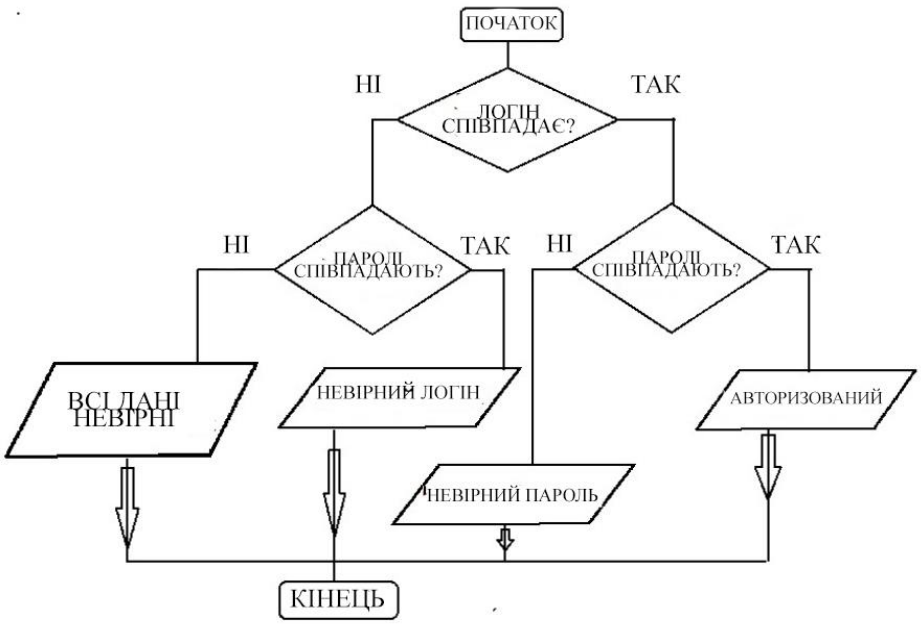

Рисунок 2.1 — Алгоритм роботи сторінки авторизації

2.3 Розробка сторінки створення тестів та сторінки тестування

На етапі створення майже найважливішої сторінки під назвою «Створення тесту» реалізовано класовий компонент форми, що складається з: назви сторінки, форми «Введіть запитання», варіанту тесту та поля вибору вірної відповіді, та двох кнопок «створити тест» та «додати питання».

Класовий компонент — це функція яка є реакт компонентом, в той же час є об'єктом з даним. Такі компоненти називаються "функціональними компонентами", оскільки вони буквально є JavaScript функціями.

Форма «Введіть запитання» та форма «варіант» це компоненти класу «Input» і являють собою поле для вводу інформації, в даному випадку тексту.

Кнопка «створити тест» — відповідає за створення всього тесту, а кнопка «додати питання» — за додавання конкретного питання.

Сценарій користувача:

— перейти на сторінку «створити тест»;

— натиснути та ввести запитання в поле вводу «Введіть запитання»;

— натиснути та ввести варіанти відповіді в поле вводу «варіант;

— повторити попередню дію 4 рази;

— натиснути на поле вибору правильної відповіді позначивни під яким номером відображається вірна відповідь;

— натиснути на кнопку «додати питання»;

— повторити попередню дію необхідну кількість раз;

— натиснути кнопку «створити тест»;

Після чого тест будестворено, та відправлено на зберігання в базу даних, звідки в подальшому будуть братись ці дані для використання та побудування інших сторінок.

На сторінці «Створення тесту» в полях вводу реалізовано потрібну перевірку на мінімальну кількість символів, яка становить одне слово. Все що більше одного слова, вважається коректними даними. Якщо данні в поле вводу

введено не вірно текст підкреслюється червоним та воводиться сповіщення «Поле введено некоректні дані».

Також перевіряється заповнено всі поля чи ні. Якщо поле пусте — текст підкреслюється червоним та воводиться сповіщення «Питання не може бути порожнім» та клавіші «створити тест» та «додати питання» не будуть доступні.

В додатку реалізовано сторінку «список тестів». На цій сторінці виводяться всі створені тести в вигляді посилань. При натисканні на посилання переходимо на сторінку «тестування» яка складається з тексту «Запитання» та багатофункціональних кнопок «варіант відповіді». При натисканні на «варіант відповіді» користувача перенаправляється на наступний тест.

На сторінці «тестування» реалізовано перевірку на вибрану відповідь. Вірна відповідь при натисканні підкреслюється зеленим кольором, не вірна – червоним. Користувач не може вибрати іншу відповідь в момент збереження даних, якщо він натисне на іншу відповідь програма проігнорує його, та перейде через три секунди до наступного тесту.

Сценарій користувача:

— перейти на сторінку «список тестів»;

— зі списку вибрати вибрати тест;

— на сторінці «тестування» вибрати правильну відповідь і зачекати іншого запитання;

— повторити дію стільки раз скільки доступно тестів;

— отримати результат тестування;

— натиснути клавішу «Завершити тестування» або «Повторити».

Сторінка «результат тестування» складається з таких елементів: кількість правильних відповідей у відношенні до кількості питань, а також з двох кнопок «Завершити тестування» та «Повторити». При кліку на які користувач завершує цикл «тестування» та переходить на таступні сторінки, де розпочинається новий цикл дій користувача.

2.4 Розробка методів обробки та зберігання даних

Для того, щоб веб додаток працював, йому як і всім додаткам потрібний сервер на якому він буде зберігати дані.

Сервером, можна назвати процес який запущений на комп'ютері. В наш час так уже ніхто не говорить, і тому сервером називають любий пристрі на якому запущено серверний додаток.

Сервер в нашому додатку виконує функцію зберігача даних, та їх обробки. Також він є хостингом веб додатку.

Для підключення до серверу використовується в ролі клієнта браузер, який через доменне ім'я підключається до VDS і отримує від нього інформацію. Доменне ім'я, моментально транслюється в стандартну IP-адресу. І адреса може залишатися незмінною, навіть якщо домен був у якийсь момент змінено.

Для реалізації цієї задачі використано платформу від Google під назвою Farebase.

Firebase — це флагманська платформа, розроблена Google для створення мобільних і веб додатків. На ній можна розміщати додатки, а також використовувати їхню базу даних для збереження даних додатку.

Для реалізації збереження, даних використовується база даних в реальному часі.

База даних реального часу – це Firebase, що підтримує дані JSON, і всі підключені до неї користувачі отримують оперативні оновлення після кожної зміни.

Вона зберігає дані користувача, та дані створених тестів. Дані користувача порівнюються, в випадку співпадіння користувачу надається дозвіл на вхід, тобто проводиться аутентифікація — ми можемо використовувати анонімну чи парольну автентифікацію. Для неавторизованих користувачі дані будуть додаватись в базу та зберігатись там до подальшої спроби авторизуватися.

Як виглядає меню зберігання даних користувача наведено на рисунку 2.2.

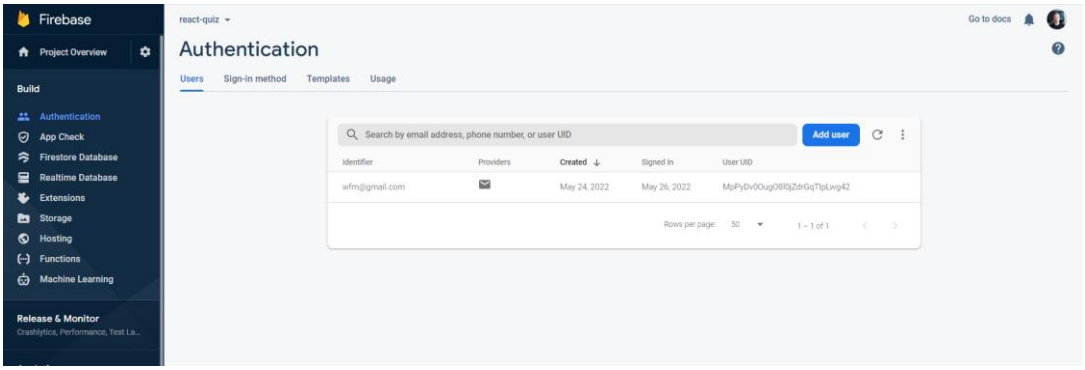

Рисунок 2.2 — Сторінка збереження даних користувача в базі даних firebase

Для збереження даних створених тестів використовується «realtime database» яка дозволяє зберігати дані у вигляді json файлів, що дає змогу надсилати запити на мові javaScript і отримувати відповіді у реальному часі в вигляді даних тесту.

Як виглядає меню зберігання даних в базі даних firebase в наведено на рисунку 2.3.

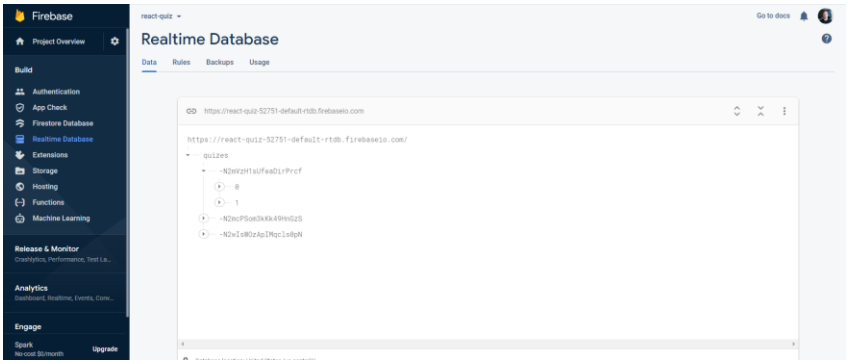

Рисунок 2.3 — Сторінка збереження даних в базі даних firebase

Дії для підключення Firebase:

— створено обліковий запис Firebase;

— створено програму Firebase;

— підключено програму до бази даних в реальному часі;

— реалізація звязків з сервером;

— реалізація адресного простору;

— безпосередня передача даних.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПЛАТФОРМИ

3.1 Варіантний аналіз і обгрунтування вибору програмних засобів

Перед початком розробки програмних засобів необхідно провести огляд доступних програмних засобів та відібрати такі, що найкраще підходять для розробки даного веб додатку.

Для розробки буде проведено огляд мови програмування JavaScript та її фреймворків React + Redux з використанням мов розмітки CSS HTML.

CSS в перекладі — Каскадні таблиці стилів — це мова розмітки стилів, яка використовується для стилізування html документа.

Каскадні таблиці стилів призначені для того, щоб можна було використовувати окремо стилізацію для макетів, шрифтів, вказувати власні кольори. Цей спосіб дає змогу використовувати декільком сторінкам одні і ті самі стилі, так дає ту гнучкість і контроль в характеристиках. Каскадні таблиці стилів підключаються в окремому файлі, що забезпечує легкий доступ та контроль структури вмісту. Файл .css може кешуватись, що забезпечує швидке завантаження сайту, який використовує однакові файли, і дає швидкий доступ користувачу до контенту.

Каскадність стилів слідує від заданих пріоритетів, зверху і в низ, тому для визначення головного стилю просто потрібно помістити його зверху списку стилів. Вони є незамінимими для придання сторінці візуальних стилів і не мають конкуренції.

Для написання структури веб додатку використовується мова розмітки сторінки HTML.

HTML – являє собою той стандарт, який використовують для структурної розмітки сторінок які відтворюються в браузерному середовищі.

Браузерам приходить документ з сервера, або з локальної пам’яті, далі вони беруть цей документ і перетворюють текст в візуальну сторінку, зберігаючи оригінальний зовнішній вигляд прописаний в документі.

Елементи HTML це просто блоки сторінок за допомогою яких будуються HTML файл. Конструкції HTML їх зображення та об’єкти, вже можуть бути вбудованими в сторінку сайту. Елементи HTML відрізняються тегами, записаними з використанням кутових дужок. Теги <img /> чи <input /> вносять певний вміст на сторінку, в даному випадку це фото, чи певний текст. Такі теги як <p>, обертають текст та виводять його на сторінку, а також можуть включати інші теги та елементи. Браузери не відображають теги HTML, вони інтерпретують їх і виводять їх зміст.

Також для реалізації веб додатку використано мову JavaScript — це мова програмування, яка є однією з основних технологій для розробки веб сайтів чи веб додатків. Майже всі сайти використовують JavaScript на стороні клієнта для відтворення поведінки сторінок, також часто використовують в зв’язці з іншими бібліотеками та фреймворками. Усі браузери мають методологію для відтворення JavaScript та для запуску коду на пристроях користувачів.

JavaScript мова високого рівня, часто скомпільована «точно вчасно», яка відповідає стандарту ECMAScript. Він має динамічну оброку тексту, орієнтацію на об’єкт на основі прототипу та першокласні функції. Використовує керування подіями, функціональне програмування. Також має в собі зачатки прикладного програмування, (API) для передачі даних, регулярними виразами, стандартними структурами даних і об’єктною моделлю документа DOM.

Стандарт ECMAScript не містить жодних засобів введення/виводу (I/O), таких як мережа, зберігання засоби для графіків. Браузер, або інша система дяя виконання введення та виводу надає API JavaScript.

Щоб структурувати код додатку та розділити його по різним сторінкам, для зручності перевикористання шаблонів, використано фреймворк React. Також

даний фреймворк є бібліотекою з великою кількістю компонентів, які було використано в розробці додатку.

React (також відомий як React.js) — це фреймворк для JavaScript із відкритим для всіх бажаючих кодом для створення веб-сайтів, та веб додатків на основі компонентів та класового підходу. Його підтримують всі популярні компанії з розробки веб продуктів, компанія Meta засновники фейсбук надають фінансову підтримку для розвитку фреймворку. React використовують як базу для розробки сторінок сайту та мобільних додатків, з використанням фреймворка Next.js. Однак React займається лише керуванням станом і відтворенням цього стану в DOM, при цьому створення веб-додатків на фреймворку React вимагає використання додаткових бібліотек для передачі чи маршрутизації даних, а також забезпечення функці які потрібні клієнтам.

Для зручності збереження локальних даних Reactпрацює в зязці з Redux, що дозволяє зберігати дані в одному місці в локальному стейті.

Redux — це фреймворк для JavaScript із відкритим для всіх бажаючих кодом для керування стейтом додатку. Зазвичай працює в зв'язці з React, для створення веб сторінок більш продуктивно, та для того, щоб пришвидчити завантаження на цих сторінках.

Він був створений Деном Абрамовим та Ендрю Кларком. З середини 2016 року основними супроводжуючими є Марк Еріксон і Тім Дорр.

Redux є маленьким фреймворком з простим та обмеженим API, розробленим для керування станом програми. Він працює подібно до функції скорочення, концепції функціонального програмування.

У результаті проведеного огляду для розробки обрано мову прорамування JavaScript, а саме її фреймворк React, на якому будуть створені основні компоненти програми. Це обгрунтовано потужністю цієї мови програмування, сучасним підходом до створення веб додатків та зручністю роботи з кодом, який має вигляд скрипту. Фреймворк React э лідером на ринку розробки веб технологій тому більшість розробників надають перевагу саме йому.

3.2 Вибір середовища розробки

Для більш зручної розробки програмних засобів використовуються IDE (інтегровані середовища розробки). Інтегроване середовище розробки — це комплекс засобів для розробки, що складається з текстового редактора коду, інструментів для компіляції та відлагодження проектів, а також з доповнень які полегшують рутинні справи при розробці.

Для порівняння середовищ розробки та для подальшого вибору буде розглянуто середовища Microsoft Visual Studio, WebStorm та Visual Studio Code.

Visual Studio Code — текстовий редактор призначений для розробки застосунків та сайтів. Visual Studio Code є безкоштовною програмою і є кросплатформеним, тобто підтримує всі сучасні операційні системи.

VS Code підтримує різноманітні плагіни, які злегістю можливо інтегрувати в нього, але він більше підходить для написання коду інтерпретованими мовами програмування, тому з VS Code буде зручно працювати змовами розмітки та з скриптами javascript.

Microsoft Visual Studio — інтегроване середовище розробки, що розробляється компанією Microsoft. Підтримує такі мови програмування як C, C++, JavaScript .Net та інші. Дане середовище підходить тільки для операційної системи «Windows».

WebStorm — інтегроване середовище розробки на JavaScript, CSS та HTML створено компанією JetBrains. Він забезпечує автодоповнення, аналіз коду на помилки, та має вбудовану систему контролю версій. Розрахований під програмування на мові JavaScript.

Важливою перевагою WebStorm є робота з проектами та рефакторинг коду JavaScript, що знаходиться в різних файлах і папках проекту, а також вкладеного в HTML. Підтримується множинна вкладеність — це коли в документ на HTML вкладено скрипт на Javascript, в який вкладено інший код HTML, всередині якого

вкладений Javascript, тобто в таких конструкціях підтримується коректний рефакторинг.

Для розробки обрано середовище Visual Studio Code через свою доступність, хорошу візуальну частину та підтримку необхідних плагінів для розробки на мові javaScript. Також в цьому середовищі є автозаповнювання, що полегшує розробку в ньому.

Також в виборі Visual Studio Code враховано підтримку зі сторони спільноти Microsoft. Використовуючи VS Code, неможливо потрапити в ситуацію, коли розробник припинить підтримку важливого для розробки розширення ну чи цей інструмент який так потрібне не матиме альтернативи.

Для того, щоб з чимось розібратись просто потрібно знайти в пошукачу відповідь, знайти пояснення та продовжити роботу, так як середовище розробки є дуже популярним і вже хтось стикався з такою проблемою і описав її вирішення. В інтернеті багато навчальних матеріалів та відповідей на всі питання, які можуть виникнути під час розробки. Для інших середовищ зі списку знайти інформацію складніше.

Також дане середовище є дуже зручним в навігації, де в одну робочу область можливо помістити всю структуру будь-якого проекту — всі файли перед очима і не потрібно нічого додатково шукати чи відкривати.

Зручність розробки полегшує плагін Emmet він вже влаштований в середовище розробки VS Code за замовчуванням.

З плагіном Emmet можливо швидше писати код за допомогою додавання коротких команд. Наприклад, якщо я наберу «! + Enter», то відразу отримаю каркас HTML документа —не потрібно запам'ятовувати довгий шаблон, звідки його копіювати або прописувати вручну. Аналогічні команди є на більшість дій, які потрібні для верстки веб додатка.

Тому вибір середовища розробки VS Code є актуальним та доречним враховуючи все вище сказане.

3.3 Розробка компонентів веб додатку

Для ефективної розробки веб додатку даний процесс розділено на декілька етапів. З самого початку розроблено вигляд компонентів додатку. Створено багатофункціональні UI компоненти, які ми можемо перевикористати при створенні додатку в необмеженій кількість раз, а саме:

— Button;
— Input;
— Loader.

Button — являє собою багатофункціональний компонент який належить до класу UI компонентів, які виконують роль зразка з візуальним станом для використання в інших компонентах. По своїй структурі, це звичайна кнопка з кастомним дизайном. Те, що це окремий компонент дозволяє його перевикористати по коду кілька раз, для цього просто потрібно змінити його назву.

Даний компонент має вигляд який вказано на рисунку 3.1:

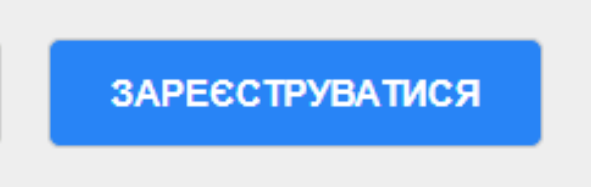

Рисунок 3.1 — Зовнішній вигляд UI компонента Button

Код компоненту Button (Лістинг 3.1).

Лістинг 3.1 – Код компоненту Button

```
const Button = props => {const cls = [classes.Button, classes[props.type]]

 return ( <button onClick={props.onClick} className={cls.join(' ')}

disabled={props.disabled} > {props.children}</button>) } export default Button
```

Input — це багатофункціональний компонент який належить до класу UI компонентів які виконують роль зразка з візуальним станом для використання в інших компонентах. По своїй структурі, це звичайне поле вводу тексту для вводу даних, але він має кастомні класи і візуальну частину. Перевагою даного класу є те, що його можна перевикористати просто надавши іншу умову введення тексту (Лістинг 3.2).

Даний компонент має вигляд який вказано на рисунку 3.2:

Рисунок 3.2 — Зовнішній вигляд UI компонента Input

Лістинг 3.2 – Код компоненту Input

```
function isInvalid({valid, touched, shouldValidate})
{ return !valid && shouldValidate && touched }
const Input = props => {
const inputType = props.type || 'text'
const cls = [classes.Input]
const htmlFor = `${inputType}-${Math.random()}`
if (isInvalid(props)) {cls.push(classes.invalid)}
return (<div className={cls.join(' ')}>
<label htmlFor={htmlFor}>{props.label}</label>
<input type={inputType} id={htmlFor} value={props.value}
onChange={props.onChange} />
```

{isInvalid(props) ? {props.errorMessage || 'Введіть правильне значення'}</span : null}) }

export default Input

Loader — це багатофункціональний компонент який належить до класу UI компонентів які виконують роль зразка з візуальним станом для використання в інших компонентах. Цей компонент зовсім кастомний і індивідуальний. Являє собою обєкт який виводиться під час завантаження сторінок, використаний на кожній сторінці веб–додатку.

Даний компонент має вигляд який вказано на рисунку 3.3:

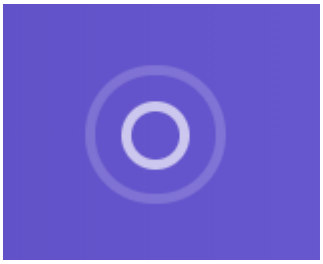

Рисунок 3.3 — Зовнішній вигляд UI компонента Loader

Реалізовано за допомогою стрілкової функції (Лістинг 3.3).

Лістинг 3.3 — Код компоненту Loader

const Loader = props => (

)

export default Loader

Далі в ході розробки реалізовано основні компоненти, які слугують базовими сторінками веб додатку, а саме:

— Auth;

— Quiz;

— quizCreator;

— quizList.

Далі компоненти, що наведені вище, об’єднуються в файлі App.js — цей файл слугує так званою оболонкою, в якій всі компоненти імпортуються і об’єднуються. На основі цього файлу компілюється веб додаток.

Імпорт компонентів (Лістинг 3.1).

Лістинг 3.4 — Імпорт компонентів

```
import Quiz from './containers/Quiz/Quiz'
import QuizList from './containers/QuizList/QuizList'
```

Компонент Auth — це сторінка авторизації, використовуючи цей компонент задається стан авторизації користувача. Для розробки даного компоненту використано UI компонент Input та Button. Дані компоненти перевикористано два рази. Також попрацював над реалізацією мінімальної перевірки введених даних. Для реалізації цього компонента використано класовий підхід, це коли використовується не функція, а викликається клас.

Зовнішній вигляд сторінки авторизації наведено на рисунку 3.4

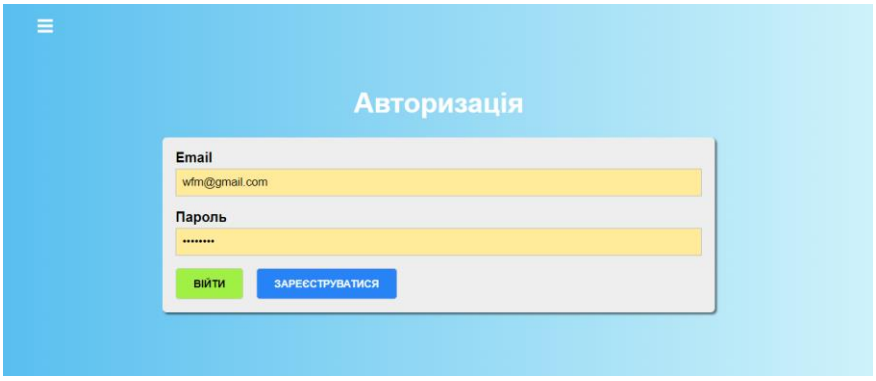

Рисунок 3.4 — Зовнішній вигляд компонента Auth «Авторизація»

На кожен input накладено перевірку. Умовою даної перевірки є те, що в випадку якщо користувач введе замість пошти чи паролю певні невірні дані чи

мінімальна кількість літер буде менше ніж 6 (Лістинг 3.4), клавіші «Ввійти» та «Авторизуватися» будуть неактивні.

Лістинг 3.4 — Код перевірки на введені дані

```
validateControl(value, validation) {
 if (!validation) { return true} let isValid = true
if (validation.required) {isValid = value.trim() !== '' && isValid}
if (validation.email) {isValid = is.email(value) && isValid}
if (validation.minLength) {isValid = value.length >= validation.minLength &&
isValid}
return isValid}
```

Зовнішній вигляд валідації наведено на рисунку 3.5

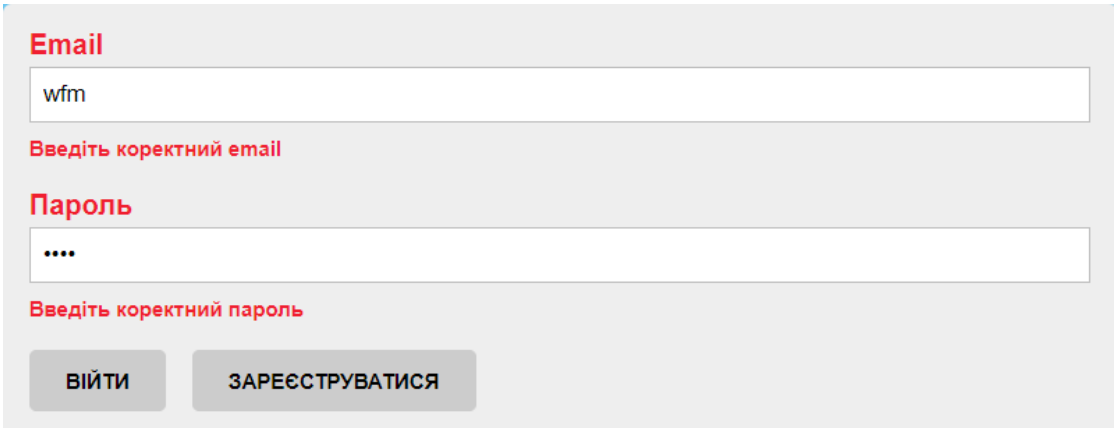

Рисунок 3.5 — Зовнішній перевірки в компоненті Auth

Загальний код рендеру компонента, в якому реалізовано імпортування всіх багатофункціональних компонентів та функцій (Лістинг 3.5).

Лістинг 3.5 – Функція створення та створення Button

```
render() {
return <React.Fragment> {!this.props.isAuthenticated ? (
```

  <h1>Авторизація</h1>
<form onSubmit={this.submitHandler} className={classes.AuthForm}>
{this.renderInputs()}
<Button type="success" onClick={this.loginHandler}
disabled={!this.state.isFormValid} > Ввійти </Button> <Button type="primary"
onClick={this.registerHandler} disabled={!this.state.isFormValid} >
Зареєструватися </Button> </form>
 : <Navigate to='/'/> } </React.Fragment> } }

Компонент quizCreator — це сторінка «Створення тесту», в реалізації даного компоненту використано класово-функціональний підхід — це коли створюється клас і функції, які рендеряться в середині класу і потім експортуються. Зовнішній вигляд сторінки «Створення тесту» наведено на рисунку 3.6

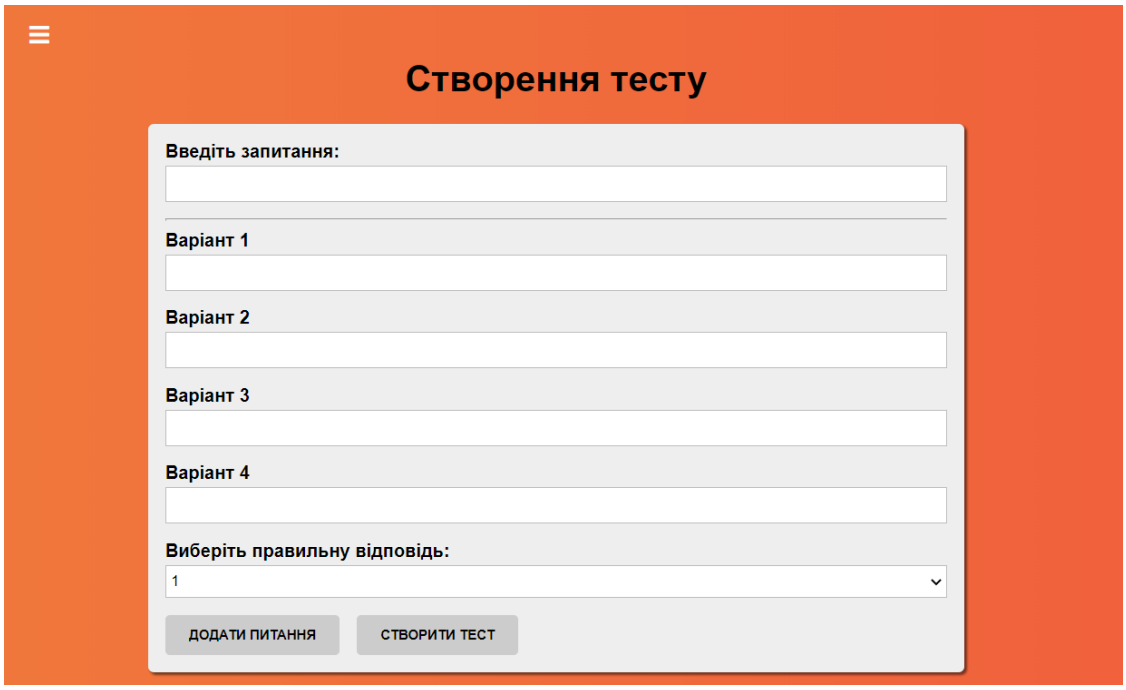

Рисунок 3.6 — Зовнішній вигляд сторінки «Створення тесту»

Даний компонент складається з UI компонентів Input для введення запитання та варіантів відповіді (Лістинг 3.6).

Лістинг 3.6 – Функція створення Інпутів та варіантів

```
function createFormControls() {
{question: createControl({
label: 'Введіть запитання:',
errorMessage: 'Питання не може бути порожнім' }, {required: true}),
  option1: createOptionControl(1), option2: createOptionControl(2),
  option3: createOptionControl(3), option4: createOptionControl(4)}}
  function createOptionControl(number) {return createControl({
  label: `Варіант ${number}`, errorMessage: 'Значення не може бути порожнім',  id: number}, {required: true})}
```

Також реалізовано поле вибору вірної відповіді і два компонента Button з текстом «Додати питання» та «Створити тест». Дані про створений тест, зберігаються в стейт та відправляються на сервер (Лістинг 3.7).

Лістинг 3.7 – Функція збереження даних та створення Button

```
state = {quiz: [], isFormValid: false,  rightAnswerId: 1, formControls: createFormControls() }
await axios.post('/quizes.json', this.state.quiz)
<Button type="primary" onClick={this.addQuestionHandler} disabled={!this.state.isFormValid}>Додати питання </Button>
<Button type="success"  onClick={this.createQuizHandler} disabled={this.state.quiz.length === 0}>Створити тест</Button>
```

Далі ми рендеримо компонент за допомогою функції render() для того, щоб його злегкістю можна було потім імпортувати і використати в нашій гілці програми (Лістинг 3.8).

Лістинг 3.8 – Рендер компонента «Створення тесту»

```
render() { const select = <Select  label="Виберіть правильну відповідь:
" value={this.state.rightAnswerId} onChange={this.selectChangeHandler}
options={[{text: 1, value: 1}, {text: 2, value: 2}, {text: 3, value: 3},
{text: 4, value: 4}]}/>
return (<div className={classes.QuizCreator}> <div> <h1>
Створення тесту</h1>
<form onSubmit={this.submitHandler}>{ this.renderControls() }
{ select }
<Button  type="primary" onClick={this.addQuestionHandler}
 disabled={!this.state.isFormValid} >Додати питання</Button>
<Button type="success" onClick={this.createQuizHandler}
disabled={this.state.quiz.length === 0}>Створити тест</Button>
</form></div></div>)}
```

Компонент Quiz — це сторінка тестування, на яку ми потрампляємо з сторінки «список тестів», на даній сторінці рендериться тест за допомогою функції render() (Лістинг 3.8).

Лістинг 3.8 – Рендер сторінки тестування

```
render() {
return (<div className={classes.Quiz}>
<div className={classes.QuizWrapper}>
```

```
<h1>Дайте відповідь на питання</h1>
{this.state.loading ? <Loader />: this.state.isFinished? <FinishedQuiz
results={this.state.results} quiz={this.state.quiz}
onRetry={this.retryHandler}/>: <ActiveQuiz
answers={this.state.quiz[this.state.activeQuestion].answers}
question={this.state.quiz[this.state.activeQuestion].question}
onAnswerClick={this.onAnswerClickHandler}
quizLength={this.state.quiz.length}
answerNumber={this.state.activeQuestion + 1}
state={this.state.answerState}
results={this.state.results}/>}</div></div>)}
```

Вивід даних реалізовано методом виводу тесту який береться з сервера, зберігається в стейті компонента (Лістинг 3.9).

Лістинг 3.8 – Стан сторінки тестування

```
state = { results: {}, // {[id]: success error} isFinished: false,
activeQuestion: 0, answerState: null, // { [id]: 'success' 'error' }
quiz: [],loading: true}
```

Після того, як виводяться дані тесту, ми оброблюємо натискання по варіанту відповіді, і порівнюємо з правильною відповіддю. За це відповідає функція onAnswerClickHandler (Лістинг 3.9).

Лістинг 3.9 – Функція обробник правильної відповіді.

```
onAnswerClickHandler = answerId => {this.state.answerState})
```

```
if (this.state.answerState) {const key = Object.keys(this.state.answerState)[0]
if (this.state.answerState[key] === 'success') {return }}
const question = this.state.quiz[this.state.activeQuestion]
const results = this.state.results
if (question.rightAnswerId === answerId) {if (!results[question.id]) {
results[question.id] = 'success'}
this.setState({answerState: {[answerId]: 'success'},results})
 const timeout = window.setTimeout(() => {if (this.isQuizFinished()) {
 this.setState({isFinished: true})} else {this.setState({
 activeQuestion: this.state.activeQuestion + 1,answerState: null})}
 window.clearTimeout(timeout)}, 1000)} else {
results[question.id] = 'error'
this.setState((prev) =>
({...prev,answerState:
{[answerId]: 'error'},results}))
const timeout = window.setTimeout(() => {
if (this.isQuizFinished()) {this.setState({isFinished: true} else {
this.setState({activeQuestion: this.state.activeQuestion + 1,
answerState: null})}
window.clearTimeout(timeout)},
```

1000)}

Зовнішній вигляд сторінки «тестування» наведено на рисунку 3.8

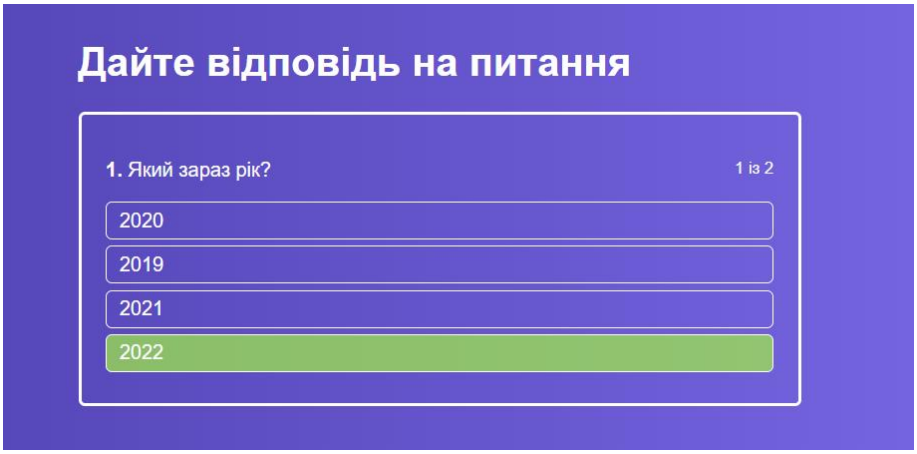

Рисунок 3.8 — Зовнішній вигляд сторінки «тестування»

Рендерінг всього додатку відбувався в компоненті <App>, куди імпортовано всі компоненти та підключено всі необхідні роути за допомогою <Routes>

3.4 Тестування компонентів та модулів веб додатку

Тестування — це процес аналізу та дослідження, який дозволяє виявити інформацію про якість продукту, пов’язану з умовами використання продукту.

Методи тестування також включають:

— процес пошуку дефектів, помилок і збоїв;

— перевірку програмних компонентів для оцінки готовності програмного продукту до використання.

Результати тестування оцінюються за такими критеріями:

— виконувати вимоги, надані розробниками та дизайнерами;

— відповідність вихідних даних;

— допустимий час виконання функцій;

— практичність;

— відповідати вимогам замовника.

Кількість тестів, навіть для простих програмних компонентів, майже необмежена, тому стратегія тестування повинна полягати в тому, щоб робити тільки те, що необхідно, враховуючи доступний час і ресурси. Тому програмне забезпечення перевіряється виконанням стандартної процедури для виявлення помилок, помилок чи інших дефектів.

Існує багато типів тестування: одні зазвичай виконують самі розробники, інші — професійні команди. У нашому випадку будуть використані системні тести.

Тестування системи — це виконання програмного забезпечення в його остаточній конфігурації, інтегрованого з іншими програмними та апаратними системами.

Існує три основних методи тестування:

— тестування білого ящика;

— тестування чорного ящика.

— smoke тести

Техніка "білого ящика" — це не зовнішня поведінка програми, а внутрішня поведінка програми. Потрібно перевірити, чи всі елементи програми сконструйовані правильно і чи вони взаємодіють один з одним.

Зазвичай аналізуються контрольні зв'язки елементів, рідше аналіз — інформаційні зв'язки.

Тестування білого ящика характеризується ступенем виконання або охопленням логіки програми (вихідного тексту).

А одним із методів дослідження цієї проблеми є дослідження методу «чорного ящика».

Тестування «чорного ящика». показують, що:

— Як виконувати функцію програми;

— Як отримати вихідні дані;

— Як були отримані результати;

— Як зберегти цілісність зовнішньої інформації.

Розроблені тест кейси описано в таблиці 3.1.

Таблиця 3.1 Тест кейси для тестування веб додатку

Код тест-кейса	Опис тест-кейса		
	Хід тестування		Очікуваний результат
	Дата тестування	Результат	Примітка
001	Перевірка сторінки авторизації		
	1. Перйти на сторінку веб–додатку. 2. Ввести дані користувача в форму. 3. Незереєстрований користувач: натиснути кнопку зареєструватись 4. Авторизованому користувачу: натиснути кнопку ввійти		Коректне виведення даних Перейде на сторінку «список тестів» Перейде на сторінку «список тестів»
	19.04.2021	Пройдено	-
002	Сторінка «список тестів»		
	1. - Перейти на сторінку «список тестів» та вибрати тест. 2. - На сторінці «тестування» вибрати правильну відповідь і зачекати іншого запитання. 3. - отримати результат тестування.		Результатом буде сторінка з результатом тестування
	19.04.2021	Пройдено	-
003	Перевірка сторінки «створення тесту»		
	1. Перейти на сторінку «створити тест» 2. Клікнути на інпут «Запитання» – ввести запитання. 3. Клікнути на інпут «варіант» - ввести варіанти відповіді – повторити 4 рази. 4. Клікнути на чекбокс, і вибрати правильну відповідь. 5. Натиснути кнопку «додати питання» - при необхідності повторити необхідну кількість раз. 6. Натиснути кнопку «створити тест» 7. Натиснути кнопку додати тест		На сторінці «список тестів» буде відображатись створений тест
	20.04.2021	Пройдено	-
004	Перевірка можливості редагування та видалення тесту.		
	1. Перейти в файрбейс 2. Вибрати потрібний тест 3. Натиснути видалити його		На сторінці «список тестів» відсутній видалений тест
	20.04.2021	Пройдено	-

При тестуванні «чорного ящика» враховуються системні характеристики програми та ігнорується внутрішня логічна структура програми. Комплексне тестування зазвичай неможливе.

Наприклад, якщо програма мала 10 вхідних значень, кожне з яких приймає 10 значень, кількість варіантів тесту буде 1010.

Багато функцій тестового «чорного ящика» не реагують на помилки програмного забезпечення.

Тестування програмних засобів буде доречно проводити використовуючи методику «чорного ящика».

Вона базується на використанні шаблонів тестування, бо ж тест кейсів. Це означає що буде створено декілька ситуацій у яких перевіряється працездатність додатку, коректності основних функцій, буде доречно проводити використовуючи.

3.5 Інструкція користувача

Для початку роботи потрібно запустити веб додаток. Для цього просто потрібно перейти на сторінку веб додатку. Є два вида користувачів — незареєстровані — це всі користувачі і зареєстровані користувачі — це користувачі які мають доступ до сторінки створення тесту.

Інструкції по користуванню модулями веб додатку для користувачів.

Інструкції по користуванню модулем «авторизація»:

— новий користувач який ще не зареєстрований, для користування додатком повинен ввести пошту та пароль і натиснути на кнопку під назвою «зареєструватись», після чого дані відправляються на сервер, і користувач переходить на сторінку «Список тестів» уже будучи зареєстрованим і авторизованим;

— у випадку, якщо користувач уже має акаунт і зареєстрований, він просто вводить свої дані і тисне клавішу під назвою «ввійти», після чого дані відправляються на сервер, і користувач переходить на сторінку «Список тестів» будучи авторизованим.

Інструкції по користуванню модулем «створення тесту»:

— перейти на сторінку «створити тест»;

— натиснути на поле вводу «Запитання» — ввести запитання;

— натиснути на поле вводу «варіант» — ввести варіанти відповіді — повторити 4 рази;

— натиснути на поле вибору правильної відповіді;

— натиснути кнопку «додати питання» — при потребі повторити необхідну кількість раз;

— натиснути кнопку «створити тест».

Інструкції по користуванню модулем «список тестів»:

— перейти на сторінку «список тестів» та вибрати тест;

— на сторінці «тестування» вибрати правильну відповідь і зачекати іншого запитання;

— повторити попередній пункт, поки не закінчиться тестування;

— отримати результат тестування.

Сторінка «результат тестування» складається з оцінки та кількості правильних чи не правильних відповідей. І матиме 2 кнопки «завершити тестування» — після кліку на неї потраплятиме на сторінку «список тестів» і кнопки «повторити» — після кліку на яку буде потрапляти на повторне проходження тесту.

Кожна сесія триває близько одної години, якщо користувач перебуває в цей час на сайті, сесія буде продовжувати до того моменту поки він не покине вкладку додатку. Після того як він це зробить, його буде розлогінено з додатку, та при наступній спробі зайти, додаток попросить його авторизуватись знову.

Для зручності авторизації, в додатку присутня функція автоматичного вводу пароля, для цього користувачу потрібно зберегти свій пароль в браузері, і при наступному вході, йому запропонують ввійти за допомогою збереженого пароля.

ВИСНОВКИ

У ході виконання бакалаврської дипломної роботи проведено розгляд веб додатків для тестування студентів з можливістю самотестування. Вибір фреймворка зумовлено тим що react являє собою найкращий інструмент на ринку для реалізації веб додатків, а його класовий та компонентний підхід, полегшує розробку на мові програмування javaScript. Вибір самої мови програмування завершено на мові програмування javaScript, супутніх мовах розмітки Html та Css. Використання власної системи для тестування студентів, дає можливість подальшого вдосконалення веб додатку та його оптимізації під поставлені цілі.

Розроблено структуру програми, складено алгоритми роботи компонентів, створення на проходження тестування. Продумані користувацькі сценарії та проведене тестування. Виконано варіативний аналіз та аргументування вибору програмних засобів при розробці платформи, а саме: мова програмування javaScript, фреймворк react, мови розмітки html та css, сервер firebase та середовища розробки Visual Studio Code. Описано програмну реалізацію веб додатку. Створено веб додаток, для тестування студентів з можливістю самотестування, створення тестів, проходження їх та з можливістю реєстрації. Розглянуто основні методики тестування, виконане тестування методикою «чорної скриньки», складено перелік тест кейсів для тестування, а також розроблено інструкцію користувача.

Таким чином всі поставлені задачі вирішено і мета дослідження досягнута.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Матросов А. HTML 4.0 в подлиннике. – СПб: Bhv, 2016. – 672с.

2. «Сучасний підручник JavaScript» [Електронний ресурс] Режим доступу: https://uk.javascript.info/.

3. О Програмні системи створення веб–сайтів, CMS [Електронний ресурс] / Програмні системи створення веб–сайтів, CMS – Режим доступу: http://www.znannya.org/?view=WebDev.

4. Эрик А. Мейер. CSS. Каскадные таблицы стилей. Подробное руководство. - Издательство: Символ-Плюс 2014 р. Переводчик: Н. Шатохина.

5. JavaScript Підручник. Основи веб–програмування [Електронний ресурс] Режим доступу: https://w3schoolsua.github.io/js/index.html.

6. WEB-дизайн Руководство пользователя. Под ред. Леонтьева. Познавательная книга, 2000.

7. React в дії / Марк Тиленс Томас, 2019.

8. «Підручник: Введення в Реакт» [Електронний ресурс] Режим доступу: https://learn-reactjs.ru/tutorial.

9. Принципи якості веб–сайтів з культури / ред. М.Т.Н. Темпера, А. Темпера. - М., 2006.

10. Документація JavaScript, Typescript [Електронний ресурс]. – 2020. – режим доступу: http://www.w3.org/ .

11. Документація веб стандартів [Електронний ресурс]. – 2020. – режим доступу: http://web-standards.ru/ .

12. Документація по JavaScript [Електронний ресурс]. – 2020. – режим доступу: https://javascript.com/ (дата звернення 04.06.2020).

13. Документація по JavaScript (стандарт ES6) [Електронний ресурс]. – 2020. – режим доступу: https://learn.javascript.com/.

14. Документація по fast-ethernet [Електронний ресурс]. – 2014. – режим доступу: http://www.ixbt.com/comm/tech-fast-ethernet.html.

15. habr.com [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/450282/

16. habr.com [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/137388/

17. hrliga.com [Електронний ресурс]. – Режим доступу: https://hrliga.com/index.php?module=profession&op=view&id=167

18.computerhope.com [Електронний ресурс]. – Режим доступу: https://www.computerhope.com/jargon/c/css.htm

19.tutorialspoint.com [Електронний ресурс]. – Режим доступу: https://www.tutorialspoint.com/css/what_is_css.htm

20.hrliga.com [Електронний ресурс]. – Режим доступу: https://hrliga.com/index.php?module=profession&op=view&id=1676

ДОДАТОК А

Технічне завдання

Міністерство освіти та науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ ВНТУ
д.т.н., проф.
_____ О. Д. Азаров
«24» березня 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської дипломної роботи
«Програмне забезпечення тестування студентів з можливістю самотестування»
08-23.БДР.040.00.000 ПЗ

Науковий керівник к.т.н. доц. каф. ОТ
_____ Черняк О.І.
Студент групи 1КІ–20мс
_____ Хмель С.А.

ВНТУ 2022

1 Підстави для виконання бакалаврської дипломної роботи (БДР) наступні:

— актуальність розробки, полягає у вирішенні проблеми великого попиту на сервіси які забезпечують можливість тестування студентів та забезпечити їх надійним додатком для тестування, без лишніх функцій, який для користування не потребує лишніх дій, а просто дає змогу самостійно протестувати рівень знань;

— наказ про затвердження теми бакалаврської дипломної роботи.

2 Мета і призначення БДР наступні:

— метою БДР є розробка програмного забезпечення тестування студентів з метою самотестування на мові програмування javaScript з використанням фреймворка React;

— призначення розробки полягає у виконанні бакалаврської дипломної роботи із подальшим впровадженням та розвитком.

3 Вихідні дані для виконання БДР наступні:

— технічний опис програмного застосунку;

— мова програмування JavaScript;

— фреймворк React;

— веб додаток;

— Опис етапів тестування;

— середовище розробки Visual Studio Code.

4 Вимога до виконання БДР наступна:

— створення веб додатку;

— створення сторінки авторизації;

— створення сторінки для додавання тестування;

— створення сторінки проходження тестування;

— технічний опис сторінок додатку.

5 Етапи БДР та очікувані результати

Робота виконується за вісім етапів, таблиця А.1.

Таблиця А.1 — Етапи виконання роботи

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Постановка задачі роботи	09.01.22	09.01.22	Вступ
2	Пошук матеріалів по технологіям розробки веб-додатків на мові JavaScript	10.02.22	25.02.22	Розділ 1
3	Структурне проектування додатку тестування студентів з можливістю самотестування.	26.02.22	28.02.22	Розділ 2
5	Обгрунтування та вибір засобів реалізації системи	29.02.22	04.03.22	Розділ 2
5	Розробка головного головної сторінки, сторінки створення тесту, авторизації.	05.03.22	29.04.22	Розділ 3
6	Розробка алгоритму збереження та обробки даних.	05.03.22	28.04.22	Розділ 3
7	Оформлення пояснювальної записки та ілюстративного матеріалу	28.04.22	01.06.22	Пояснювальна записка
8	Перевірка якості виконання бакалаврської роботи та усунення недоліків	07.06.22	07.06.22	Презентація

6 Матеріали, що подаються до захисту БДР, наступні: пояснювальна записка БДР, графічні і ілюстративні матеріали, протокол попереднього захисту БДР на кафедрі, відгук наукового керівника, відгук рецензента, протоколи складання

державних екзаменів, анотації до БДР українською та іноземною мовами, нормоконтроль про відповідність оформлення ДР діючим вимогам.

7 Порядок контролю виконання та захисту БДР

Виконання етапів графічної та розрахункової документації ДР контролюється науковим керівником згідно зі встановленими термінами. Захист ДР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлення БДР

При оформлюванні БДР використовуються:

— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104-2006 «Едина система конструкторської документації. Основні написи»;

— Методичні вказівки. Кафедра обчислювальної техніки 2022;

9 Вимоги щодо технічного захисту інформації в ДР з обмеженим доступом відсутні.

Технічне завдання до виконання отримав _____ Хмель С.А,

ДОДАТОК Б

Код основного документу веб додатку

```
import React, {Component} from 'react'
import Layout from './hoc/Layout/Layout'
import {Route, Routes, Navigate, useLocation} from 'react-router-dom'
import Quiz from './containers/Quiz/Quiz'
import QuizList from './containers/QuizList/QuizList'
import Auth from './containers/Auth/Auth'
import QuizCreator from './containers/QuizCreator/QuizCreator'
import {connect} from 'react-redux'
import Logout from './components/Logout/Logout'
import {autoLogin} from './store/actions/auth';
class App extends Component {
  componentDidMount() {
    this.props.autoLogin()
  }
  render() {
    let routes = (
      <Routes>
        <Route path="/auth" element={<Auth/>} />
        <Route path="/quiz/:id" element={<Quiz />} />
```

```
    <Route path="/" exact    element={<QuizList/>} />
    <Route
    path="/"
    element={<Navigate to="/" replace />}
    />
  </Routes>
)
if (this.props.isAuthenticated) {
  routes = (
    <Routes>
      <Route path="/test-creator" element={<QuizCreator/>} />
      <Route path="/quiz/:id" element={<Quiz />} />
      <Route path="/logout" element={<Logout/>} />
      <Route path="/" exact element={<QuizList/>} />
      <Route path="/auth" element={<Auth/>} />
    </Routes>
  )
}
return (
  <Layout>
```

```
        { routes }
      </Layout>
    )
  }
}
function mapStateToProps(state) {
  return {
    isAuthenticated: !!state.auth.token }
}
function mapDispatchToProps(dispatch) {
  return {
    autoLogin: () => dispatch(autoLogin())}}
function withRouter(Child) {
  return function withRouter(props) {
    const location = useLocation();
    // other relevant props// ...
    return <Child {...props} location={location} />;};}
export default withRouter(connect(mapStateToProps, mapDispatchToProps)(App))
```

ДОДАТОК В

Схема роботи сторінки авторизації

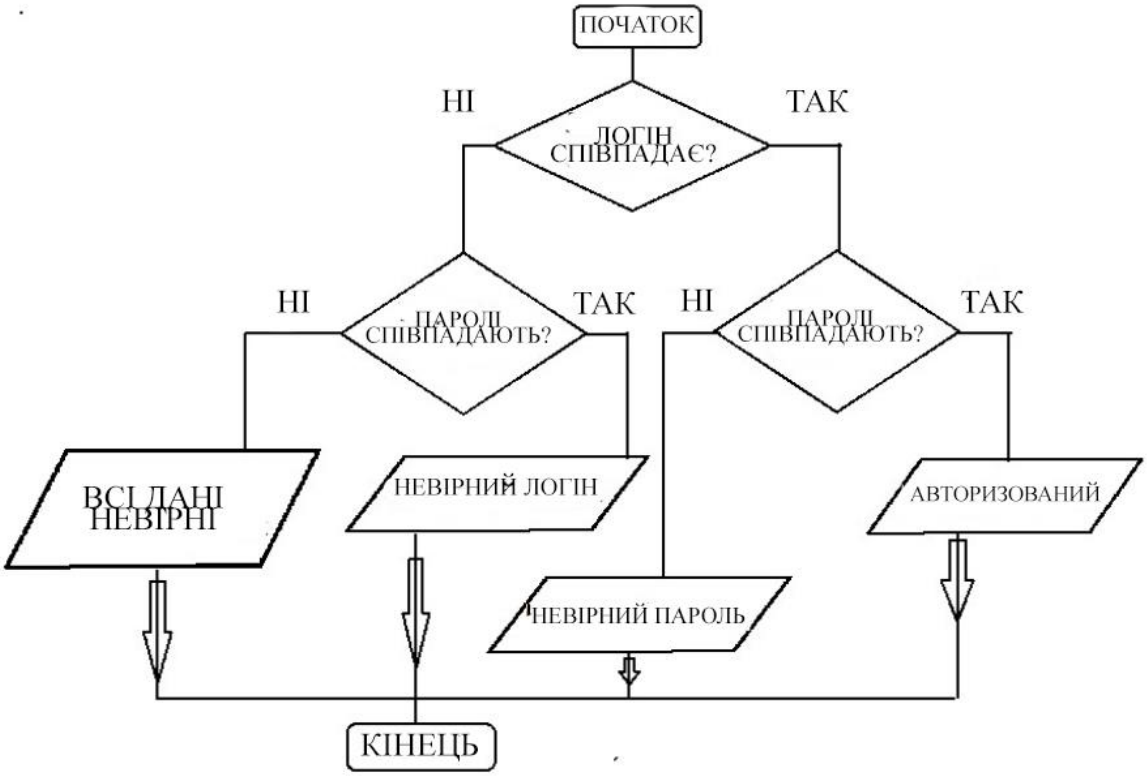

Рисунок Б.1 — Схема роботи сторінки авторизації

ДОДАТОК Г

Структура бази даних додатку

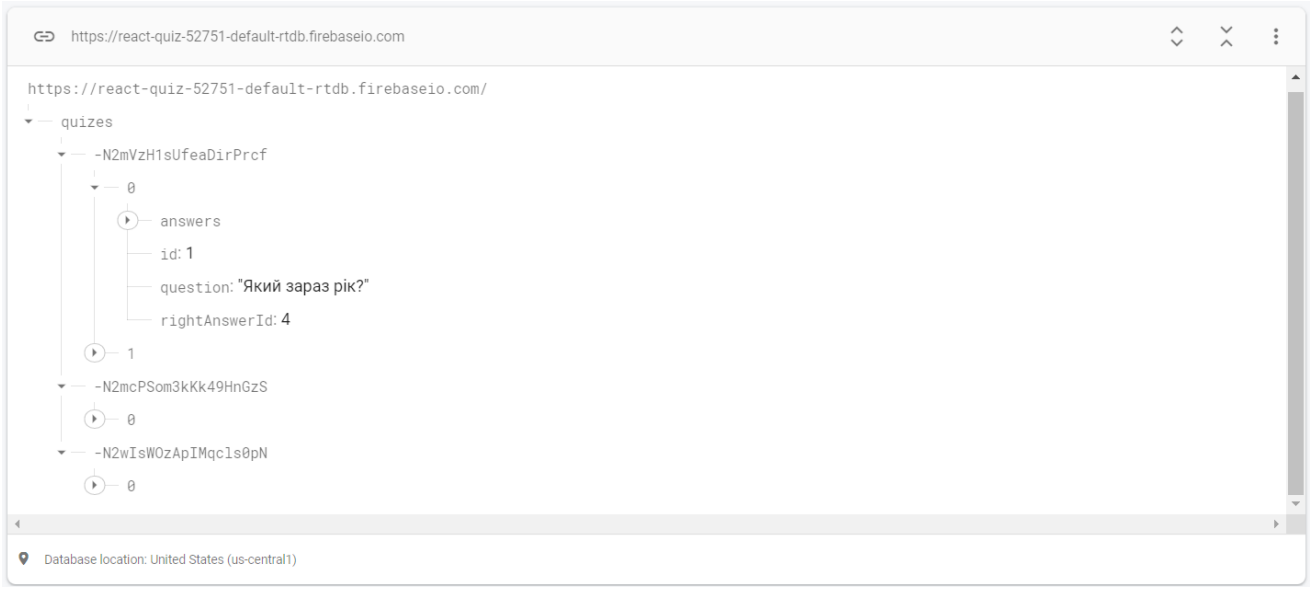

Рисунок Г.1 — Структура бази даних додатку

ДОДАТОК Д

Код сторінки створення тестування

```
import React, {Component} from 'react'
import classes from './QuizCreator.css'
import Button from '../../components/UI/Button/Button'
import Input from '../../components/UI/Input/Input'
import Select from '../../components/UI/Select/Select'
import {createControl, validate, validateForm} from '../../form/formFramework'
import Auxiliary from '../../hoc/Auxiliary/Auxiliary'
import axios from '../../axios/axios-quiz
function createOptionControl(number) {
  return createControl({ label: `Варіант ${number}`,
    errorMessage: 'Значення не може бути порожнім', id: number
  }, {required: true})}
function createFormControls() {return {question: createControl({
label: 'Введіть запитання:', errorMessage: 'Питання не може бути порожнім'
}, {required: true}),
option1: createOptionControl(1), option2: createOptionControl(2),
option3: createOptionControl(3), option4: createOptionControl(4)}}
export default class QuizCreator extends Component {
```

```
  state = {quiz: [],
  isFormValid: false,rightAnswerId: 1,formControls: createFormControls()}
   sibmitHandler = event => {event.preventDefault()} addQuestionHandler = event =>
{event.preventDefault() const quiz = this.state.quiz.concat()   const index = quiz.length
    const {question, option1, option2, option3, option4} = this.state.formControls
    const questionItem = {question: question.value, id: index,
    rightAnswerId: this.state.rightAnswerId, answers: [
    {text: option1.value, id: option1.id}, {text: option2.value, id: option2.id},
    {text: option3.value, id: option3.id},text: option4.value, id: option4.id}]}
    quiz.push(questionItem) this.setState({
    quiz, isFormValid: false, rightAnswerId: 1, formControls: createFormControls()})}
createQuizHandler = async event => {    event.preventDefault()
try { await axios.post('/quizes.json', this.state.quiz)
this.setState({quiz: [],isFormValid: false, rightAnswerId: 1,formControls:
createFormControls()})
} catch (e) {console.log(e)}}changeHandler = (value, controlName) => {
const formControls = { ...this.state.formControls }
const control = { ...formControls[controlName] }
control.touched = true  control.value = value
control.valid = validate(control.value, control.validation)
formControls[controlName] = control
```

```
this.setState({formControls, isFormValid: validateForm(formControls)})}
  renderControls() {
    return Object.keys(this.state.formControls).map((controlName, index) => {
   const control = this.state.formControls[controlName]return (
  <Auxiliary key={controlName + index}><Input label={control.label}
   value={control.value} valid={control.valid}shouldValidate={!!control.validation}
   touched={control.touched} errorMessage={control.errorMessage}
   onChange={event => this.changeHandler(event.target.value, controlName)} />
  { index === 0 ? <hr /> : null }</Auxiliary>)})}selectChangeHandler = event => {
    this.setState({rightAnswerId: +event.target.value})}
  render() {const select = <Select label="Виберіть правильну відповідь:"
   value={this.state.rightAnswerId}onChange={this.selectChangeHandler}
   options={[{text: 1, value: 1},{text: 2, value: 2},{text: 3, value: 3},{text: 4, value:
4}]}/>return (<div className={classes.QuizCreator}><div>
  <h1>Створення тесту</h1><form onSubmit={this.submitHandler}>
{ this.renderControls() }{ select }<Button type="primary"
onClick={this.addQuestionHandler}disabled={!this.state.isFormValid}>
Додати питання</Button>
<Button type="success" onClick={this.createQuizHandler}
disabled={this.state.quiz.length === 0}>Створити тест</Button>
</form></div></div>)}}
```

ДОДАТОК Е

Код сторінки тестування

```
import React, {Component} from "react";

import { useParams } from "react-router-dom";

import classes from './Quiz.css'

import ActiveQuiz from "../../components/ActiveQuiz/ActiveQuiz";

import FinishedQuiz from "../../components/FinishedQuiz/FinishedQuiz";

import axios from "../../axios/axios-quiz";

import Loader from "../../components/UI/Loader/Loader";

class Quiz extends Component {state = {results: {}, isFinished: false,activeQuestion:
0, answerState: null, quiz: [],loading: true}

onAnswerClickHandler = answerId => {if (this.state.answerState) {

const key = Object.keys(this.state.answerState)[0] if (this.state.answerState[key] ===
'success') {return}}const question = this.state.quiz[this.state.activeQuestion]

const results = this.state.results (question.rightAnswerId === answerId) {if
(!results[question.id]) {results[question.id] = 'success'}

this.setState({answerState: {[answerId]: 'success'},results})

const timeout = window.setTimeout(() => {if (this.isQuizFinished())
{this.setState({isFinished: true})} else { this.setState({activeQuestion:
this.state.activeQuestion + 1,answerState: nul})}window.clearTimeout(timeout)},
1000)} else {results[question.id] = 'error'this.setState((prev) => ({...prev,

    answerState: {[answerId]: 'error'},results}))
```

```
const timeout = window.setTimeout(() => {if (this.isQuizFinished()) {
this.setState({isFinished: true})} else {this.setState({
activeQuestion: this.state.activeQuestion + 1, answerState: null})}
window.clearTimeout(timeout)}, 1000)}}isQuizFinished() {
return this.state.activeQuestion + 1 === this.state.quiz.length }
retryHandler = () => {this.setState({activeQuestion: 0,answerState: null,
isFinished: false,results: {}})}async componentDidMount() {
try {const response = await axios.get(`/quizes/${this.props.id}.json`)
const quiz = response.data this.setState({
quiz, loading: false})} catch (e) {}render() {return (
<div className={classes.Quiz}><div className={classes.QuizWrapper}>
<h1>Дайте відповідь на питання</h1>{
this.state.loading? <Loader />: this.state.isFinished? <FinishedQuiz
results={this.state.results}quiz={this.state.quiz}onRetry={this.retryHandler}
/>: <ActiveQuiz answers={this.state.quiz[this.state.activeQuestion].answers}
question={this.state.quiz[this.state.activeQuestion].question}onAnswerClick={this.on
AnswerClickHandler}
quizLength={this.state.quiz.length}answerNumber={this.state.activeQuestion + 1}
state={this.state.answerState} results={this.state.results}/>}
</div></div>)}}const QuizHOC = () => {const params = useParams();return <Quiz
id={params.id}/>}export default QuizHOC;
```

ДОДАТОК Ж

Код сторінки авторизація

```
class Auth extends Component {state = {isFormValid: false,formControls: {
email: {value: '',type: 'email',label: 'Email',errorMessage: 'Введіть коректний email',
valid: false,touched: false,validation: {required: true,email: true}},
password: {value: '',type: 'password',label: 'Пароль',errorMessage: 'Введіть
коректний пароль',valid: false, touched: false,
validation: {required: true,minLength: 6}}},isAuthorizated: false}
loginHandler = () => {this.props.auth(this.state.formControls.email.value,
this.state.formControls.password.value,true}
  registerHandler = () => {
    this.props.auth(
      this.state.formControls.email.value,
      this.state.formControls.password.value,
      false )}
  submitHandler = event => {
    event.preventDefault()}
  validateControl(value, validation) {
    if (!validation) {return true}
    let isValid = true
    if (validation.required) {
      isValid = value.trim() !== '' && isValid}
    if (validation.email) {
      isValid = is.email(value) && isValid}

    if (validation.minLength) {
      isValid = value.length >= validation.minLength && isValid
    }return isValid}
```

```
onChangeHandler = (event, controlName) => {
  const formControls = { ...this.state.formControls }
  const control = { ...formControls[controlName] }
  control.value = event.target.value
  control.touched = true
  control.valid = this.validateControl(control.value, control.validation)
  formControls[controlName] = control
  let isFormValid = true
  Object.keys(formControls).forEach(name => {
    isFormValid = formControls[name].valid && isFormValid})
  this.setState({
    formControls, isFormValid})}
renderInputs() {
  return Object.keys(this.state.formControls).map((controlName, index) => {
    const control = this.state.formControls[controlName]
    return (<Input key={controlName + index}
    type={control.type}value={control.value}
    valid={control.valid}touched={control.touched}
    label={control.label}shouldValidate={!!control.validation}
    errorMessage={control.errorMessage}
    onChange={event => this.onChangeHandler(event, controlName)}/>)})}
render() {return <React.Fragment>
    {!this.props.isAuthenticated ? (
    <div className={classes.Auth}>
      <div><h1>Авторизація</h1>
        <form onSubmit={this.submitHandler} className={classes.AuthForm}>
          {this.renderInputs()}
          <Buttontype="success"
            onClick={this.loginHandler}
```

```
            disabled={!this.state.isFormValid}>
            Ввійти</Button><Button
            type="primary"
            onClick={this.registerHandler}
            disabled={!this.state.isFormValid}>
            Зареєструватися
          </Button></form></div></div>
    ) : <Navigate to='/'/> }
    </React.Fragment>}}
function mapStateToProps(state) {
  return {
    isAuthenticated: !!state.auth.token}
function mapDispatchToProps(dispatch) {
  return {
    auth: (email, password, isLogin) => dispatch(auth(email, password, isLogin))}}
export default connect(mapStateToProps, mapDispatchToProps)(Auth)
```

ДОДАТОК К

Код файлу зборки проекту в html файл

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="theme-color" content="#000000">
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json">
    <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.1.0/css/all.css" integrity="sha384-lKuwvrZot6UHsBSfcMvOkWwlCMgc0TaWr+30HWe3a4ltaBwTZhyTEggF5tJv8tbt" crossorigin="anonymous">
    <title>Tester App</title>
  </head>
  <body>
    <noscript>
      You need to enable JavaScript to run this app.
    </noscript>
    <div id="root"></div>
  </body>
</html>
```

ДОДАТОК Л

ПРОТОКОЛ ПЕРЕВІРКИ
КВАЛІФІКАЦІЙНОЇ РОБОТИ НА
НАЯВНІСТЬ ТЕКСТОВИХ
ЗАПОЗИЧЕНЬ

Назва роботи: Програмне забезпечення тестування студентів з можливістю самотестування

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності
Unicheck

Оригінальність 85,4% Схожість 14,6%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Хмель С.А,
(підпис) (прізвище, ініціали)

Керівник роботи _____ Черняк О.І.
(підпис) (прізвище, ініціали)