**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ**
**Кафедра програмної інженерії**

# КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА СИСТЕМИ ВІДСТЕЖЕННЯ ПОМИЛОК З ВИКОРИСТАННЯМ REACT»**

Виконав: студент 4 курсу, групи 6.1219-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
В.В. Кістрінь
(ініціали та прізвище)
Керівник старший викладач кафедри програмної інженерії, PhD, Чопорова О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

**ЗАТВЕРДЖУЮ**
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ 07 ” 02 2023 р.

# З А В Д А Н Н Я

## НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Кістріню Владиславу Віталійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка системи відстеження помилок з використанням React

керівник роботи Чопорова Оксана Володимирівна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » січня 2023 року № 102-с

2. Строк подання студентом роботи 07.06.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Проєктування програмного забезпечення.
4. Реалізація та тестування програмного доповнення.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою докладу

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання 07.02.2023 р.

## К А Л Е Н Д А Р Н И Й  П Л А Н

| № | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | Розробка плану роботи. | 08.02.2023 | |
| | | | |
| 2. | Збір вихідних даних. | 24.02.2023 | |
| | | | |
| 3. | Обробка методичних та теоретичних | 10.03.2023 | |
| | джерел. | | |
| | | | |
| 4. | Розробка першого та другого розділу. | 03.04.2023 | |
| | | | |
| 5. | Розробка третього розділу. | 05.05.2023 | |
| | | | |
| 6. | Оформлення та нормоконтроль | 01.06.2023 | |
| | кваліфікаційної роботи бакалавра. | | |
| | | | |
| 7. | Захист кваліфікаційної роботи. | 21.06.2023 | |

Студент _____ В.В. Кістрінь
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.В. Чопорова
(підпис) (ініціали та прізвище)

**Нормоконтроль пройдено**

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

# РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка системи відстеження помилок з використанням React»: 43 с., 21 рис., 25 джерел.

БАГ-ТРЕКІНГОВА СИСТЕМА, CSS, EXPRESS, HTML, JAVA, MONGODB, NODE.JS, REACT.JS, SCRIPT.

Об’єктом дослідження – процес розробки вебдодатків з використанням React

Предмет дослідження: засоби створення вебсайту із використанням технології REACT.

Мета роботи: розробка системи відстеження помилок вебдодатків з використанням React.

Метод дослідження – аналітичний.

Для розробки застосунку використовувались такі технології, як React.js, Node.js, MongoDB, Express.js та середовище розробки VS Code.

Розроблена баг-трекінгова система призначена для відслідковування помилок в проєкті.

# SUMMARY

Bachelor’s qualifying paper «Development of an Bug Tracking System using React»: 43 pages, 21 figures, 25 references.

BUG TRACKING SYSTEM, CSS, EXPRESS, HTML, JAVA, MONGODB, NODE.JS, REACT.JS, SCRIPT.

The object of the study is the process of developing web applications using React.

The aim of the study is a developing a web application error tracking system using React.

The research method is analytical.

The method of research is analytical.

To develop the application, we used technologies such as React.js, Node.js, MongoDB, Express.js, and the VS Code development environment.

The developed bug tracking system is designed to track bugs in the project.

**ЗМІСТ**

# СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

| | |
|---|---|
| CSS | Cascading Style Sheets |
| DOM | Document Object Model |
| HTML | HyperText Markup Language |
| JS | Java Script |
| UML | Unified Modeling Language |
| NPM | Node Package Manager |
| VS Code | Visual Studio Code |

# ВСТУП

У наш час, вебдодатки є невід'ємною частиною сучасного цифрового світу і широко використовуються як засіб комунікації, розваги, електронної комерції та багатьох інших сфер життя. Це ставить перед розробниками вимогу створення якісних, ефективних та зручних інтерфейсів користувача. Під час розробки таких додатків нерідко зустрічаються помилки та проблеми, які потрібно відстежувати та виправляти. Забезпечення ефективного процесу відстеження та усунення помилок стає ключовим фактором для забезпечення якості та стабільності вебдодатків.

Метою цієї роботи є розробка системи відстеження помилок, яка буде забезпечувати збір, відстеження та аналіз помилок, виникаючих під час роботи вебдодатків, реалізованих з використанням React. Застосунок має забезпечувати зручний та ефективний інтерфейс для управління помилками, а також надавати звіти та статистику щодо створених помилок.

Для досягнення поставленої мети в роботі буде проведений аналіз існуючих систем відстеження помилок, використання React для розробки вебдодатків.

Отримані результати та розроблений прототип системи відстеження помилок можуть бути використані розробниками вебдодатків для полегшення процесу виявлення та усунення помилок, що допоможе покращити якість та стабільність їх додатків.

В даній роботі будуть розглянуті основні етапи розробки системи відстеження помилок з використанням React та Node.js, а також детально описані використані технології та методики розробки.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Короткі відомості та актуальність систем відслідковування помилок

Система виявлення та відслідковування помилок – це система, що дозволяє створювати та зберігати звіти виявлених помилок, в тому чи іншому програмному забезпеченні та супроводжувати до його вирішення. Баг-трекінг є важливим компонентом кожного проєкту, так як одна помилка може мати декілька способів відтворення або ж навпаки, викликати декілька багів. Тому в таких випадках і використовується функціонал систем відстеження для встановлення залежності між баг-репортами.

Баг-репорт – це документ, який містить в собі чітку та важливу інформацію про виявлену проблему, яка призвела до некоректної роботи ПО та способи її відтворення.

Особливостями баг-репортів в системі відстеження помилок є можливість керування ними. Такі як:

– переміщувати до нового етапу розробки;
– відкласти через брак інформації по відтворенню;
– блокування через неможливість вирішення;
– переадресація баг-репорту до іншої категорії чи розробника;
– задавати пріоритет.

Пріоритет виконання може залежати від масштабу проблеми, що вона зачіпає або на скільки серйозну перешкоду вона становить для просування проєкту. Баг-репорти можуть створюватись не лише для вирішення помилок, а і покращення ПО, таким завданням зазвичай призначається низький або нульовий пріоритет.

## 1.2 Порівняння сучасних систем відслідковування помилок

В цьому розділі розглянемо порівняльну характеристику популярних баг-трекінгових систем створених для задоволення потреб розробників різних галузь.

Jira – безкоштовний інструмент для команд до 10 користувачів від компанії Atlassian з широким функціоналом управління проєктами та можливостями відстеження проблем, яка дозволяє підключати плагіни для ще більшого спектру. Jira використовує agile-методологію для керування, яка передбачає розбиття проєкту на фази та наголошує на безперервній співпраці та вдосконаленні [1]. Вона була вперше опублікована в 2002 році, що робить її однією з перших систем відстеження проблем у світі [2].

MantisBT – це інструмент із відкритим кодом. Незважаючи на досить простий інтерфейс, має широкий функціонал і пропонує всі необхідні функції для відстеження помилок. Працює з різними базами даних, інтегрується з чатами та інструментами відстеження часу [3].

Bugzilla – це надійна, багатофункціональна і досвідчена система відстеження дефектів. Яка дозволяє командам розробників ефективно відстежувати невирішені помилки, проблеми, питання, вдосконалення та інші запити на зміни у своїх продуктах. Прості можливості відстеження дефектів часто вбудовані в інтегровані середовища управління вихідним кодом, такі як GitHub, або інші веб, або локально встановлені аналоги [4].

Redmine – гнучкий крос-платформовий та крос-базовий (міжбазовий) вебзастосунок для управління проєктами. Має користувацькі поля для завдань, записів часу, проєктів та користувачів, інтеграцію з SCM (SVN, CVS, Git, Mercurial та Bazaar) та можливість створювати завдання за допомогою електронної пошти [5].

## 1.3 Вибір технологій та інструментів розробки

В якості інструменту для ефективної розробки інтерактивних інтерфейсів вебдодатку баг-трекінгової системи використовувалась відкрита JS бібліотека – React. Для розорбки серверної частини та обробки запитів використовувався express, який є фреймворком на базі Node.js. За для зберігання та маніпуляції даними було вирішено використовувати документо-орієновану базу даних MongoDB.

### 1.3.1 Обґрунтування вибору React.js

React.js – JavaScript бібліотека для створення користувацьких інтерфейсів, яка з самого початку була створена за для вирішення проблеми часткового оновлення та таким чином, щоб додавати стільки React скільки потрібно користувачу. React дозволяє розробляти складні інтерфейси завдяки маленьким частинам коду, іменованими – компонентами, які зручно збирати, повторно використовувати та тестувати. Компоненти створювати за допомогою JSX, який дозволяє писати HTML-елементи всередині JavaScript. Нижче можна побачити легкий для порівняння приклад (див. рис. 1.1) взятий з офіційного сайту React [6].

```
class HelloMessage extends React.Component {
  render() {
    return React.createElement(
      "div",
      null,
      "Привіт, ",
      this.props.name
    );
  }
}

root.render(React.createElement(HelloMessage, { name: "Taylor" }));
```

```
class HelloMessage extends React.Component {
  render() {
    return <div>Привіт, {this.props.name}</div>;
  }
}

root.render(<HelloMessage name="Taylor" />);
```

Рисунок 1.1 – Порівняння React-компоненту без та з використання JSX

Логіка компонентів написана на java script, замість звичних шаблонів, тому з легкістю можна передавати структуровані набори даних не звертаючись до DOM

DOM – це незалежний програмний інтерфейс, який дозволяє отримувати доступ до складу мов гіпертекстових розміток та змінювати їх.

React.js використовує Virtual DOM, яка в свою чергу являється абстрацією реального DOM зберігаючи в пам’яті. Так як RDOM являє собою деревоподібну структуру йому потрібен час та ресурси для перерисовки(render) дочірніх елементів. Змінюючи стан самої компоненти, React порівнює зміни зі старим станом та перебудовує VDOM, виконуючи необхідні зміни в реальному DOM, таким чином зберігаючи ресурси [7].

### 1.3.2 Аналіз додаткових бібліотек для розробки

React.js, маючи стан бібліотеки JavaScript, має безліч власних бібліотек та розширених пакетів, які розширюють функціонал її можливостей або ж надають додаткові можливості для розробки. Одною з найпоширеніших у використанні, є – бібліотека Redux.

Redux – це менеджер станів, який найчастіше використовується в поєднанні з React та Angular для зручного керування користувацькими інтерфейсами при розробці [8].

Основними концепціями Redux являються:

– редюсер(reducer);

– дія(action);

– сховище(Store).

Material-ui – це React бібліотека для користувацьких інтерфейсів. Вона надає набір готових компонентів та інструментів, яка реалізує дизайн інтерфейсу від Google, для полегшення створення красивих та адаптивних інтерфейсів [9].

React-Router – зазвичай у вебсайтів, браузер завантажує весь документ з сервера, та при кожному натисканні оновлює та завантажує його заново. React-Router дозволяє оновлювати посилання(URL-адресу) після натискання без повторного завантаження документу з серверу, що значно пришвидшує роботу сайту [10].

Більшість сайтів на останніх етапах потребують відносин з API. Axios – це promise-based HTTP-клієнт, який дає змогу користуватися ассинхронним JavaScript для більш читабельного коду. Даний клієнт обладнаний перехопленням, скасовуванням та вбудованим захистом від фальшивих запитів [11].

### 1.3.3 Вибір середовища для серверної частини

Вибір середовища розробки для серверної частини проєкту є важливим етапом, який може вплинути на продуктивність, швидкість розробки і підтримку вашого додатку.

Node.js та express.js обрані для серверної частини проєкту з таких причин:

Використання Node.js забезпечує високу продуктивність та швидкість обробки запитів завдяки своїй подійній моделі та неблокуючим операціям вводу/виводу. Express.js, як легковагий фреймворк, спрощує розробку серверної логіки.

JavaScript використовується як на фронтенді, так і на бекенді, що дозволяє розробляти обидві частини проєкту з використанням однієї мови програмування, що спрощує обмін кодом та забезпечує єдність в дизайні та логіці програми.

Node.js та express.js мають велику та активну спільноту розробників, що дозволяє використовувати різноманітні розширення та пакети для швидкої розробки та отримання підтримки [12].

Express.js забезпечує простоту в розробці серверної логіки з його

зрозумілим API, а також надає гнучкість для вибору архітектури та підходів розробки.

Враховуючи ці фактори, використання Node.js та express.js є розумним вибором для реалізації серверної частини проєкту, оскільки це дозволяє досягти швидкості, ефективності, простоти в розробці та гнучкості, а також забезпечує єдність мови програмування на фронтенді та бекенді [12].

### 1.3.4 Загальний огляд баз даних

База даних – це структурована система, яка описує, зберігає та взаємодіє між елементами за певною концепцією. База даних має великий спектр використання у розробці та експлуатації ПЗ, найчастіше її використовують для динамічних сайтів з великим обсягом інформації.

Бази даних розділяють на реляційні та нереляційні:

Реляційні – бази, які зберігають дані у виді таблиць з рядків та стовбців, які надають високу надійність та гнучкість для вирішення більшості задач. Запити в реляційних базах даних формують за допомогою структурованої мови SQL**.** Вона дозволяє робити вибірки, проводити агрегації та угруповання, змінювати та видаляти дані, модифікувати структуру БД, керувати доступом користувачів до тих чи інших операцій. До реляційних відносять такі популярні бази даних: MySQL, SQLite3 і PostgreSQL [13].

Нереляційні(NoSQL) – бази даних, які формуються як ключ-значення, та зберігаються в документному вигляді без використання таблиць. Найчастіше до нереляційної бази даних вдаються для організації сучасних додатків, які керують великими неструктурованими обсягами даних. До нереляційних відносять бази даних: Cassendra, Hbase, MongoDB, і Neo4 [14].

### 1.3.5 Вибір бази даних

Для збереження даних була обрана нереляційна база даних MongoDB, так як в проєкті задіяний великий обсяг неструктурованих даних.

MongoDB – нереляційна база даних документів з відкритим вихідним кодом, побудована на архітектурі горизонтального масштабування, яка використовується для створення високодоступних і масштабованих інтернет-додатків. Кожен запис у базі даних MongoDB є документом, описаним у BSON, двійковому представленні даних. Додатки можуть отримувати цю інформацію у форматі JSON [15]. Приклад опису історичної людини на офіціному сайті MongoDB (див. рис. 1.2).

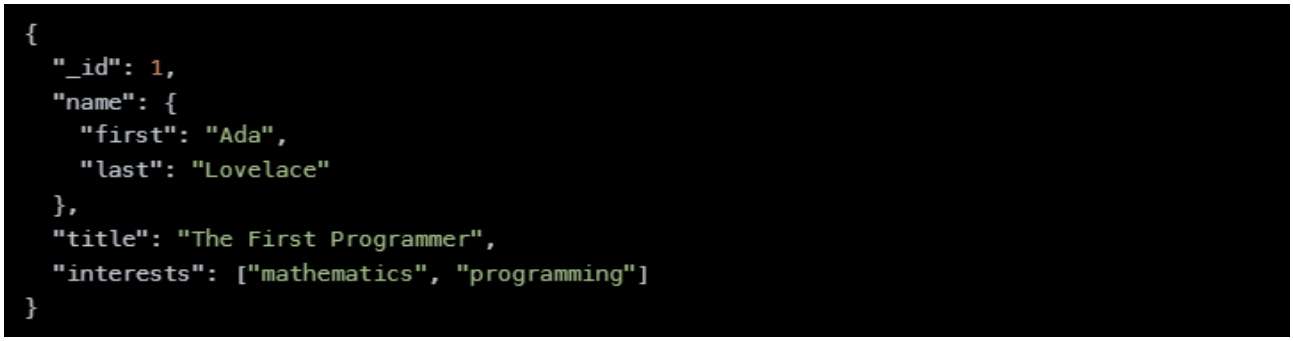


Рисунок 1.2 – Приклад структури опису елементу в базі даних

## 1.4 Опис основних етапів створення вебсайту

Аналіз вимог: полягає в розумінні вимог та потреб користувачів, які функціональні можливості має виконувати вебсайт, які дані потрібно зберігати та для якого кола користувачів розробляється.

Розробка технічного завдання: є невід`ємною частиною розробки кожного проєкту. На цьому етапі будо прописано всі основні деталі: план сайту, кількість сторінок, компонентів, основні деталі дизайну, функціонал та можливості.

Макетування: створюються макети сторінок та окремих компонентів прописаних в технічному завданні.

Верстка та розробка користувацького інтерфейсу: створюється зручний та привабливий користувацький інтерфейс, за вже існуючими макетами.

Проєктування бази даних: проєктування бази даних, яка буде використовуватись для зберігання юзерів, проєктів, багів, задач, коментарів та інших важливих даних про проєкти. Та вибір самої нереляційної бази даних;

Розробка функціональності: на цьому етапі розробляється функціональність самого вебсайту, підключення бібліотек та повний стек розробки.

Тестування: виконується повна перевірка проєкту на наявність помилок та коректність відображення всіх компонентів [16].

## 1.5 Висновок до розділу 1

В першому розділі було розглянено основні переваги тенології React, її бібліотеки. Переглянуто основні поняття баг-трекінгової системи. Порівняли сучасні баг-трекінгової системи та виділили їх переваги. Розібрали що таке бази даних, їх види та обрано найзручнішу для власного проєкту. Також були представлені основні етапи розробки проєкту.

# 2 ПРОЄКТУВАННЯ ВЕБДОДАТКУ

## 2.1 Загальне призначення

Основним призначенням розробки баг-трекінгового вебсайту заключається в розробці зручного та інтуітивно-зрозумілого додатку для створення баг-репортів та відслідковування просування рішень проблем. Основна мета – полегшення процесу виявлення, внесення та відстеження багів у програмному забезпеченні.

## 2.2 Проєктування користувацького інтерфейсу

Проєктування користувацького інтерфейсу відбувалося з метою полегшити користувачам доступ до всіх потрібних функцій застосунку та підняти продуктивність вирішення проблем з помилками.

Додаток має велику кількість функцій:

– створення, видалення та редагування власного проєкту;
– підключення до проєкту за посиланням;
– додавання, видалення та редагування колонок стану завдань;
– створення, редагування та видалення завдань тощо.

Цільовою аудиторією даного застосунку є не лише розробники, менеджери та тестувальники, а і звичайні користувачі можуть користуватися для записів To do листів.

Дизайн інтерфейсу було обрано темно-синього кольору, який надає м’яку та легку для очей атмосферу, що підкреслює професійну основу. Для контрастності було обрано виділення та підкреслення активних елементів жовтим кольором, що символізує другу частину назви застосунку Hive (вулик).

## 2.3 Проєктування вебсайту

Проєктування сайту – це один з перших етапів розробки сайту. На цьому етапі розробляється архітектура майбутнього сайту, з урахуванням специфіки бізнесу та цілей вебпроєкту.

UML – це мова моделювання в об’єктно-орієнтованому програмуванні для моделювання та візуалізації програмних систем. Є невід’ємною частиною процесу розробки програмного забезпечення. UML представляє собою набір діаграм, які допомагають розібрати, аналізували та детальніше зрозуміти структуру додатку [17].

Діаграма прецедентів (Use Case diagram) використовується для моделювання функціональності системи з точки зору її користувачів та прецедентів. Допомагає показати акторів(користувачів) та їх взаємодію з різними функціональними частинами системи.

Діаграма класів (Class diagram) використовується для відображення структури класів системи, їх атрибутів, методів, спадковості та взаємозв’язків між класами[17].

Діаграма компонентів (Component diagram) використовується для відображення поділ системи на компоненти та залежності.

Діаграма розгортання (Deployment diagram) використовується для моделювання вузлів та атрефактів [17].

### 2.3.1 Діаграма прецедентів

Діаграма прецедентів – діаграма, яка відображає взаємодію між акторами(користувачами) та прецедентами, яка описує функціональність та поведінку (див. рис. 2.1) [17].

Користувачі (Actor): актори представляють ролі, які взаємодіють з системою. Актор може бути людиною, іншим програмним засобом.

Прецеденти (Use Cases): прецеденти описують конкретні функціональні можливості системи. Вони представляють конкретні дії або функціональність, яку актор може виконати або використовувати в системі.

Взаємозв’язки (Relationships): взаємозв’язки показують взаємодію між акторами і прецедентами. Найпоширеніші взаємозв’язки включають асоціацію, спадкування, включення та розширення.


Рисунок 2.1 – Діаграма прецедентів баг-трекінгового додатку

Нижче наведено опис прецедентів.

Прецедент «Створення проєкту» – надає можливість “користувачу” створювати та видаляти проєкти. Після створення проєкту “користувачу” присвоюється статус “адміністратора”.

Прецедент «Підключення до проєкту» – надає адміністратору можливість додавати/видаляти користувачів та надавати їм кастомні або ролі за замовчуванням (розробник, тестувальник, менеджер).

Прецедент «Додавати/видаляти колонки» – надає всім користувачам можливість управляти колонами.

Прецедент «Додавати/видаляти завдання» – надає всім користувачам можливість управляти завдання.

Прецедент «Призначення багу» – надає всім користувачам можливість призначати репорт певному користувачу.

Прецедент «Блокувати/відкладати завдання» – надає всім користувачам можливість блокувати та відкладати назначене їм завдання при неможливості вирішення даної проблеми або при низькому пріоритеті може відкластися до наступного спринту, розміщуються в спеціальне відділення архіву.

### 2.3.2 Функціональна діаграма

Функціональними називають діаграми, які створюються на початкових етапах для відображення взаємодії між функціями проєкту та дозволяють розробнику визначити її складові [18].

Функціональна діаграма сайту представлена на рисунку 2.2.

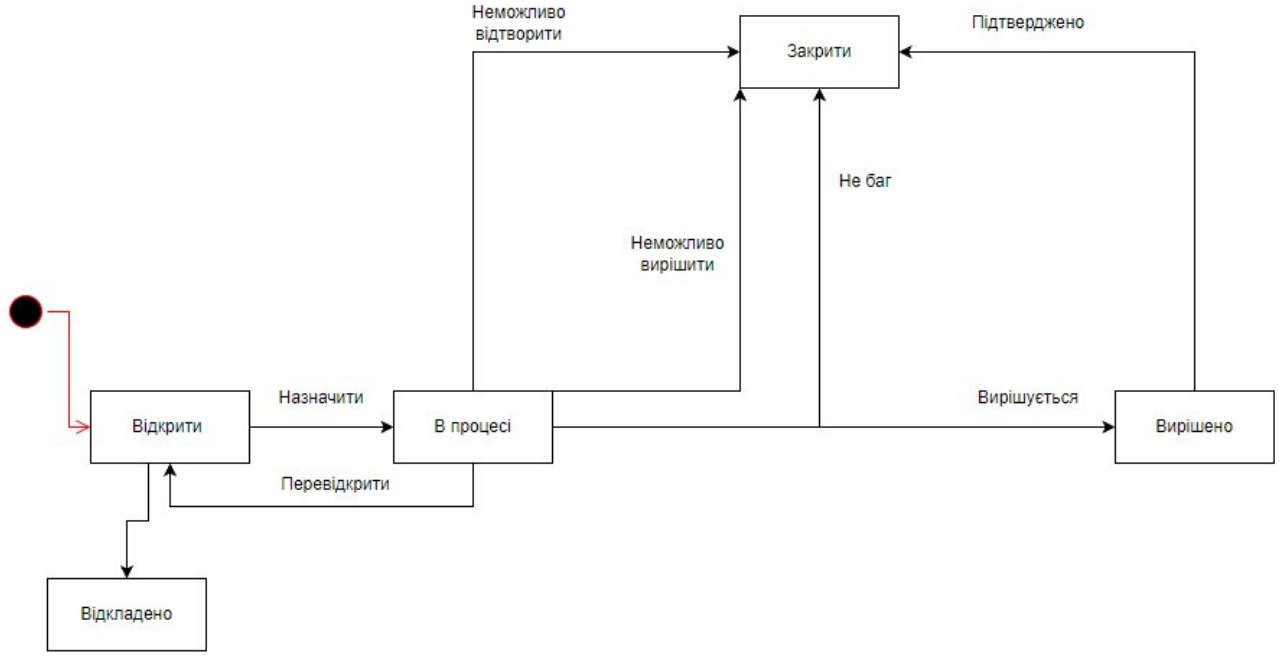


Рисунок 2.2 – Функціональна діаграма живого циклу баг-репорту

## 2.4 Визначення основних компонентів користувацького інтерфейсу

Основні компоненти баг-трекінгового вебсайту писані нижче.

Навігаційна панель: елемент, на якому користувачі бачать логотип та назву застосунку.

Компонент Dashboard: компонент дозволяє переглядати графіки, статистику та інформацію про продуктивність проєкту.

Компонент Backlog: компонент який демонструє зміни за певний проміжок часу, дозволяє переглядати інформацію: що змінено та ким змінено.

Архів: компонент, в якому містяться завершені, заблоковані та видалені баг-репорти певного проєкту з функціями відновлення та видалення з архіву.

Інформація про користувача: компонент який зберігає інорфмацію про користувача, дату реєстрації, створені та під'єднанні проєкти.

Створення нового баг-репорту: компонент, завдяки якому користувач може додавати нові баг-репорти.

Перегляд та редагування репорту: компонент у вигляді popup-вікна, який дозволяє переглядати інформацію про обраний репорт, змінювати стан, переназначати, та при необхідності видаляти та блокувати.

Коментування: компонент, в якому описується додаткова інформація, проводиться дискусія між командою проєкту, прикріплюються скріншоти та відео з проблемою або способом відтворення.

## 2.5 Структурна схема сайту

Структурна схема – це діаграма, яка використовується для візуалізації структури системи або компонентів. Діаграма демонструє взаємозв'язок між різними елементами системи та описує їх ієрархію та взаємодію. Ця схема допомагає зрозуміти організацію та структуру системи, сприяє аналізу та розумінню коду, сприяє комунікації між розробниками та архітектурою

системи [19]. На рисунку 2.3 описана структурна схема додатку BugHive.

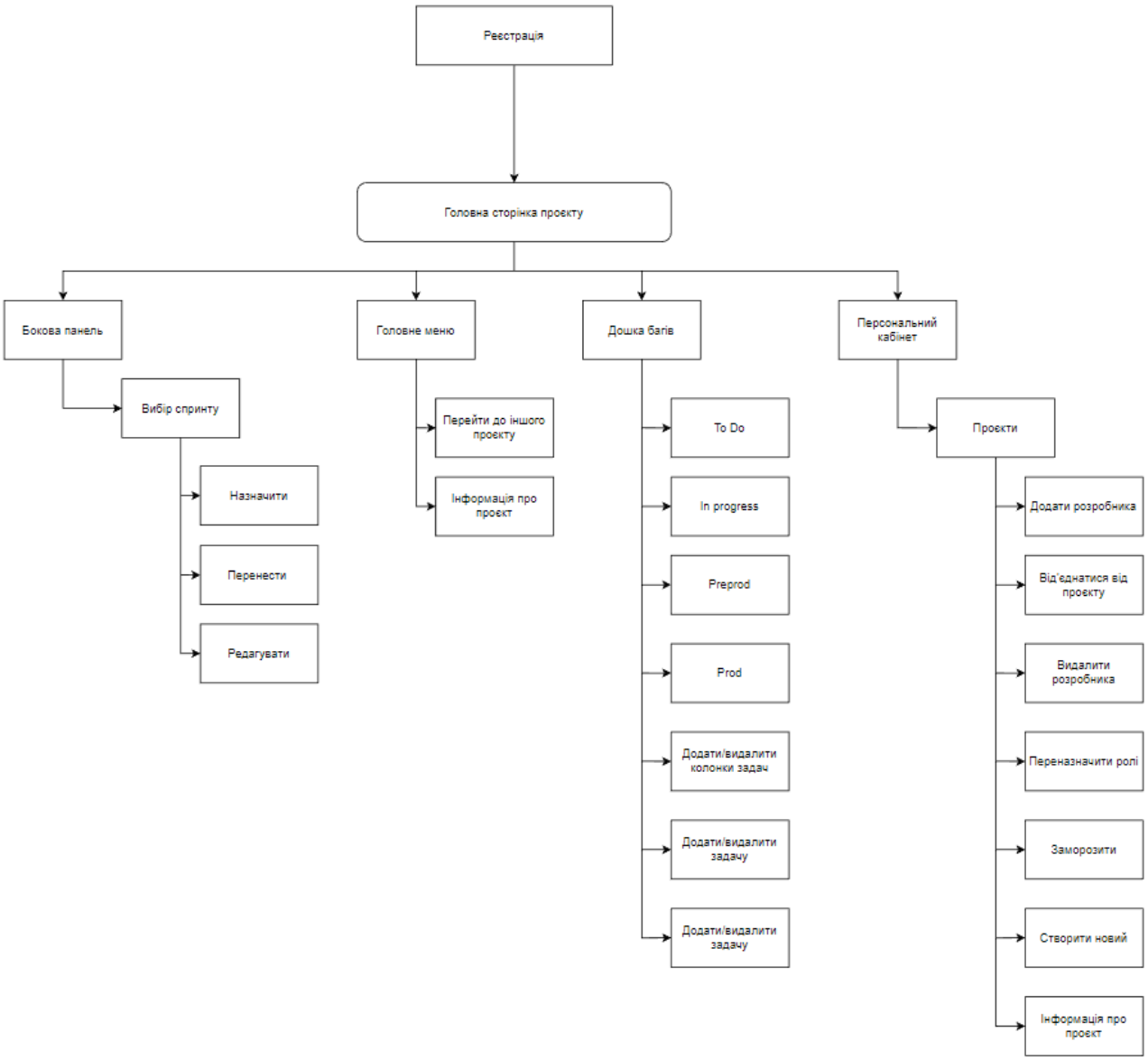


Рисунок 2.3 – Структурна схема додатку BugHive

## 2.6 Висновки до розділу 2

Отже, в другому розділі було розглянуто призначення розробки, мету, цілі та цільову аудиторію яка може використовувати даний застосунок. Розібрали основні поняття UML діаграм та важливість їх розробки на ранніх етапах, наведені приклади використання при розробці сайту. Переглянули короткі

відомості про макетування, етапи які виконувалися та наведено декілька шаблонів власного застосунку.

# 3 СТВОРЕННЯ ВЕБСАЙТУ

## 3.1 Розробка логотипу «BugHive»

Розробка логотипу є важливим етапом процесу створення професійного стилю та ідентичності бренду. Логотип виступає як визначний графічний символ, що ідентифікує застосунок для цільовою аудиторією.

На основі багатьох грубих набросків логотипу та встановлених цілей було розроблено різні концепції логотипу «BugHive». Після вибору найбільш привабливих концепцій було прийнято рішення щодо вибору елементів дизайну, таких як символи, шрифти та кольори. Важливим аспектом було забезпечити відповідність цих елементів назві застосунку.

Результатом цього етапу є готовий логотип для «BugHive» (див. рис. 3.1), який відображає цілі бренду, передає його цінності та створює впізнаваність серед конкурентів.

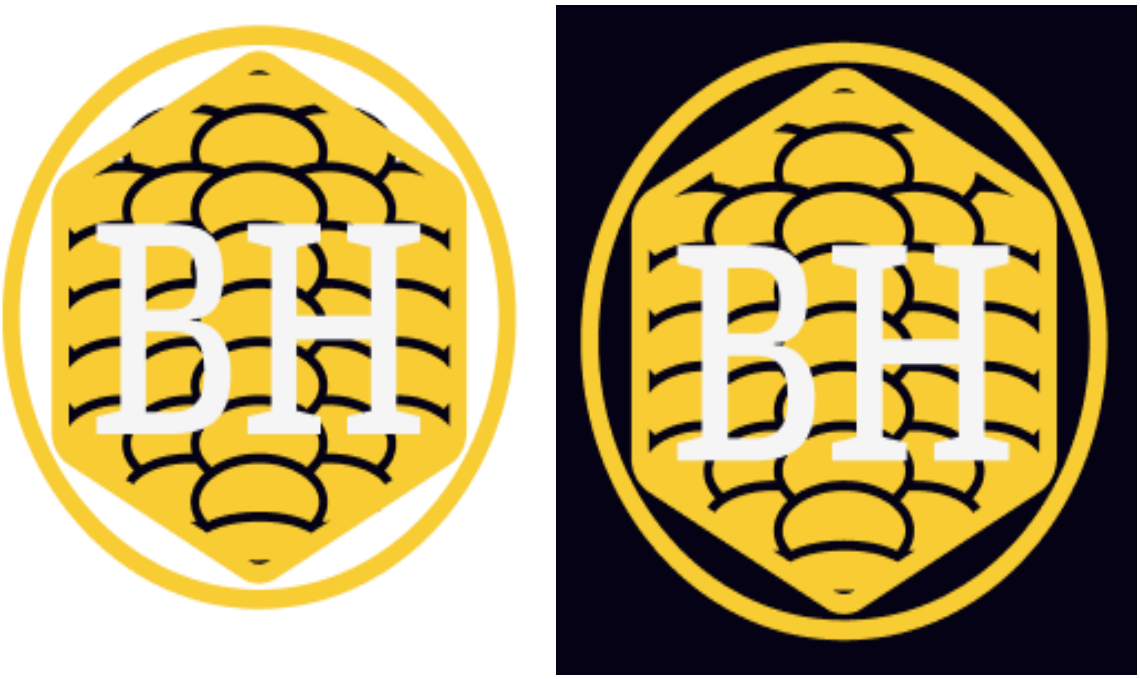


Рисунок 3.1 – Логотип вебзастосунку “BugHive”

## 3.2 Встановлення середовища розробки React.js та додаткових бібліотек

Першим кроком є вибір та встановлення текстового редактора з-за допомогою якого буде створюватися вебзастосунок на React.js. З-поміж декількох популярних варіантів, таких як Visual Studio Code, Sublime Text, Atom та WebStor вибір пав на Visual Studio Code (див. рис. 3.2).

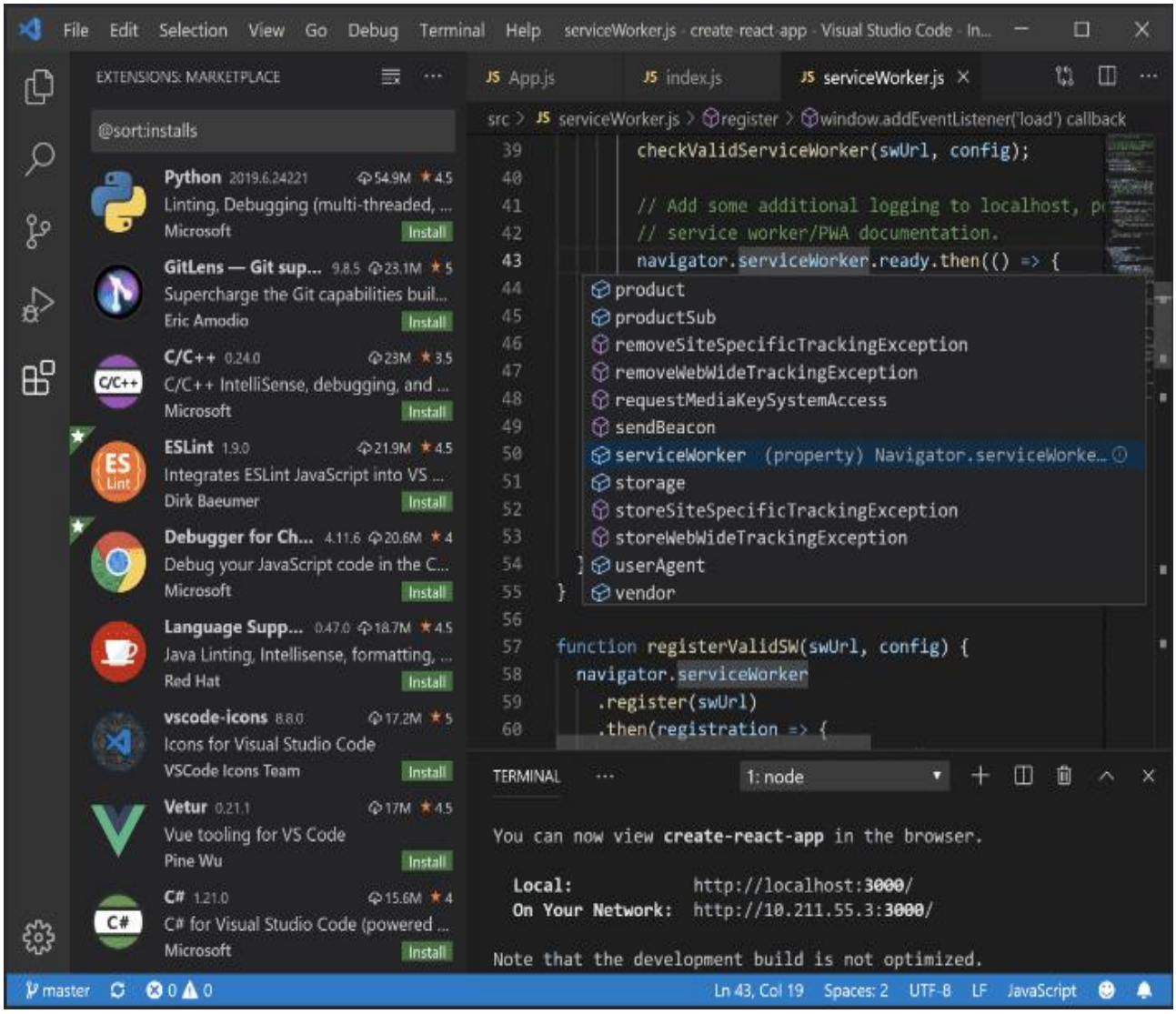


Рисунок 3.2 – Текстовий редактор VS Code

VS Code має кілька переваг, які зробили його популярним серед розробників [20]:

– має потужну систему розширень;

– інтеграцію з різними інструментами розробки, такими як системи контролю версій Git;

– надає потужні інструменти для розробки на React та JavaScript

### 3.2.1 Підключення Node.js

Наступним кроком є встановлення платформи Node.js та диспетчера пакетів NPM. React.js потребує наявності цих компонентів для подальшої розробки та запуску.

NPM – це менеджер пакетів для Node.js з відкритим вихідним кодом, створений, щоб допомогти розробникам JavaScript легко обмінюватися упакованими модулями коду.

Реєстр npm – це загальнодоступна колекція пакунків з відкритим кодом для Node.js, інтерфейсних вебдодатків, мобільних додатків, роботів, маршрутизаторів та інших потреб JavaScript-користувачів [21].

### 3.2.2 Створення React-проєкту

При створенні проєкту ще одним важливим кроком є перевірка наявності компонентів за допомогою команди “node –v” для перевірки версії Node.js і “npm –v” для перевірки версії NPM. Якщо компоненти не встановлені, слід встановити їх, перейшовши на вебсайт Node.js і завантаживши останню версію Node.js.

Для використання інструменту create-react-app спочатку потрібно встановити його глобально для системи (див. рис. 3.3): npm install -g create-react-app. Цей інструмент автоматично налаштовує структуру проєкту, Webpack та Babel, без необхідності налаштовувати всі ці речі вручну [22].

Та створюємо сам проєкт (див. рис. 3.3): create-react-app my-test-site.

```
C:\Users\vladi>D:

D:\>cd React

D:\React>npm install create-react-app
npm WARN saveError ENOENT: no such file or directory, open 'D:\React\package.json'
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN enoent ENOENT: no such file or directory, open 'D:\React\package.json'
npm WARN React No description
npm WARN React No repository field.
npm WARN React No README data
npm WARN React No license field.

+ create-react-app@3.4.1
added 98 packages from 46 contributors and audited 98 packages in 15.262s
found 0 vulnerabilities


D:\React>create-react-app my-test-site

Creating a new React app in D:\React\my-test-site.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...


> core-js@2.6.11 postinstall D:\React\my-test-site\node_modules\babel-runtime\node_modules\core-js
> node -e "try{require('./postinstall')}catch(e){}"


> core-js@3.6.5 postinstall D:\React\my-test-site\node_modules\core-js
> node -e "try{require('./postinstall')}catch(e){}"


> core-js-pure@3.6.5 postinstall D:\React\my-test-site\node_modules\core-js-pure
> node -e "try{require('./postinstall')}catch(e){}"

+ cra-template@1.0.3
+ react-dom@16.13.1
+ react@16.13.1
+ react-scripts@3.4.1
added 1616 packages from 751 contributors and audited 1620 packages in 181.758s
found 1 low severity vulnerability
  run `npm audit fix` to fix them, or `npm audit` for details
```

Рисунок 3.3 – Створення React проєкту

Після успішного створення React-проєкту переходимо до його директорії та запускаємо за допомогою наступних команд (див. рис. 3.4): cd my-test-site, npm start.

```
D:\React>cd my-test-site

D:\React\my-test-site>npm start

> my-test-site@0.1.0 start D:\React\my-test-site
> react-scripts start
? Something is already running on port 3000.

Would you like to run the app on another port instead? Yes
i [wds]: Project is running at http://192.168.0.55/
i [wds]: webpack output is served from
i [wds]: Content not from webpack is served from D:\React\my-test-site\public
i [wds]: 404s will fallback to /
Starting the development server...
Compiled successfully!

You can now view my-test-site in the browser.

  Local:            http://localhost:3001
```

Рисунок 3.4 – Завантаження проєкту

При коректному виконанні всіх кроків створення та завантаження проєкту в браузері має відкритися стандартна сторінка React (див. рис. 3.5).

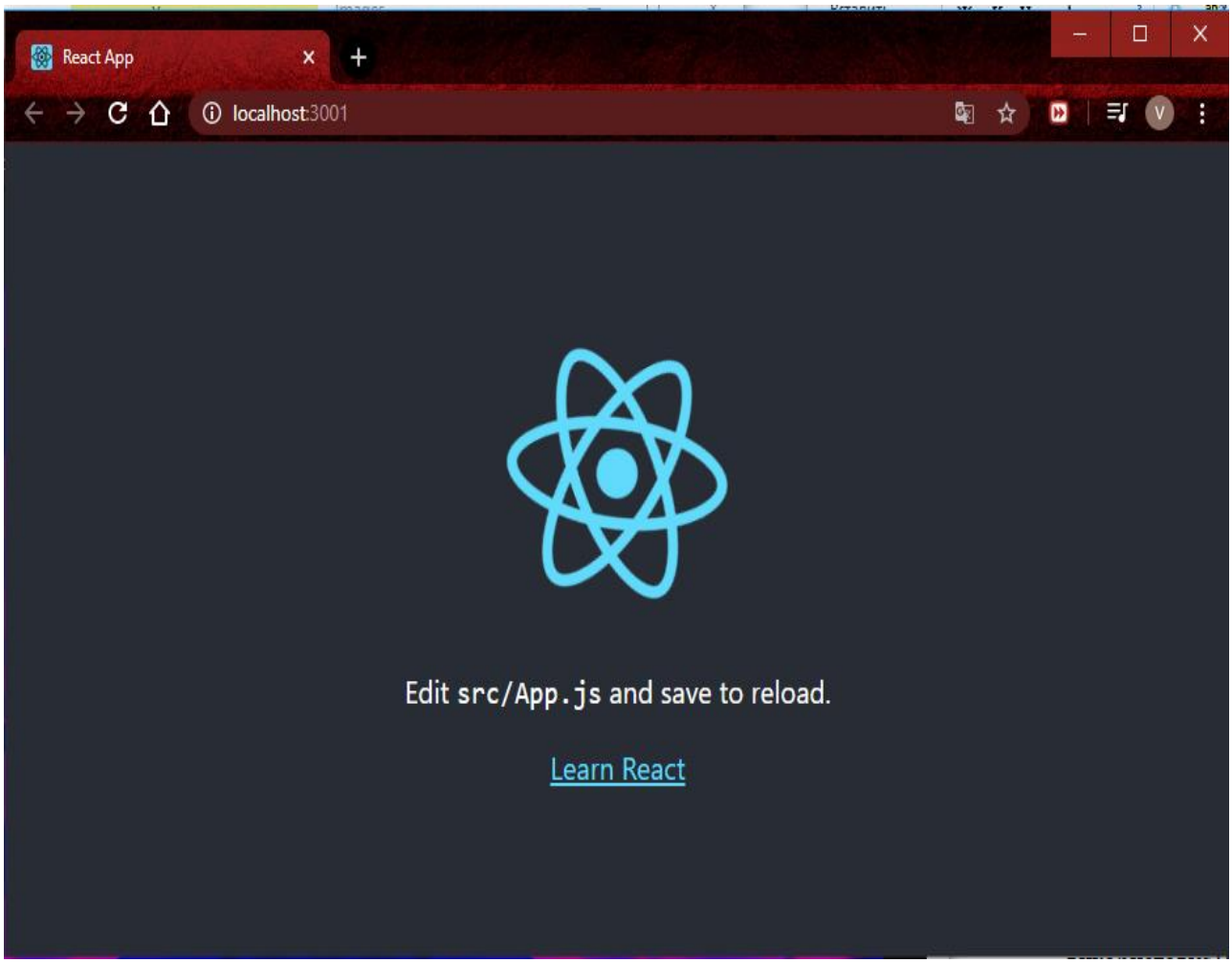


Рисунок 3.5 – Стандартна сторінка React

У процесі створення React-проєкту, ви можливе підключення додаткових бібліотек, які допоможуть вам розширити функціональність та поліпшити вигляд проєкту. Нижче наведено кілька популярних бібліотек, які використовувались для створення проєкту.

Redux – бібліотека управління станом, яка використовувалась для полегшення спільного використання даними між компонентами [23]. Для підключення в терміналі в теці проєкту потрібно додати строку (див. рис 3.6): npm install redux react-redux.

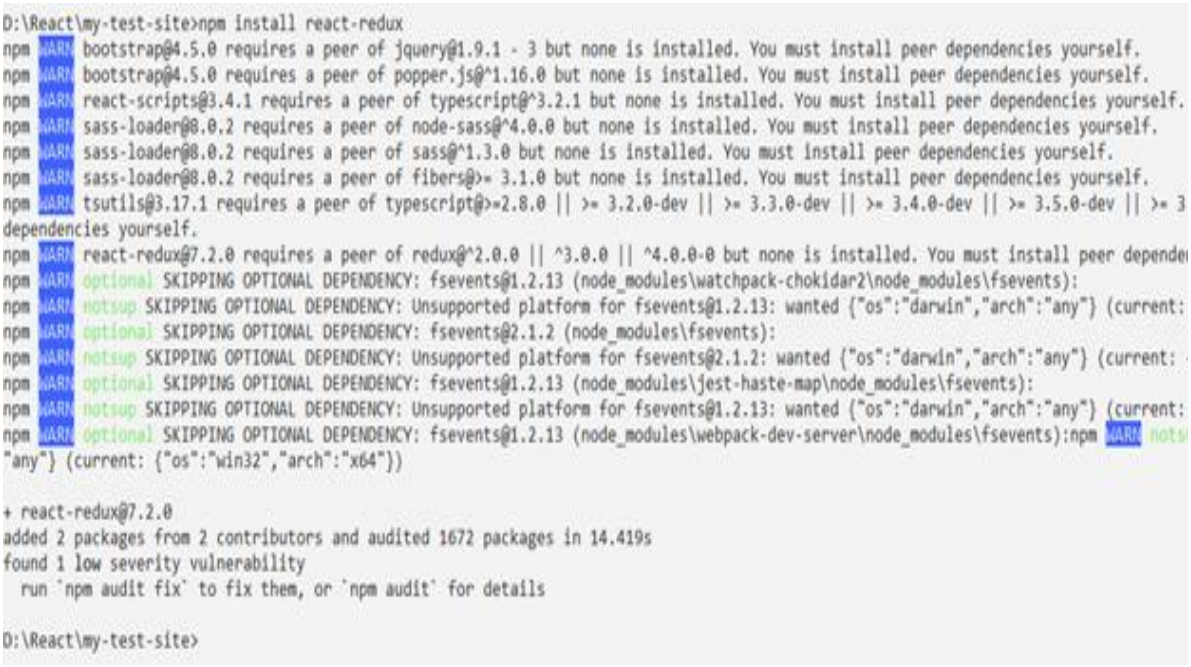


Рисунок 3.6 – Підключення React-Redux

Приклад використання Redux для створення дії (actions) для різних операцій, пов’язаних з багами (див. рис. 3.7).

bugActions.js

```
export const addBug = (bug) =>
{
  return {
    type:
     'ADD_BUG',
    payload:
     bug
  };
};
export const resolveBug = (bugId) => {
  return {
    type:
     'RESOLVE_BUG',
    payload:
     bugId
  };
};
```

Рис. 3.7 – Приклад дії (actions) пов’язаної з багами

Встановлення Material-UI та його залежності, за допомогою команди: npm install @mui/material @emotion/react @emotion/styled.

Імпортувавши компоненти Material-UI у файлі, можна їх використовувати, налаштовувати вигляд та стиль за допомогою пропсів та тем: import { Button, TextField } from '@mui/material'.

Ця бібліотека надає багато готових компонентів та стилів для швидкої і красивої розробки інтерфейсу вашого проєкту [24].

Axios – це бібліотека, яка дозволяє виконувати HTTP-запити з додатку. Вона надає простий та зрозумілий API для взаємодії з вебсерверами та здійснення запитів до API [25].

Основні методи, які доступні в Axios:

– axios.get: виконує GET-запит до вказаної URL-адреси та повертає проміс з результатом;

– axios.post: виконує POST-запит до вказаної URL-адреси з переданими даними та повертає проміс з результатом;

– axios.put: виконує PUT-запит до вказаної URL-адреси з переданими даними та повертає проміс з результатом;

– axios.delete: виконує DELETE-запит до вказаної URL-адреси та повертає проміс з результатом;

– axios.patch: виконує PATCH-запит до вказаної URL-адреси з переданими даними та повертає проміс з результатом;

– axios.request: повертає об’єкт обіцянки (Promise) зі спеціально налаштованою конфігурацією.

Для встановлення Axios, в командній строці виконується команда: npm install axios

Прикладом використання axios в розробленому проєкті є HTTP-запит який через метод post за певною модельною структурою описаною на бекенді, створює новий проєкт в базі даних. В разі виключення, перехоплюється за допомогою .catch() та виводить помилку до консолі (див. рис. 3.8).

```
const handlePlusClick = () =>
{
   axios.post('http://localhost:3001/project-add')
    .then((response) =>
{
      const createdProject = response.data.project;
      setData([...data, createdProject]);
    }
)
    .catch((error) =>
    {
      console.log('Error:', error);
    }
    );
  };
```

Рисунок 3.8 – Приклад застосування axios

### 3.3 Клієнтська частина вебдодатку

Розробка користувацького інтерфейсу почалася зі створення макетів сторінок проєкту, визначення дизайну, підбору кольорової палітри, типографіки та розбір місцеположення елементів і компонентів, які потрібно реалізувати. Все це розроблялося в застосунку Figma.

Figma – це дуже зручний онлайн-сервіс розробки інтерфейсів з можливістю прототипування.

Після визначення головних аспектів перейшов до створення статичних зображень макетів. Макети представляють собою візуальні представлення окремих сторінок та елементів інтерфейсу. Вони відображають орієнтацію, компонування, колірну схему, типографіку та стилізацію, що будуть використовуватися в розробленому вебдодатку (див. рис. 3.9).

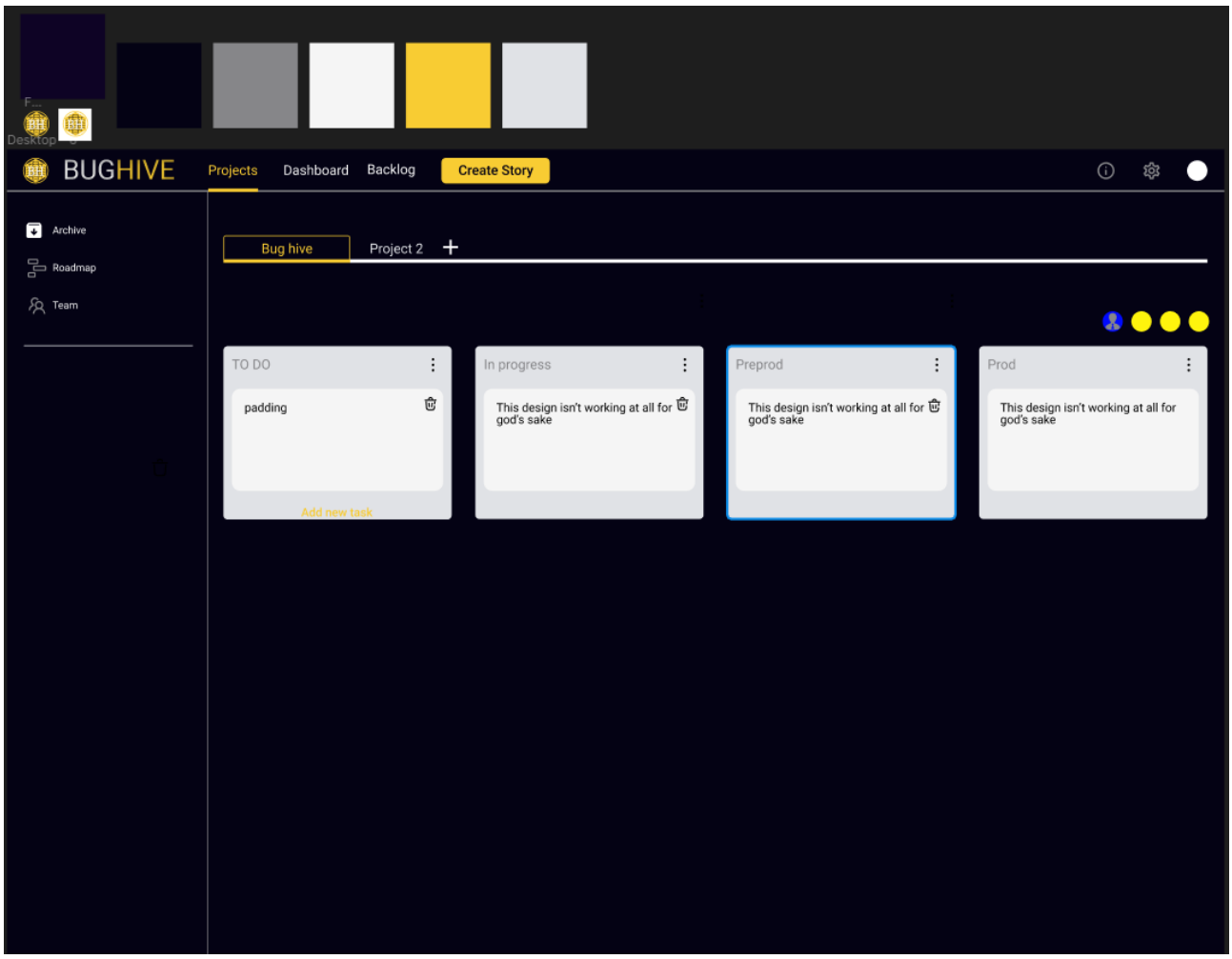


Рисунок 3.9 – Макет головної сторінки з компонентом дошки проєкту

Після розробки макетів сторінок проєкту у Figma, перейшов до наступного етапу – розробка цих макетів на реальний користувацький інтерфейс. Для цього використовувалися можливості React.js, який дозволив перетворити макети у функціональні компоненти.

За його допомогою створив компоненти, які відповідають кожній сторінці та елементу, що були визначені у макеті. Наприклад, для головного меню створив окремий компонент, в якому розміщені необхідні елементи: лого, назва, кнопки та зображення (див. рис. 3.10).

Аналогічно, для інших сторінок та компонентів також створюються відповідні компоненти. Наприклад компонент Project (див. рис. 3.11) складається ще з двох окремо описаних компонентів: шапка (див. рис. 3.12) та контент проєкту (див. рис. 3.13) тобто сама дошка багів.

Далі наведений компонент ProjMenu, який являється шапкою компоненти Project.

```
const NavBar = () => {
  return (
    <nav className="navbar__1">
      <div className="left__nav">
      <div className="nav__logo">
        <img src={logo} alt="Logo" />
      </div>
        <h1 className="Name_Bughive">BUGHIVE</h1>
        </div>
      <div className="nav-buttons">
        <Link to="/projects">
          <button  className={activeButton  ===  0  ?  'active'  :  ''}  onClick={()  =>
setActiveButton(0)}>Projects</button>
        </Link>
        <Link to="/dashboard">
          <button  className={activeButton  ===  1  ?  'active'  :  ''}  onClick={()  =>
setActiveButton(1)}>Dashboard</button>
        </Link>
        <Link to="/people">
          <button  className={activeButton  ===  2  ?  'active'  :  ''}  onClick={()  =>
setActiveButton(2)}>People</button>
        </Link>
        <button className="Create_button">Create</button>
      </div>
      <div className="right__nav">
        <img src={settings} alt="settings" />
        <img src={user} alt="user" /></div></nav>};


export default NavBar;
```

Рисунок 3.10 – Окремий компонент головного меню NavBar

```
const Projects = () => {
   const [selectedTab, setSelectedTab] = useState(0);
   const [selectedProject, setSelectedProject] = useState(data[0]);
   const handleProjectSelect = (project) => {
    setSelectedProject(project); };
   const handleStateChange = (newState) => {
    setSelectedProject((prevProject) => ({ ...prevProject,
     tabs: newState[selectedProject.id]?.tabs || [],}));};
   console.log('Обраний проєкт:', selectedProject);
return (
   <div className="class__home">
    <div className="main__container">
     <div className="main__container__menu">
     <ProjMenu projects={data}selectedProject={selectedProject}
       onProjectSelect={handleProjectSelect}/></div>
     <div className="main__column">
      <div className="task__container">
      <MainProjdata={selectedProject.tabs}
        selectedProject={selectedProject}
        onStateChange={handleStateChange}/></div>
     </div></div></div>);};
export default Projects;
```

Рисунок 3.11 – Компонент Project

```
const ProjMenu = ({ projects, selectedProject, onProjectSelect }) => {
return (
<div className="container">
    <div className="bloc-tabs">
     {data.map((project) => (
      <button
       key={project._id}
       onClick={() => handleProjectSelect(project)}
       className={project._id === selectedProject?._id ? 'tabs active-tabs' :
'tabs'}>{project.title}
      </button>))}
     {!showInput && (
      <button className={"plus_proj"}  onClick={handlePlusClick}>
      <img src={plus} alt="plus" />
     </button>)}
     {showInput && (
      <div className="new-project">
       {/* <input type="text" placeholder="Enter project name"
        value={newProjectName}
        onChange={handleInputChange}
        onBlur={handleInputBlur} /> */}
      </div>
    </div>
   </div>  );};
export default ProjMenu;
```

Рисунок 3.12 – Компонент ProjMenu

Та компонент MainProj, який являється контентом проєкту або дошкою багів зі своїми компонентами такими як модальне вікно(popup), Drag and Drop тощо.

```
function MainProj({data, selectedProject}) {
  return (
    <div className="App">
      <DragDropContext onDragEnd={handleDragEnd}>
        {_.map(state, (data, key) => {
          const droppableId = key.toString();
          return(
            <div key={key} className={"column"}>
              <Droppable droppableId={droppableId}>
              <Draggable/>
              </Droppable>
              <div className={"add_item"}>
          <button className={"plus"} onClick={() => addItem(key)}><img src={plus}
alt="plus" /></button>
            </div>
        </div>) })}
        </DragDropContext>
        <div className={"add_state"}>
         <button className={"plus"} onClick={addState}>
           <img src={plus} alt="plus" />
         </button>
         </div>
      </div>);}
export default MainProj;
```

Рисунок 3.13 – Компонент MainProj

Під час розробки користувацького інтерфейсу використовувався CSS для структури та стилізації компонентів. Завдяки CSS можна задавати розміщення елементів, кольори, шрифти та інші властивості, щоб відтворити дизайн, який був визначений у макеті. Одним із важливих підключень є reset.css який несе мету скинути та нормалізувати стилі за замовчуванням, що використовуються браузерами для HTML-елементів.

### 3.4 Серверна частина вебдодатку

Для серверної частини, використовувалось середовище виконання – Node.js та програмний каркас для серверних частин – express.js.

Перш ніж розпочати роботу зі серверною частиною, потрібно доінсталювати express.js, виконавши команду в терміналі “npm install express”. Ця команда встановить express.js і залежності, необхідні для його роботи в проєкті.

На серверній частині виконувались дії описані далі.

Підключення до бази даних: за допомогою пакету mongoose, який дозволяє здійснювати з’єднання з MongoDB базою даних (див. рис. 3.14). Після встановлення з’єднання з базою даних, використовуються параметри useNewUrlParser та useUnifiedTopology для забезпечення сумісності з останніми версіями MongoDB та перевіряється на збої у з’єднанні.

```
require('dotenv').config();
mongoose.connect(
process.env.MONGOOSECONNECT, {
  useNewUrlParser: true, useUnifiedTopology: true,}).then(() => {
    console.log('Connected to MongoDB');}).catch((error) => {
    console.log('Error connecting to MongoDB:', error); });
```

Рисунок 3.14 – Підключення до бази даних

Запити до змін в базі даних: використовується модель Project з пакету projV2.js, яка відповідає за взаємодію з базою даних. Для отримання записів з бази даних використовується метод find(), який повертає всі проєкти (див. рис. 3.15). Для створення нового проєкту використовується метод save(), який зберігає новий проєкт у базі даних.

```
app.get('/projects', (req, res) => {
  Project.find()
    .then((projects) => {
      if (!projects) {
        return res.status(404).json({ error: 'No projects found' }); }
      res.json(projects);})
    .catch((error) => {
      console.log('Error retrieving projects:', error);
      res.status(500).json({ error: 'Failed to retrieve projects' });});
});  });
```

Рисунок 3.15 – Запит для зчитування проєктів з бази даних

Схеми моделей для бази даних: використовується схема моделі Project, яка описує структуру даних проєкту. Схема визначає поля, типи даних та будь-які обмеження для цих полів. У даному випадку, модель Project має поля title, description, creator, team, дата створення та зміни, columns і кожна колонка містить поля name та bugs тощо (див.рис. 3.16) Це дозволяє створити консистентну структуру для зберігання проєктів у базі даних.

```
import mongoose from 'mongoose';
const projectSchema = new mongoose.Schema({
  id: { type: String, },
  title: {: String, },
  description: { type: String, },
  creator: { type: mongoose.Schema.Types.ObjectId,
    ref: 'Admin', },
  team: { type: mongoose.Schema.Types.ObjectId,
    ref: 'User', },
  columns: [
    { id: { type: String, },
      name: { type: String, },
      bugs: [ { id: { type: String, },
                  title: {type: String, required: true, },
                 description: {type: String, required: true, },
          priority: { type: String, required: true, },
          status: { type: String, required: true, },
              ], },  ],
}, {timestamps: true,});
export default mongoose.model('Project', projectSchema);
```

Рисунок 3.16 – Модельна схема проєкту

Загальною метою цих дій є забезпечення взаємодії серверної частини з базою даних, зберігання і отримання даних про проєкти, а також забезпечення правильної структури даних за допомогою схем моделей.

### 3.5 Висновки до 3 розділу

В третьому розділі будо розглянуто створення логотипу, переглянуто процес встановлення середовища розробки для React.js. Були розроблені логотип, встановлено середовище розробки React.js, реалізована клієнтська та серверна частини вебдодатку. Завдяки цьому забезпечив привабливий та функціональний вебсайт, який готовий до подальшого розвитку та додавання нових функцій.

# ВИСНОВКИ

В ході написання кваліфікаційної роботи розглянуто розроблений баг-трекінговий застосунок з використанням MERN (MongoDB, Express.js, React і Node.js) технологій, основною метою якого є створення системи для ефективного відстеження помилок застосунків з використанням сучасних інструментів та методів. Проаналізовано та виявлено переваги сучасних застосунків для відслідковування помилок. Розглянуті сучасні технології, такі як React для реалізації інтерфейсу користувача, Express.js для створення серверної частини, а також MongoDB для зберігання даних. Це дозволило досягти високої швидкодії та забезпечити ефективну роботу з даними.

Процес розробки поділений на кілька етапів. Починаючи з розробки логотипу та встановлення середовища розробки React.js, до реалізації клієнтської та серверної частини застосунку. Було створено моделі даних, налаштовано підключення до бази даних MongoDB та розроблено API для взаємодії з клієнтом.

В результаті тестування розробленого застосунку було виявлено, що він успішно виконує свої основні функції – відстеження помилок, користувачі мають можливість додавати нові проєкти, колонки, баг-репорти, встановлювати пріоритети та статуси для кожної помилки, а також отримувати звіти та статистику про роботу з проєктами.

# ПЕРЕЛІК ПОСИЛАНЬ

1. What is Agile? Atlassian. URL: https://www.atlassian.com/agile (дата звернення: 28.03.2023).

2. Top 6 Best Issue Tracking Systems in 2023. Brainhub: Software Development Agency. URL: https://brainhub.eu/library/best-issue-tracking-systems (дата звернення: 28.03.2023).

3. How to Choose Bug Tracking Software: Jira and Its Alternatives – QA Madness. QA Madness. URL: https://www.qamadness.com/how-to-choose-bug-tracking-software/ (дата звернення: 28.03.2023).

4. About. Bugzilla. URL: https://www.bugzilla.org/about/ (дата звернення: 30.03.2023).

5. Overview – Redmine. URL: https://www.redmine.org/projects/redmine/wiki (дата звернення: 18.02.2023).

6. React – JavaScript-бібліотека для створення користувацьких інтерфейсів. URL: https://uk.legacy.reactjs.org/ (дата звернення: 22.04.2023).

7. Початок роботи – React. React – JavaScript-бібліотека для створення користувацьких інтерфейсів. URL: https://uk.legacy.reactjs.org/docs/getting-started.html (дата звернення: 18.04.2023).

8. Redux Fundamentals, Part 3: State, Actions, and Reducers. Redux – A predictable state container for JavaScript apps. URL: https://redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers (дата звернення: 18.04.2023).

9. Overview – Material UI. MUI: The React component library you always wanted. URL: https://mui.com/material-ui/getting-started/overview/ (дата звернення: 22.04.2023).

10. Tutorial v6.11.2. Home v6.11.2. React Router. URL: https://reactrouter.com/en/main/start/tutorial (дата звернення: 18.04.2023).

11. PaulHalliday. How To Use Axios with React. DigitalOcean. The Cloud for

Builders. URL: https://www.digitalocean.com/community/tutorials/react-axios-react (дата звернення: 20.04.2023).

12. Node.js – Express Framework. Online Courses and eBooks Library. URL: https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm (дата звернення: 20.04.2023).

13. What is a database?. Oracle. Cloud Applications and Cloud Platform. URL: https://www.oracle.com/database/what-is-database/ (дата звернення: 20.04.2023).

14. DBMS. Types of Databases – javatpoint. www.javatpoint.com. URL: https://www.javatpoint.com/types-of-databases (дата звернення: 22.04.2023).

15. About Us – Our Story. MongoDB. MongoDB. URL: https://www.mongodb.com/company (дата звернення: 22.04.2023).

16. Етапи створення веб сайтів – WebTune. Webtune. URL: https://webtune.com.ua/statti/web-rozrobka/etapy-stvorennya-veb-sajtiv/ (дата звернення: 22.04.2023).

17. Діаграми UML для моделювання процесів і архітектури проєкту. Evergreen – web розробка і діджиталізація бізнесу за допомогою AI продуктів. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html (дата звернення: 18.04.2023).

18. CORE – Aggregating the world’s open access research papers. URL: https://core.ac.uk/download/pdf/11320265.pdf (дата звернення: 18.04.2023).

19. Жихаревич В. Професійна практика програмної інженерії. Чернівці, 2014. 384 с.

20. Microsoft. Visual Studio Code Frequently Asked Questions. Visual Studio Code – Code Editing. Redefined. URL: https://code.visualstudio.com/docs/supporting/faq (дата звернення: 26.03.2023).

21. Npm About. npm. URL: https://www.npmjs.com/about (дата звернення: 26.03.2023).

22. Getting Started. Create React App. Create React App. URL: https://create-

react-app.dev/docs/getting-started (дата звернення: 18.03.2023).

23. Redux Fundamentals, Part 3: State, Actions, and Reducers. Redux – A predictable state container for JavaScript apps. URL: https://redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers (дата звернення: 22.04.2023).

24. Overview – Material UI. MUI: The React component library you always wanted. URL: https://mui.com/material-ui/getting-started/overview/ (дата звернення: 20.04.2023).

25. Axios API. Axios. URL: https://axios-http.com/docs/api_intro (дата звернення: 20.04.2023).