Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему «Розробка Web-додатку Military-енкциклопедія»

Виконав: студент групи ІПЗ19-1
Спеціальність 121 «Інженерія програмного забезпечення»
Карпінський Б. В.
(прізвище та ініціали)

Керівник к. т. н., доц.Чупілко Т.А.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент: Департамент з питань цифрового розвитку, цифрових трансформацій та цифровізації ДМСУ
(місце роботи)
головний державний інспектор відділу розробки та супроводження програмного забезпечення
(посада)
Бахтін О. В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2023

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці Web-додатку «Military-енциклопедія» на основі метаданих API Wikipedia.

Метою роботи є створення зручного та доступного додатку для отримання інформації про військові події, озброєння, військову історію тощо.

У роботі використовується API Wikipedia для отримання актуальних та достовірних даних.

Кваліфікаційна робота містить наступні результати: розроблено функціональний дизайн Web-додатку, імплементовано метадані API Wikipedia для отримання потрібної інформації, створено інтерфейс для зручного використання додатку, проведено тестування та оцінку його ефективності.

Наукова новизна полягає у використанні метаданих API Wikipedia для створення Military-енциклопедії та практичне значення полягає у створенні зручного додатку, який може бути використаний для отримання інформації.

У першому розділі наведено теоретичні відомості про використані інструменти для вирішення задачі, проведено аналіз та порівняльну характеристика технологій для розробки проектованого ПЗ, зроблено вибір інструментів для вирішення поставленого завдання на основі проведеного аналізу.

У другому розділі проведено аналіз предметної області та специфікації вимог, надана коротка характеристика об'єкту автоматизації та предметної галузі, проведено огляд аналіз існуючих аналогів, наведено специфікацію вимог до системи.

У третьому розділі проведено практичну реалізацію проектованої системи, надано інформаційне, технічне та системне забезпечення розробки.

Ключові слова: WEB-ДОДАТОК, MILITARY-ЕНЦИКЛОПЕДІЯ, МЕТАДАНІ, API WIKIPEDIA.

ABSTRACT

The qualification work is devoted to the development of a web application "Military-Encyclopedia" based on Wikipedia API metadata.

The aim of the work is to create a convenient and accessible application for obtaining information about military events, weapons, military history, etc.

The work uses the Wikipedia API to obtain up-to-date and reliable data.

The qualification work contains the following results: a functional design of the Web application was developed, Wikipedia API metadata was implemented to obtain the necessary information, an interface for convenient use of the application was created, and its effectiveness was tested and evaluated.

The scientific novelty lies in the use of Wikipedia API metadata to create a Military Encyclopedia and the practical significance lies in the creation of a user-friendly application that can be used to obtain information.

The first section provides theoretical information about the tools used to solve the problem, analyzes and compares the technologies for developing the designed software, and selects tools to solve the problem based on the analysis.

The second section analyzes the subject area and requirements specification, provides a brief description of the automation object and the subject area, reviews the analysis of existing analogs, and presents the specification of requirements for the system.

In the third section, the practical implementation of the designed system is carried out, information, technical and system support for the development is provided.

Keywords: WEB APPLICATION, MILITARY ENCYCLOPEDIA, METADATA, WIKIPEDIA API.

ЗМІСТ

ВСТУП

XXI століття характеризується швидким розвитком інформаційних технологій та їх впливом на різні сфери нашого життя. Інтернет став невід'ємною частиною нашої повсякденності, надаючи нам доступ до безлічі інформації. Але не завжди існує не перевантажений, простий та зручний в використанні сервіс а головне із перевіреною та актуальною інформацією

У сучасному світі вже існують деякі веб-енциклопедії та платформи, присвячені військовим темам. Однак, багато з них мають обмежені можливості, складний інтерфейс або не забезпечують достатньо актуальну інформацію а інформація з тих чи інших причин не відповідає дійсності. Розробка «Military-енциклопедія» вирішує ці проблеми шляхом створення простого у використанні та зручного веб-додатку, який надає доступ до актуальних даних про військову сферу.

Метою даної кваліфікаційної роботи є розробка веб-додатку, котрий базується на використанню сучасних технологій, інструментів та засобів розробки веб-додатків, метаданих API Wikipedia.

Сьогодення характеризується швидким розвитком інформаційних технологій та їх впливом на різні сфери нашого життя. Інтернет став невід'ємною частиною нашої повсякденності, надаючи нам доступ до безлічі інформації. Військова справа не стала винятком і вимагає актуальної та достовірної інформації для опису своїх рішень. Це спонукало до ідеї розробки Web-додатку «Military-енциклопедія», який є надійним джерелом військової інформації та дозволяє зручно отримувати доступ до широкого спектру знань про військові події, озброєння, військову історію та інше.

Актуальність розробки веб-додатку веб-додатку «Military-енциклопедія» полягає в постійному попиті на інформацію , яка стає дедалі важливішою для обізнаності звичайних громадян, дослідників, журналістів, просто любителів військової тематики. Автоматизація процесу збору та

представлення інформації в онлайн-форматі дозволить зробити його більш ефективним та доступним для широкого кола користувачів.

Об'єктом дослідження кваліфікаційної роботи є технології, інструменти та засоби розробки веб-додатків.

Предметом дослідження є розробка веб-додатку «Military-енциклопедія», включаючи вибір оптимальних технологій, розробку функціонального дизайну та інтуїтивно зрозумілого інтерфейсу.

Методи дослідження: метод спостереження, метод узагальнення, метод аналізу, метод синтезу, гіпотетичний метод, метод емпіричного рівня, методи експериментально-теоретичного рівня.

Завдання дослідження:

– провести аналіз предметної області з обґрунтуванням актуальності теми кваліфікаційної роботи;

– сформулювати постановку задачі до кваліфікаційної роботи;

– проаналізувати методи розробки, що можуть бути використані для вирішення поставленої задачі;

– розробити макет додатку;

– розробити фронтенд-частину додатку;

– розробити бекенд-частину додатку;

– скласти інструкцію використання для користувача.

В результаті виконання завдання очікується, що розроблений веб-додаток буде мати такі технічні характеристики, як простий зрозумілий інтерфейс, можливість швидкого пошуку інформації, інтуїтивна навігація між розділами та можливість збереження обраних статей для подальшого використання. Він забезпечить широкому колу користувачів легкий та зручний доступ до інформації про мілітарі тематику, що сприятиме збільшенню їхньої обізнаності та розуміння важливих аспектів військової історії та впливу конфліктів на вигляд сучасності.

В результаті виконання завдань кваліфікаційної роботи бакалавра можна підтвердити наступні програмні результати, що відповідають освітній програмі 121 «Інженерія програмного забезпечення»:

ПР01 – аналіз, цілеспрямований пошук і вибір необхідних для вирішення професійних завдань інформаційно-довідникових ресурсів і знань з урахуванням сучасних досягнень науки і техніки;

ПР03 – знання основних процесів, фаз та ітерацій життєвого циклу програмного забезпечення;

ПР06 – уміння вибирати та використовувати відповідну задачі методологію створення програмного забезпечення;

ПР08 – вміння розробляти людино-машинний інтерфейс;

ПР11 – вибір вихідних даних для проектування, керуючись формальними методами опису вимог та моделювання;

ПР18 – знання та вміння застосовувати інформаційні технології обробки, зберігання та передачі даних;

ПР23 – вміння документувати та презентувати результати розробки програмного забезпечення.

У процесі розробки веб-додатку «Military-енциклопедія» необхідно забезпечити правильну організацію роботи, уникнення непередбачених проблем та досягнення високої якості результуючого продукту.

Кваліфікаційна робота має детально описати процес розробки Web-додатку «Military-енциклопедія» і надати практичні рекомендації та інструкції з його використання. Результати цієї роботи сприятимуть поліпшенню доступу до перевіреної інформації та покращенню розуміння сфери військової справи.

Структура кваліфікаційної роботи: вступ, три розділи, висновки, список використаних джерел, що містить 43 джерела, три додатки (мають 3 таблиці). Робота містить 27 рисунків, 60 сторінок.

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області веб-додатку «Military-енциклопедія»

У сучасному світі інформація про мілітарі тематику, зброю, військові конфлікти та історичні події має велике значення для багатьох груп користувачів, таких як військові, історики, студенти, любителі військової тематики та багато інших. Існує попит на надійне та зручне джерело інформації, яке охоплює широкий спектр тематики, пов'язаної зі зброєю та військовими питаннями. Розробка веб-додатку «Military-енциклопедія» є актуальною, оскільки вона відповідає потребам цільової аудиторії, яка прагне отримати доступ до достовірної та зручної інформації про мілітарі тематику.

На сьогоднішній день існують деякі подібні додатки та веб-сервіси, які надають інформацію про військову тематику. Наприклад, Military.com є популярним англомовним веб-порталом, який пропонує новини, статті та ресурси, пов'язані з військовою сферою. Іншим прикладом є GlobalSecurity.org який зосереджений на зброї, військовій стратегії та глобальній безпеці. Ці сервіси мають велику кількість інформації, проте їхній функціонал та дизайн можуть бути складними для користувачів, особливо для новачків.

Хоча подібні сервіси вже існують, деякі з них можуть мати свої обмеження. Наприклад, веб-сервіси можуть мати обмежену базу даних або не забезпечувати достатньо широкий охоплюючий контент або взагалі розповсюджувати інформацію яка не відповідає дійсності та дискредитує відповідний аспект. Крім того, складний інтерфейс та незручний пошук можуть ускладнювати пошук інформації для користувачів. Розробка нового додатку дозволить уникнути таких обмежень і надати зручну та зрозумілу платформу для отримання військової інформації.

Деякі з причин, які обґрунтовують доцільність та необхідність розробки нового сервісу, включають:

1. Зручний та легкий доступ до інформації: Розробка простого та зрозумілого інтерфейсу дозволить користувачам швидко знаходити потрібну інформацію без зайвих складнощів.

2. Актуальність та оновлення: Розробка нового додатку дозволить забезпечити актуальну та оновлену інформацію щодо мілітарі тематики, події та історичні факти.

3. Пристосування до різних аудиторій: Новий додаток буде розроблений з урахуванням різних груп користувачів, надаючи змогу знайти інформацію, яка цікава саме для них. Наприклад, для військових професіоналів, студентів, істориків або журналістів, редакторів різних видань або любителів військової тематики.

4. Сучасний дизайн та функціонал: Новий додаток матиме сучасний дизайн і інтуїтивно зрозумілий функціонал, що зробить його привабливим для користувачів.

Загалом, розробка нового веб-додатку «Military-енциклопедія» є актуальною через потребу в зручному та доступному джерелі інформації про військову тематику. Розробка з простим, зручним функціоналом та дизайном дозволить задовольнити потреби різних груп користувачів та забезпечити актуальну та перевірену інформацію.

1.2 Аналіз варіантів вирішення завдання

Аналіз існуючих варіантів вирішення поставленого завдання, а саме розробки веб-додатку подібного масштабу та його опрацювання, є важливою та складною проблематикою. Даний аналіз передбачає ретельне дослідження різних підходів, методів та інструментів, які можуть бути використані для створення функціонального та ефективного веб-додатку в одночас з його простотою роботи та функціональністю, що задовольняє вимоги поставленої задачі.

Починаючи з аналізу можливостей технологій, переваг мов програмування вибору фреймворків, системи управління базами даних, дослідник має зрозуміти, який стек технологій буде найбільш доцільним та сприятиме ефективній розробці веб-додатку. Важливо врахувати функціональні вимоги, масштабованість, швидкодію та можливості розширення, щоб вибрати оптимальний набір інструментів.

Крім того, аналіз включає вивчення різних архітектурних рішень. Необхідно розглянути, які підходи до організації компонентів веб-додатку будуть найбільш ефективними з точки зору функціональності та масштабованості. Дизайн інтерфейсу також є одним з ключових аспектів, які потрібно ретельно проаналізувати, оскільки він впливає на користувальницький досвід та зручність використання веб-додатку.

Забезпечення безпеки також важливо врахувати під час аналізу. Необхідно розглянути потенційні загрози та визначити заходи, які можуть бути прийняті для захисту веб-додатку від атак. Крім того, важливо розглянути питання масштабованості, щоб забезпечити стабільну та ефективну роботу додатку при збільшенні обсягів трафіку та користувачів.

Усі ці аспекти аналізу існуючих варіантів вирішення поставленого завдання допоможуть визначити найоптимальніший шлях розв'язання задачі та реалізувати веб-Додаток, який відповідає вимогам і сприяє досягненню поставлених цілей. Такий аналіз є важливим кроком перед початком самої розробки, оскільки він допомагає зрозуміти потенційні труднощі та вибрати оптимальні рішення для успішної реалізації проекту.

У контексті веб-розробки, розумна стратегія полягає в тому, щоб обирати технології, які найкраще підходять для конкретного проекту. Навіть якщо існують універсальні технології, які можуть самостійно забезпечити потрібний результат, зазвичай використовується комбінація різних інструментів, щоб досягти оптимального результату.

Однак, широкий вибір доступних технологій веб-додатків може створювати проблему вибору найкращого рішення. При виборі стеку

технологій необхідно враховувати різні аспекти конкретного проекту. Важливо проаналізувати вимоги щодо функціональності, масштабованості, безпеки та продуктивності, а також врахувати специфіку розробки зовнішнього та внутрішнього програмного коду.

Правильний вибір стеку технологій може мати вирішальний вплив на продуктивність та якість веб-додатку. Відповідно, процес аналізу та вибору відповідних інструментів є критичним етапом перед розробкою. Варто ретельно вивчити та порівняти можливості різних технологій, оцінити їхню сумісність та підтримку специфічних вимог проекту.

Загалом, збалансований підхід до аналізу і вибору стеку технологій веб-розробки дозволяє забезпечити оптимальну комбінацію інструментів, що підвищує продуктивність розробки та якість результуючого веб-додатку. Ретельний аналіз різних аспектів, таких як технології, архітектура, дизайн та безпека, допомагає визначити найкращі рішення для досягнення поставлених цілей веб-додатку [1].

1.3 Оцінка доступних засобів розв'язання поставленої задачі

Аналіз існуючих варіантів вирішення поставленого завдання, а саме розробки веб-додатку «Military-енциклопедія», включає порівняння різних шляхів реалізації, включаючи мови програмування та фреймворки для фронтенду та бекенду, а також бази даних.

Для кращого розуміння, почнемо з розгляду мов розмітки та стилізації, таких як HTML та CSS.

HTML (HyperText Markup Language): HTML є основною мовою розмітки веб-сторінок. Вона використовується для створення структури та визначення елементів на веб-сторінці. HTML забезпечує семантичну структуру документа, що дозволяє пошуковим системам та іншим інструментам краще розуміти вміст сторінки.

CSS (Cascading Style Sheets): CSS використовується для стилізації веб-сторінок, задання зовнішнього вигляду та макету. Він дозволяє визначати кольори, шрифти, розміри, позиціонування елементів та інші аспекти дизайну. CSS забезпечує більш гнучкий та контрольований підхід до стилізації, порівняно з готовими UI стилями.

У порівнянні з готовими UI стилями, які можуть бути використані для швидкої розробки інтерфейсу, використання CSS надає більшу гнучкість та можливість налаштування стилів під конкретні потреби проекту. Водночас, готові UI стилі можуть забезпечити швидшу розробку та консистентний вигляд інтерфейсу.

Для розробки веб-додатку «Military-енциклопедія» можна використовувати різні мови програмування та фреймворки для реалізації фронтенду та бекенду.

Давайте розглянемо деякі з них:

1. JavaScript (JS). Одна з найпопулярніших мов програмування, яка використовується для розробки веб-додатків. Вона має широку підтримку, багато ресурсів та велику спільноту розробників [2].

Фреймворки: для фронтенду можна використовувати React.js, Angular або Vue.js, які надають потужні інструменти для створення інтерфейсу [3].

2. TypeScript є розширенням мови JS, яке додає статичну типізацію та інші функціональні можливості, підтримується багатьма фреймворками, дозволяє знизити кількість помилок і покращити підтримку коду [4].

Фреймворки: TypeScript можна використовувати з React.js, Angular або Vue.js, працюючи з ними на базі JavaScript [3].

3. Python є високорівневою, інтерпретованою мовою програмування з акцентом на простоту та читабельність коду. Вона має широке застосування в різних областях розробки, включаючи веб-розробку [5].

Фреймворки: для реалізації веб-додатку можна використовувати фреймворки, такі як Django або Flask, які забезпечують швидку розробку та масштабування [6].

4. C# є мовою програмування, розробленою Microsoft, і використовується для розробки різноманітних додатків, включаючи веб-додатки. Вона має сильну типізацію і широкий набір інструментів [7].

Фреймворки: для розробки веб-додатку на C# можна використовувати ASP.NET або ASP.NET Core, які надають потужні засоби для створення веб-сайтів та веб-служб [8].

5. Java є високопродуктивною мовою програмування, яка використовується для розробки різноманітних додатків, включаючи веб-додатки. Вона працює на віртуальній машині Java (JVM) і має багату екосистему [9].

Фреймворки: для розробки веб-додатків на Java можна використовувати фреймворки, такі як Spring або JavaServer Faces (JSF), які надають широкий спектр інструментів та можливостей [10].

6. PHP є мовою програмування, спеціально розробленою для розробки веб-додатків, що має простий синтаксис та велику спільноту розробників [11].

Фреймворки: для розробки веб-додатків на PHP можна використовувати фреймворки, такі як Laravel або Symfony, які надають потужні засоби для створення веб-сайтів та додатків [12].

У виборі мови програмування та фреймворку важливо враховувати наявність документації, активність спільноти розробників, масштабованість, продуктивність та власні навички [13].

Складемо таблицю порівняння мов програмування для фронтенду та бекенду (Додаток А).

1.4 Аналіз можливостей використання СУБД

Щодо баз даних, основні типи баз даних, які можна використовувати для зберігання даних в веб-додатку, включають: реляційні бази даних (наприклад, MySQL, PostgreSQL) та нереляційні бази даних (наприклад, MongoDB, CouchDB). Перші використовуються для структурованого зберігання даних,

забезпечують можливості для виконання складних запитів та мають широку підтримку, а другі зберігають дані у вигляді документів та надають гнучкість при зміні схеми даних.

Розглянемо більш детально особливості та недоліки деяких згаданих СУБД та наведемо приклади проєктів, для яких вони можуть використовуватися.

1. Oracle.

Особливості: Oracle є потужною реляційною базою даних з широким набором функціональних можливостей. Вона підтримує транзакції, кластеризацію, реплікацію та інші розширені функції.

Недоліки: Oracle може бути складною у налаштуванні та має високі вимоги до ресурсів. Ліцензійні витрати також можуть бути значними.

Приклади проєктів: Oracle широко використовується в корпоративних системах, банківських системах, електронній комерції та інших великих проєктах [14].

2. MySQL.

Особливості: MySQL є легкою у використанні реляційною базою даних з доброю швидкодією. Вона підтримує стандарти SQL і має велику спільноту розробників.

Недоліки: MySQL може бути менш масштабованою в порівнянні з деякими іншими базами даних і може мати обмеження в розширених функціях.

Приклади проєктів: MySQL застосовується в багатьох веб-додатках, блогах, форумах, системах керування вмістом та інших середнього розміру проєктах [15].

3. PostgreSQL.

Особливості: PostgreSQL є потужною реляційною базою даних з багатьма продвинутими функціями. Вона підтримує повний стандарт SQL, розширені можливості геоданих та динамічні запити.

Недоліки: PostgreSQL може мати вищі вимоги до ресурсів порівняно з деякими іншими базами даних. Деякі функції можуть бути складними у використанні.

Приклади проєктів: PostgreSQL використовується в системах управління відносинами з клієнтами (CRM), геодезичних системах, системах електронного голосування та інших проєктах [16].

4. MongoDB.

Особливості: MongoDB є документ-орієнтованою нереляційною базою даних, яка дозволяє гнучко зберігати та організовувати дані. Вона має гнучку схему та підтримує горизонтальне масштабування.

Недоліки: MongoDB може бути менш швидкою за деякими реляційними базами даних, а також може вимагати більше місця для зберігання даних.

Приклади проєктів: MongoDB використовується в системах управління контентом, журналістиці, аналітиці даних та інших проєктах, де гнучкість схеми даних є важливою [17].

5. CouchDB.

Особливості: CouchDB є нереляційною базою даних типу документ, яка підтримує реплікацію та розподілену архітектуру. Вона забезпечує легку реплікацію та синхронізацію даних.

Недоліки: CouchDB може бути менш швидкою у порівнянні з деякими іншими базами даних, а також може мати обмежені можливості для складних запитів та аналітики даних.

Приклади проєктів: CouchDB використовується в системах синхронізації даних між пристроями, колективній роботі над документами та інших проєктах, де реплікація та легка синхронізація є важливими [18].

6. Redis.

Особливості: Redis є базою даних типу ключ-значення, яка забезпечує дуже високу швидкодію та низьку латентність. Вона підтримує широкий набір структур даних та функцій кешування.

Недоліки: Redis не має повноцінної підтримки SQL і може бути обмеженою в розширених функціях реляційних баз даних.

Приклади проєктів: Redis використовується для кешування даних, розподіленого сесійного сховища, реалізації черг повідомлень та інших проєктах, де вимагається висока швидкодія [19].

У поділі баз даних за типом архітектури реляційні бази даних (наприклад, Oracle, MySQL, PostgreSQL) орієнтовані на зберігання структурованих даних та використовують SQL для операцій з даними. Нереляційні бази даних (наприклад, MongoDB, CouchDB) зберігають дані у вигляді документів або ключ-значення і забезпечують гнучкість при зміні схеми даних. Порівняльна таблиця для баз даних Oracle, MySQL, PostgreSQL, MongoDB, CouchDB, Redis за об'єктно-орієнтованими можливостями та функціональними можливостями (Додаток А).

1.5 Оптимізація ресурсів розробки

Важливо розглянути оптимізацію ресурсів розробки, зокрема використання спільної мови програмування для фронтенду і бекенду. Якщо наприклад фронтенд реалізований на JavaScript, то розумно розглянути можливість також використовувати JavaScript для бекенду. Це дозволяє ефективніше використовувати ресурси розробки, так як з'являється можливість використовувати спільні підходи і інструменти для розробки як фронтенду, так і бекенду.

Наприклад, комбінація JavaScript на фронтенді з Node.js на бекенді є досить поширеною і забезпечує єдину мову програмування для всього стеку додатка. Це дозволяє переходити з однієї частини розробки додатка до іншої без значних зусиль, а також сприяє підтримці коду і взаємодії між фронтендом і бекендом.

Однак, варто враховувати особливості проекту та вимоги до швидкодії, масштабованості та інших факторів. Наприклад, якщо в проекті вимагається

висока продуктивність або використовуються специфічні функціональні можливості, можуть бути розглянуті інші комбінації мов та фреймворків.

У таблиці надано порівняльну характеристику деяких комбінацій мов програмування для фронтенду та бекенду(Таблиця А.2)

1.6 Висновки до першого розділу

Таким чином, провівши аналіз можливих засобів для розробки веб-додатку «Military-енциклопедія» можна дійтти до висновку доцільності використовувати комбінацію JavaScript або TypeScript для фронтенду з використанням фреймворку React.js або Vue, та Node.js з фреймворком Express.js для бекенду. У випадку бази даних можна вибрати між реляційною базою даних, такою як MySQL, або нереляційною базою даних, наприклад MongoDB, залежно від специфіки завдання проекту. Важливо також пам'ятати, що вибір конкретних технологій залежить від багатьох факторів, таких як рівень володіння мовами програмування, досвід розробки, масштаб проекту, вимоги до продуктивності та безпеки.

Отже, аналіз предметної області «Military-енциклопедія» показує її актуальність, потребу в достовірній інформації, використання сучасних технологій та сприяє розвитку обізнаності та освіти. Враховуючи ці аспекти, можна сформулювати постановку задачі та продовжити процес розробки веб-додатку «Military-енциклопедія». Створення «Military-енциклопедія» сприятиме покращенню обізнаності користувачів яких цікавить мілітаарі тематика. Інформаційний ресурс надасть можливість детально ознайомитися з різними аспектами військової сфери, що сприятиме освіті та розвитку інтересу до даної галузі.

РОЗДІЛ 2.
АНАЛІЗ ВИРІШЕННЯ ЗАДАЧІ РОЗРОБКИ ВЕБ-ДОДАТКУ «MILITARY-ЕНЦИКЛОПЕДІЯ»

Для успішної реалізації розробки веб-додатку «Military-енциклопедія» було проведено детальний аналіз та вибір стеку технологій MongoDB, Express JS, React JS та Node JS що являє собою стек MERN . Вибір стеку технологій є критичним етапом, оскільки він визначає ефективність, швидкодію та розширюваність додатку.

2.1 Обґрунтування вибору стеку технологій для розробки

Обґрунтування вибору стеку MERN для реалізації веб-додатку «Military-енциклопедія»:

1. Спільна мова програмування: Однією з основних переваг використання стеку MERN є те, що весь додаток розробляється з використанням JavaScript. Це означає, що для розробки використовується одна мова програмування як для фронтенд-частини (React), так і для бекенд-частини (Node.js), що спрощує розробку, підтримку, та масштабування додатку.

2. Ефективна комунікація між компонентами: Використання React у фронтенді дозволяє розбити інтерфейс на незалежні компоненти, які можуть ефективно комунікувати між собою.

У React компоненти можуть взаємодіяти через передачу пропсів (props) та виклик функцій з батьківського компонента до дочірніх. Це дозволяє передавати дані та властивості між компонентами і реагувати на їх зміни. Крім того, React має можливість використовувати стан (state), що дозволяє зберігати та оновлювати дані в межах компонента.

Цей підхід розділення на компоненти дозволяє розподілити функціональність додатку на малий набір самодостатніх компонентів, які

можуть бути перевикористані та змінені без впливу на інші частини коду додатку. Таким чином, забезпечується модульність, розширюваність та підтримка коду.

Використання React для ефективної комунікації між компонентами допомагає додатку "Military-енциклопедія" швидко реагувати на зміни та забезпечити перевикористання компонентів, що сприяє швидкій роботі додатку та покращує користувацький досвід.

3. Гнучкість нереляційної бази даних: MongoDB, яка є нереляційною базою даних в стеці MERN, використовується для зберігання та організації даних, пов'язаних з користувачами додатку "Military-енциклопедія". У контексті розробки додатку , MongoDB використовується для зберігання інформації про користувачів, таку як їх логіни та захешовані паролі, а також для зберігання списку обраних статей користувачами.

Гнучкість MongoDB дозволяє змінювати схему даних легко та швидко. Це дозволяє додавати нові поля або змінювати існуючі, що є корисним у випадку зберігання додаткової інформації про користувачів або розширення функціональної частини щодо реалізації особистого кабінету користувача

Використання MongoDB для зберігання даних користувачів та обраних статей у додатку "Military-енциклопедія" допомагає забезпечити ефективне та гнучке управління інформацією користувачів та забезпечує зручний функціонал для обрання статей та збереження їх в обліковому записі користувача.

4. Фреймворк для розробки бекенду: Express.js, що використовується в стеку MERN, є легковаговим фреймворком для створення API та обробки запитів на бекенді. Він дозволяє швидко створювати серверну частину додатку з підтримкою маршрутизації, middleware та інших функціональностей.

Обумовленість вибору стеку MERN полягає в тому, що він забезпечує повну функціональність для розробки веб-додатку «Military-енциклопедія» Використання єдиного мови програмування, гнучкої бази даних та зручного фреймворку для бекенду дозволяє ефективно розробляти та підтримувати

Додаток. Крім того, стек MERN є популярним і має велику спільноту розробників, що забезпечує доступ до багатьох ресурсів та підтримку.

Приклади подібних додатків, які використовують стек MERN, включають "Facebook", "Instagram" та "Airbnb". Ці додатки демонструють ефективність та масштабованість стеку MERN у реальних проектах. Вибір MERN для розробки «Military-енциклопедія» обумовлений його зручним функціоналом, широким спектром можливостей та сприятиме розвитку стабільного та сучасного веб-додатку.

2.2 Розробка технічного завдання

Розробку технічного завдання варто почати з макету додатка. Необхідно продумати макет сайту згідно якого буде створено дизайн та загальний вигляд сайту додатку.

Також оскільки в якості стеку технологій обрано стек MERN мовою програмування відповідно як для бекенд-частини так і для фронтенд-частини буде JavaScript, що дає змогу зручно та безперешкодно розробляти обидві частини додатку.

Для отримання даних про мілітарі тематику буде використано API Wikipedia [23]. API Wikipedia - це програмний інтерфейс, який надає доступ до величезного обсягу структурованої інформації з Вікіпедії. Він дозволяє отримувати дані з Вікіпедії, такі як статті, категорії, зображення тощо, і використовувати їх у власних програмах або веб-додатках [24].

2.2.1 Макет та його дизайн. Загальний стиль та макет. Макет буде

простим та зрозумілим для користувачів, з дотриманням принципів зручного та інтуїтивно зрозумілого дизайну.

Дизайн буде відповідати тематиці військової енциклопедії, може містити елементи, що асоціюються зі зброєю.

Меню та навігація. Сайт додатку матиме кілька основних кнопок для навігації котрі будуть являти собою перехід на відповідні сторінки:

Info – інформація про Додаток, сторінка де описано призначення додатку,

Home – головна сторінка з елементами навігації по розділам ,

Search – пошук для пошуку необхідної інформації,

Contact – контакти з розробником та підтримка в випадку проблем з роботою додатку та пропозицій

User(іконка) - особистий кабінет користувача, та можливість реєстарації та авторизації

Також для зручності буде створену кнопку для швидкої прокрутки вгору, при наведенні та кліку відбуватиметься прокрутка до верху сайту.

Кольорова палітра. Варто розглянути використання мілітарі стилю, стриманих кольорів, таких як зелений, хакі, чорний, сірий, бежевий, жовтий. Важливо підібрати кольорову гаму таким чином, щоб вона підкреслювала серйозність та авторитетність військової тематики.

Далі наведемо список кольорів які можна використати для дизайну додатку в HEX-палітрі. HEX-палітра – це набір кольорів, який використовує шістнадцяткову систему числення для представлення кольорів. Кожен колір у HEX-палітрі кодується за допомогою комбінації шести шістнадцяткових символів (цифри 0-9 та літери A-F). Кожний символ в HEX-коді представляє положення інтенсивності червоного (R), зеленого (G) та синього (B) каналів кольору [20].

Наведемо список із палітри HEX яку буде використано [21]:

- Dark Green: #164A41;
- Medium Green: #4D774E;
- Light Green: #9DC88D;
- Natural Yellow: #F1B24A;
- White: #FFFFFF.

Типографіка. Вибір шрифтів повинен бути заснований на принципах зручності та читабельності. Рекомендується використовувати чисті та сучасні шрифти, які відповідають загальному стилю додатку.

В рамках додатку використовуватиметься наступний шрифт та його властивості: Font-family: 'Exo 2', sans-serif; – сімейство шрифтів [22]. Font-stretch: normal; - накреслення тексту.

Вигляд статей та інформаційних блоків. Статті та інформаційні блоки будуть відображатись зрозуміло та зручно для користувачів. Після завантаження будь якої із статей буде відображено текст та за потреби короткий інформаційний блок з правого боку вікна браузеру, також додатково в кінці кожної статті буде розташовано інформаційні блоки в вигляді трьох елементів:

1. Елемент з кнопкою на всю ширину вікна браузеру приховати-показати, в елементі буде відображено відповідні до статті таблиці.

2. Елемент з кнопкою на всю ширину вікна браузеру приховати-показати, в елементі буде відображено відповідні посилання на джерела інформації відповідно до контенту статті.

3. Елемент з кнопкою на всю ширину вікна браузеру приховати-показати, в даному елементі буде відображатись весь медіа контент а саме зображення, схеми, gif-зображення відповідно до статті.

2.2.2 Технічне рішення для фронтенд-частини додатку. Додаток матиме вигляд односторінкового (Single Page Application, SPA) додатку, побудованого на основі React JS.

SPA (Single Page Application) – це тип веб-додатку, який працює на одній сторінці, без перезавантаження сторінки при взаємодії користувача. В SPA весь необхідний HTML, CSS та JavaScript завантажується разом з першою сторінкою, а далі взаємодія з сервером відбувається асинхронно шляхом обміну даними у форматі JSON [25].

У контексті проекту, розробюваний проект є SPA додатком, що означає, що весь веб-інтерфейс додатку буде побудований на одній сторінці. React JS використовується як основа для розробки SPA додатку. Завдяки використанню React JS, зможемо створити реактивний інтерфейс, який буде оновлюватись та відображати нові дані без необхідності перезавантаження сторінки.

React JS – це відкрита JavaScript-бібліотека, яка використовується для розробки інтерфейсу користувача. Він дозволяє створювати компоненти, які відображають різні частини інтерфейсу та забезпечують їх взаємодію [26]. Однією з головних переваг React JS є використання JSX (JavaScript XML) для опису компонентів.

JSX є розширенням синтаксису JavaScript, яке дозволяє писати HTML-подібний код прямо в JavaScript. Це спрощує створення компонентів, оскільки вони можуть включати як HTML-подібну розмітку, так і JavaScript-логіку. JSX забезпечує зручний та ефективний спосіб опису вигляду і поведінки компонентів у React [27].

Також при розробці додатку буде застосовуватись функціональний підхід який є основною концепцією при розробці компонентів у React. Компоненти можуть бути написані у вигляді функцій, які отримують вхідні дані (props) та повертають JSX-елементи, які відображають вміст компонента. Цей підхід дозволяє писати прості, перевикористовувані та легко масштабовані компоненти [28].

Також буде використовуватися бібліотека react-router-dom@6 для керування маршрутизацією в SPA додатку. React Router є популярним роутинговим рішенням для React-додатків, яке дозволяє нам визначати та керувати різними маршрутами нашого додатку без перезавантаження сторінки.

З використанням React Router зможемо визначати маршрути, пов’язані з різними компонентами, і налаштовувати їх показ та поведінку. Це дозволить нам перехоплювати та обробляти URL-адреси, змінювати компоненти, які

відображаються на сторінці, і забезпечувати навігацію в додатку без перезавантаження [29, 30].

Використання react-router-dom@6 дозволяє нам побудувати маршрутизацію в нашому SPA додатку, де різні URL-шляхи будуть відповідати різним компонентам, що відображатимуться на сторінці. Це дозволить нам створювати навігаційні посилання, здійснювати переходи між сторінками та забезпечувати зручну навігацію для користувачів.

У проекті також буде використовуватися бібліотека Axios для здійснення HTTP-запитів з SPA додатку до сервера бази даних. Axios є популярною бібліотекою для здійснення асинхронних запитів з використанням Promise-об'єктів.

Promise-об'єкти є частиною стандарту JavaScript ES6 і використовуються для управління асинхронними операціями. Вони представляють результат асинхронної операції, яка може мати два стани: виконано (resolved) або відхилено (rejected). Promise-об'єкти дозволяють працювати з асинхронними запитами, які повертають результати пізніше, і виконувати певні дії після їх завершення, наприклад, обробляти результати або виконувати наступні дії у ланцюжку запитів [31].

Цей підхід дозволяє зручно працювати з асинхронним кодом, замість того, щоб вкладати його в глибокі колбеки. Використання Promise-об'єктів спрощує управління асинхронними операціями, дозволяє створювати послідовність запитів та ефективно обробляти їх результати.

Axios надає зручний і простий інтерфейс для виконання HTTP-запитів, таких як GET, POST, PUT, DELETE тощо. Він підтримує обробку запитів на основі обіцянок (Promises), що дозволяє зручно обробляти результати запитів і оброблювати помилки [32].

Детальніше про використання Axios у сполученні з React можна прочитати в офіційній документації Axios, де наведені приклади використання та описані різні можливості цієї бібліотеки [32].

2.2.3 Технічне рішення для бекенд-частини додатку. Оскільки для реалізації додатку використовуватиметься стек MERN, то відповідно платформою буде Node.js.

Node.js є відкритим середовищем виконання JavaScript, яке дозволяє виконувати JavaScript-код на сервері. У контексті розроблюваного додатку, Node.js використовується як програмна платформа для реалізації бекенд-частини. Використання Node.js матиме кілька переваг для розробки додату. Перш за все, воно забезпечує швидку та ефективну обробку запитів, оскільки Node.js працює на однопотоковому, подієвому та неблокуючому вводі/виводі (non-blocking I/O). Це означає, що сервер може обробляти багато запитів одночасно без затримок, що робить його дуже масштабованим та ефективним [33].

Крім того, Node.js базується на рушії двигуну V8 JavaScript, який є швидким та потужним. Він забезпечує високу продуктивність додатку, що особливо важливо для реалізації бекенд-частини, де потрібно обробляти великий обсяг даних та виконувати складні операції [34].

Для реалізації серверної логіки та особистого кабінету користувача з функціями реєстрації, авторизації та додавання обраних статей, буде використовуватись бібліотека Express.js.

Express.js – це легковаговий фреймворк для створення веб-додатків на платформі Node.js. У контексті розроблюваного додатку, Express.js використовуватиметься для реалізації бекенд-частини особистого кабінету користувача. Express.js надає потужні інструменти для розробки серверної логіки та маршрутизації [35].

В контексті додатку, Express.js буде використовуватись для обробки запитів користувача, реєстрації, авторизації та додавання обраних статей в особистий кабінет. За допомогою Express.js можна визначати маршрути (routes) та функції-обробники (handlers), які виконують певні дії при отриманні певного запиту.

Express.js також дозволить використати middleware – проміжні програмні компоненти, які обробляють запити перед тим, як вони потрапляють до функцій-обробників [36]. Це дозволить реалізувати додаткову логіку а саме перевірку аутентифікації користувача, обробку помилок, логування тощо.

Для зберігання даних користувача, таких як логіни та захешовані паролі, а також їх обрані сторінки, буде використана база даних MongoDB. MongoDB є нереляційною базою даних, яка забезпечує гнучкість в зберіганні та організації даних [17]. Також на головній сторінці особистого кабінету авторизований користувач зможе переглядати свої збережені статті та за необхідністю переходити на них.

Для забезпечення безпеки паролів(хешування) буде використана бібліотека bcrypt. Bcrypt є алгоритмом хешування паролів, який забезпечує їх безпеку шляхом застосування солі (salt) та повторного хешування. Використання bcrypt дозволяє зберігати паролі у захешованому вигляді, що унеможливлює пряме отримання початкового паролю навіть в разі компрометації бази даних [37].

При реєстрації нового користувача, пароль буде хешуватись за допомогою bcrypt, і тільки хешоване значення буде збережено в базі даних MongoDB. При подальших процедурах авторизації, введений користувачем пароль буде знову хешуватись за допомогою bcrypt, а отриманий хеш буде порівнюватись зі збереженим хешем пароля в базі даних. Це дозволяє перевірити правильність введеного пароля без необхідності зберігання самого паролю. Використання bcrypt сприяє підвищенню безпеки паролів користувачів, зменшує ризик їх викрадення та забезпечує додатковий шар захисту для особистої інформації користувачів у розроблюваному додатку.

Для зручної роботи з MongoDB використовуватиметься об’єктно-документний відображувач (ODM) [38] - бібліотека Mongoose. Mongoose надає зручний інтерфейс для взаємодії з MongoDB, дозволяючи визначати моделі даних та виконувати різноманітні операції з базою даних [39].

Для отримання інформації в додатку буде використовуватись API Wikipedia. Це дає можливість отримувати актуальну та достовірну інформацію щодо мілітарних тематик. API Wikipedia – це інтерфейс програмування застосунків, який надає доступ до контенту та функціоналу Вікіпедії. Він дозволяє розробникам отримувати доступ до статей, категорій, зображень та іншої інформації з Вікіпедії за допомогою програмних запитів. API надає структуровані дані у форматі JSON або XML, що дозволяє легко обробляти інформацію та використовувати її у власних додатках. API Wikipedia використовує різноманітні опції та параметри запитів для отримання певних статей, списків статей за критеріями, зображень тощо. Він також підтримує пошук за ключовими словами та може повертати мовно-залежні результати.

Взаємодія з API Wikipedia буде реалізована за допомогою HTTP-запитів, використовуючи відповідні ендпоінти та параметри запитів. Додаток буде взаємодіяти з API для отримання статей, даних про події, факти та іншої інформації, пов'язаної з мілітарними темами. Отримана від API Wikipedia інформація буде використовуватись для подальшої обробки та відображення в додатку [40]. Використання API Wikipedia дозволяє забезпечити доступ користувачів до актуальних та перевірених джерел інформації, що додає додатку додатковий функціонал та цінність для користувачів, особливо в контексті мілітарних тематик.

2.3 Висновки до другого розділу

У процесі роботи було проведено детальний аналіз та порівняльну характеристику різних технологій, методів розробки та інструментів, що допомогло визначити найоптимальніший стек технологій для розробки веб-додатку. Використовуючи отримані результати, було складено технічне завдання, яке визначало функціональні вимоги до додатку, основні функції та очікувані результати.

Цей розділ кваліфікаційної роботи визначає технічні рішення та компоненти, що будуть задіяні для розробки додатку. В якості стеку буде MERN-стек, що складається з ReactJS, Node.js, Express.js та MongoDB. Додаток буде реалізовано як односторінкову програму (SPA) з використанням ReactJS, що забезпечить ефективну взаємодію користувача з інтерфейсом. Для маршрутизації буде задіяно бібліотеку react-router-dom@6, а для здійснення запитів до сервера Axios та Promise. У серверній(бекенд) частині додатку буде задіяно бібліотеку Express.js, що дозволить реалізувати логіку особистого кабінету з можливістю реєстрації, авторизації та додавання обраних статей. Для зберігання даних користувача буде задіяно базу даних MongoDB, а для зручної роботи з нею об'єктно-документний відображувач (ODM) Mongoose. Для забезпечення безпеки збереження паролів користувачів бібліотеку Bcrypt, яка забезпечить хешування паролів та перевірку їх валідності, а для отримання метаданих для заповнення додатку API Wikipedia.

В результаті, зазначені технології та компоненти буде використано для забезпечення ефективної, швидкої та безпечної реалізації бекенд-частини та фронтенд-частини користувача додатку з можливістю реєстрації, авторизації, додавання обраних статей та отримання інформації з API Wikipedia.

РОЗДІЛ 3.
ВИРІШЕННЯ ПОСТАВЛЕНОЇ В КВАЛІФІКАЦІЙНІЙ РОБОТІ ЗАДАЧІ

Заключна частина кваліфікаційної роботи є результатом розробки веб-додатку «Military-енциклопедія» відповідно до поставленої задачі і вимог технічного завдання..

В якості інструменту розробки додатку було обрано редактор коду Visual Studio Code а для зберігання версій додатку під час розробки обрано систему контролю версій Git а саме платформу GitHub.

Visual Studio Code (VS Code) – це безкоштовний, відкритий та легкий редактор коду, розроблений компанією Microsoft. Він доступний для різних операційних систем, таких як Windows, macOS і Linux, і підтримує багато мов програмування та технологій [41].

GitHub – це хостингова платформа для розробки програмного забезпечення, що забезпечує можливість зберігання, спільного доступу та керування версіями коду [42]. Він дозволяє розробникам працювати над проектами, використовуючи систему керування версіями Git [43].

3.1 Створення макету додатку

Процес розробки додатку розпочався зі створення верстки сайту, що відповідає вимогам, визначеним у технічному завданні. В рамках створення верстки було дотримано зазначених вимог щодо дизайну та вигляду додатку. Застосовані технології, такі як HTML, CSS та JavaScript, вони дозволили створити користувальницький інтерфейс, який забезпечує зручну навігацію та ефективну взаємодію з користувачем. Для досягнення відповідності верстки технічному завданню були використані передові практики розробки веб-інтерфейсу, такі як використання адаптивного дизайну, що забезпечує коректне відображення на різних пристроях, включаючи мобільні пристрої, планшети та настільні комп'ютери. Створення верстки здійснювалося з

урахуванням функціональних вимог, що були визначені у технічному завданні. Основний функціонал, такий як відображення інформації про військову тематику, можливість реєстрації та авторизації користувачів, додавання обраних статей, був врахований при розробці верстки (Рисунок Б.1). На рисунку зображено загальний вигляд додатку. Можемо побачити навігаційну панель разом із логотипом додатку, кнопка меню HOME, наступна кнопка меню SEARCH, наступна кнопка SUPPORT та остання кнопка в вигляді іконки користувача. Також на рисунку можна побачити підвал сайту котрий містить посилання на соціальні мережі, а також текст щодо авторських прав на контент використаний в проєкті. На рисунку зображено вигляд кнопки прокрутки в гору сторінки сайту (Додаток Б). Якщо кількість інформації на екрані стає більше ніж висота екрану, то в нижньому правому кутку екрану з'являється кнопка зображена на рисунку при наведенні на дану кнопку спрацьовує анімація та колір із напів-прозорого змінюється на чорний та після кліку сторінка виконує скрол до верху сторінки сайту (Додаток Б).

Далі окремо розглянемо кожен елемент навігаційної панелі та відповідні їм інтерфейси:

– кнопка із зображенням логотипу(Info) та відповідний вигляд екрану (Додаток Б). Сторінка містить текст із описом інформації про додаток та його призначення;

– кнопка навігаційного меню HOME та відповідний до неї вигляд екрану додатку (Додаток Б). Сторінка головної навігації (HOME) – після натискання кнопки HOME відображається головна навігаційна сторінка, на якій розташовані три елементи (карточки) для навігації по розділам;

– кнопка навігаційного меню SEARCH та відповідний до неї вигляд екрану додатку (Рисунок Б.4). Кнопка навігаційного меню "SEARCH" – це елемент у головному меню додатку, який відповідає за пошук інформації. Після кліку на дану кнопку, відкривається екран меню "SEARCH", який має вигляд, зображений на рисунку (Рисунок Б.4). На цьому екрані користувачу надається поле введення, де він може ввести ключові слова для пошуку

необхідної інформації всередині додатку. Крім того, на екрані присутня кнопка "SEARCH", яка запускає механізм пошуку інформації згідно введених користувачем ключових слів;

– кнопка навігаційного меню із зображенням (Особистий кабінет) та відповідний до неї вигляд функціоналу додатку (Додаток Б). Після вибору даного пункту меню з зображенням (Особистий кабінет) користувач побачить відповідний екран з двома кнопками «Registration» та «Authorization»;

– кнопка «Registration» для реєстрації користувачів – надасть можливість новим користувачам зареєструватися в додатку (Додаток Б);

– кнопка «Authorization « для авторизації користувачів – ця кнопка дозволить вже зареєстрованим користувачам увійти в додаток (Додаток Б);

– наступний рисунок демонструє вигляд екрану якщо користувач не має доданих статей оскільки користувач новий (Додаток Б);

– на наступному рисунку показано вигляд майбутньої кнопки що буде відповідати за збереження обраної статті до списку збережених сторінок користувача (Додаток Б).

3.2 Створення фронтенд-частини додатку

На другому етапі розробки додатку робота зосередилась на створенні фронтенд-частини на основі розробленого макету. Макет додатку визначає загальну структуру, розташування елементів та їх зовнішній вигляд. Створення фронтенд-частини на базі макету дозволило забезпечити відповідність зовнішнього вигляду та поведінки додатку задуманому дизайну.

Для початку розробки нашого проекту створюємо базову структуру реакт-додатку та встановлюємо необхідні пакети бібліотек. Для цього використовується команда "npx create-react-app frotnend", де "frotnend" – це назва вашого проекту. Команда "npx create-react-app" є інструментом, який дозволяє швидко створити початкову структуру проекту React з усіма необхідними налаштуваннями і залежностями. Після виконання цієї команди,

отримуємо готову початкову конфігурацію React з установленими інструментами, такими як Webpack і Babel, а також деяким початковим кодом та налаштуваннями.

Після ініціалізації проекту встановлюємо додаткові пакети бібліотек, які будуть необхідні для розробки. В даному випадку, використовуємо команду "npm" і "react-router-dom" для встановлення бібліотеки React Router DOM, яка дозволить нам працювати з маршрутизацією в нашому додатку. Також, використовуємо команду "npm install axios" для встановлення бібліотеки Axios, яка дозволить нам здійснювати HTTP-запити до сервера. Після успішного встановлення цих пакетів готові розпочати розробку фронтенд-частини нашого проекту, використовуючи React, React Router DOM та Axios.

Точкою входу а отже головним виконавчим файлом є "index.js" в даному файлі міститься головний компонент додатку "App" із файлу "App.jsx". У коді створюється кореневий React-елемент "BrowserRouter", який оточує компонент "App". Все це дерево компонентів рендериться в елемент з ідентифікатором ‘root’, який знаходиться на HTML-сторінці. Компонент "BrowserRouter" забезпечує маршрутизацію на основі URL, тобто він відображатиме різні компоненти "App" в залежності від шляху в URL (Додаток В).

Наступний файл це "App.js" який являє собою компонент "App", що визначає головний інтерфейс користувача додатку (Додаток В). У цьому фрагменті коду виконується імпорт необхідних модулів та компонентів, таких як React, стилі для додатку "App.css", "react-router-dom" для маршрутизації сторінок і компоненти: "Layout", "WarsCard", "MilitaryTechnologyCard", "MilitaryConflictsCard", "InfoPage", "Home", "Random", "SupportPage", "ContactUs", "Article", "AppWiki", "FormRegLog", "RegistrationForm", "LoginForm" та "Dashboard".

У функції "App" використовується компонент "Routes" для визначення шляхів і маршрутів. Розглянемо створені маршрути та відповідні до них компоненти більш детально окремо:

– Основний маршрут "/" відповідальний за компонент "Layout" (Додаток В).

Перш за все у створеному файлі "Layout.jsx", імпортуються необхідні модулі з бібліотеки React та react-router-dom для побудови маршрутизації веб-сторінки. Також імпортуються компоненти "BurgerMenu", "Info", "Footer" і "GoTopButton" з відповідних файлів. Сам компонент Layout є функціональним компонентом, він містить основну структуру сторінки для відображення. Компонент повертає JSX-код, який представляє розмітку сторінки. Структура включає такі елементи:

– Тег "div" з класом "wrapper", який обгортає всю сторінку.header з класом "header", який містить розмітку верхньої частини сторінки так звану шапку сайту: навігаційне меню яке складається із логотипу додатку, посилань на відповідні сторнки та іконки особистого кабінету користувача. Крім того в компонентах "NavLink" вказані шляхи маршрутизації до відповідних сторінок: Home, Search, Support. А також компонент BurgerMenu, який представляє бургер-меню.

– Далі тег "main" з компонентом "Outlet", який буде місцем виведення іншого вмісту сторінки в залежності від поточного маршруту.

– Компоненти "Footer" і "GoTopButton", які відповідають за розмітку підвалу та кнопки "Піднятися вгору" відповідно.

Загалом, даний код компонент створює основну структуру сторінки з верхньою частиною так званою шапкою сайту(логотип, навігаційне меню, іконка користувача він же особистий кабінет) і основним вмістом, що виводиться в залежності від маршруту а також підвалом та кнопкою «Піднятися вгору».

– Маршрут "/info" відповідає за компонент "InfoPage" (Додаток В).

– Даний компонент із файлу "Info.jsx", код включає в себе дві компоненти - "Info" та "InfoPage".

– Компонент "Info" є функціональним компонентом, що повертає JSX-елементи. У даному випадку, повертається фрагмент, що містить один тег "img". Тег "img" використовує атрибут "src" для вказівки шляху до зображення "../logo_12.png", яке відображає логотип додатку у шапці сайту. Також використовується атрибут "placeholder" зі значенням "INFO" в випадку якщо логотип не завантажиться.

– Компонент "InfoPage" також є функціональною компонентою, яка повертає JSX-елементи. Вона містить контейнер "div" з класом "info-page". Усередині контейнера розташований абзац "p", який містить текст. Даний текст являє собою презентацію додатку, його функціонального призначення та пояснення для чого він. В кінці усі компоненти експортуються для використання в інших частинах програми.

– Маршрут "/home" відповідає за компонент "Home", а також визначені дочірні маршрути "/list_of_war", "/MilitaryTechnologyCard" і "/MilitaryConflictsCard", які відображають відповідні компоненти "WarsCard", "MilitaryTechnologyCard" і "MilitaryConflictsCard" (Додаток В).

Відповідно до маршруту файл "Home.jsx" представляє компонент React з назвою "Home". Цей компонент відповідає за відображення головної сторінки додатка з навігацією по розділам відповідно теми.

При перегляді коду можна побачити, що імпортуються необхідні залежності з React та інших файлів. Крім того, стилізація елементів здійснюється за допомогою модуля "h.module.css".

– У функції Home виконується повернення розмітки JSX, яка включає два елементи: "div" з класом "className="main_section_one" " та "Outlet". Усередині тега "div" з класом "className="main_section_one" " розміщуються три посилання "NavLink", які вказують на різні шляхи (посилання) для навігації в додатку. Кожне посилання містить компонент "WarsCard", "MilitaryTechnologyCard" або "MilitaryConflictsCard" із файлу "CardsComponents.jsx", які відображаються як картки (Додаток В).

– Компонент "Outlet" використовується для відображення дочірніх компонентів, які вказані в маршрутах (routes) додатку. Він відображається після блоку з посиланнями.

Окремо варто сказати про навігаційні картки. Кожна картка(компонент) містить в собі HTML розмітку, яка відповідає за відображення даних картки. Наприклад, у компоненті "WarsCard" бачимо розмітку, яка містить зображення, назву "WARS" та деякі додаткові стилі, які задані за допомогою CSS модуля "s". Так само, в компонентах "MilitaryTechnologyCard" та "MilitaryConflictsCard" можна побачити відповідні дані та стилі. Кожен компонент повертає відповідну розмітку, яка потім може бути відображена на сторінці React додатку. Компоненти експортуються з модуля і можуть бути використані для відображення відповідних навігаційних карток. Загалом, цей код створює головну навігаційну сторінку додатку, на якій користувач може обрати один з трьох шляхів для навігації. Кожен шлях представляється у вигляді картки з відповідною інформацією.

Також відповідно до обраного навігаційного розділу спрацьовує маршрут "/:targetPage" та відображається компонент "AppWiki".

Функціональний компонент "AppWiki" із створеного файлу "AppWiki.jsx" являє собою скрипт для отримання інформації за допомогою API Wikipedia в вигляді json та html-розмітки а також подальшого використання в додатку.

Розглянемо детальніше структуру файлу "AppWiki.jsx" (Додаток В):

Спочатку виконується імпорт необхідних модулів та компонентів для побудови веб-додатку з використанням бібліотеки React. Також імпортується зовнішній стиль з файлу "parseStyle.css". Далі виконується імпорт компонентів "TakeTables", "Infobox", "Navbox", "Reflist" з файлу "TakeTables.jsx" та компонента "AddPageButton" з функціоналом додавання закладки в особистий кабінет авторизованого користувача з папки "../components".

– Компонент "WikiText" відповідає за відображення тексту статті. Він отримує властивості "extract" (витяг тексту статті) та "onLinkClick" (функція,

що виконується при кліку на посилання). У компоненті визначаються функції "handleLinkClick", яка виконується при кліку на посилання і передається URL посилання, та "replaceLinks", яка замінює посилання в тексті статті на спеціальні обробники кліків. Компонент повертає блок div з класом "text", в який вставляється HTML-код тексту статті замість небезпечних символів за допомогою властивості "dangerouslySetInnerHTML".

– Компонент "WikiImages" відповідає за відображення зображень. Він отримує властивість "images" (масив зображень). Компонент повертає блок div з класом "images", в який вставляється HTML-код зображень замість небезпечних символів за допомогою властивості "dangerouslySetInnerHTML".

– Компонент "LanguageSelect" відповідає за вибір мови. Він отримує властивості "language" (поточно вибрана мова) та "onLanguageChange" (функція, що виконується при зміні мови). Компонент повертає блок div з класом "language", в якому розміщено випадаючий список (select) з варіантами мов. Вибраний варіант мови зберігається в стані "language".

– Компонент "AppWiki" є головним компонентом додатку. Він використовує хук "useParams" для отримання активного посилання сторінки. Також використовуються хуки "useState" та "useEffect" для збереження різних станів додатку та взаємодії з життєвим циклом React.

У компоненті визначені такі стани "useState":

– "isLoading" – вказує, чи триває завантаження сторінки (значення true) чи завершилося (значення false). Цей стан використовується для відображення ефекту завантаження протягом 1 секунди.

– "language" – вибрана мова. Значення цього стану залежить від вибору мови у компоненті "LanguageSelect".

– "wikiText" – текст статті з Wikipedia.

– "wikiImages" – масив зображень з Wikipedia.

– "pageData" – дані сторінки з Wikipedia.

– "isImagesBlock" – вказує, чи потрібно приховати блок зображень (значення true) чи показати його (значення false).

– "isTableHidden" – вказує, чи потрібно приховати блок таблиць (значення true) чи показати його (значення false).

– "isReflistHidden" – вказує, чи потрібно приховати блок посилань (значення true) чи показати його (значення false).

– "infoboxHTML" – HTML-код блоку додаткової-короткої інформації.

– "navboxHTML" – HTML-код блоку навігації.

– "reflistHTML"– HTML-код блоку посилань.

Також в компоненті "AppWiki" використовується хук "useEffect" для встановлення таймауту тривалістю 1 секунда. Після закінчення таймауту стан "isLoading" змінюється на значення false, що вказує на завершення завантаження сторінки. Даний функціонал із затримкою створений щоб покращити користувацький досвід, та користувач розумів що дані обраної статі завантажені і готові до використання. Далі компонент повертає JSX-код, в якому використовуються раніше описані компоненти "WikiText", "WikiImages" та "LanguageSelect", а також додатковий компонент "AddPageButton". Зазначені компоненти відповідають за відображення тексту статті, зображень та вибір мови, відповідно. Усі властивості та функції, необхідні для коректної роботи компонентів, передаються як атрибути. Зміни в стані компонента відбуваються за допомогою виклику відповідних функцій, які також передаються як атрибути.

Функція "show_hide" відповідає за переключення видимості блоку зображень. Вона використовує стан "isImagesBlock" та функцію "setIsImagesBlock" для зміни значення стану. Кожен раз, коли викликається ця функція, вона інвертує поточне значення "isImagesBlock", що призводить до показу або приховування блоку зображень на сторінці.

Аналогічно, функції "handleToggle_T" та "handleToggle_R" відповідають за переключення видимості блоків таблиць та посилань відповідно. Вони

також використовують стани "isTableHidden" та "isReflistHidden" для зміни видимості цих блоків.

Функція "getFirstPageExtract" отримує відповідь з сервера Wikipedia у форматі JSON та повертає першу частину вмісту статті. Вона використовує отримані дані для створення посилань на інші статті, які з'являються в тексті. Ці посилання замінюються на HTML-код з активними посиланнями.

Функція "getPageImages" отримує відповідь з сервера Wikipedia у форматі JSON та повертає HTML-код для відображення зображень сторінки. Вона фільтрує зображення за розширеннями файлів (jpg, jpeg, png, gif), отримує код мови статті та створює HTML-код для кожного зображення, включаючи зображення та його назву.

Функція "getPageData" отримує відповідь з сервера Wikipedia у форматі JSON та повертає об'єкт з даними про сторінку. Вона отримує заголовок, вміст, зображення та посилання сторінки з JSON-об'єкту та повертає ці дані у відповідному форматі.

Функція "handleLanguageChange" викликається при зміні вибраної мови статті. Вона оновлює стан language з вибраною мовою.

Хук "useEffect" використовується для завантаження вмісту сторінки з сервера Wikipedia. При зміні "targetPage" (цільова сторінка) відбувається запит до API Wikipedia для отримання вмісту сторінки на обраній мові, за замовчуванням використовується англійська мова. Отримані дані обробляються, інформація про таблиці, додатковий інформаційний блок та посилання вилучається з HTML-коду, а результати зберігаються у відповідні стани (infoboxHTML, navboxHTML, reflistHTML).

Другий хук "useEffect" також використовується для завантаження вмісту сторінки з сервера Wikipedia, також з урахуванням обраної мови але призначаний для отримання саме данних сторінки таких як додатковий інформаційний блок, таблиці, посилання зі статті та зображення відповідної сторінки. Він виконує запит до API Wikipedia, використовуючи вибрану мову та цільову сторінку, та обробляє отримані дані за допомогою функцій

"getPageData", "getFirstPageExtract" та "getPageImages". Результати зберігаються у відповідні стани (wikiText, wikiImages, pageData).(Додаток В)

Функція "handleLinkClick" викликається при кліку на посилання в сторінці. Вона перехоплює подію кліку, отримує URL посилання та виконує перехід до відповідної сторінки, використовуючи функцію "navigate" (перехід до нового шляху).

Останній блок коду містить JSX-розмітку, яка відображається на сторінці. Якщо "isLoading" (завантаження) є істинним, відображається повідомлення про завантаження. Інакше відображається вміст сторінки, включаючи блоки з інформацією про сторінку, текстом статті, таблицями, посиланнями та зображеннями.

Також додатково для роботи механізму отримання даних а саме для відображення додакового блоку інформації, таблиць та посилань на джерела вибраної статті використовується три компонента "Infobox", "Navbox", "Reflist" з файлу "./TakeTables.jsx" (Додаток В). Код містить компоненти "Infobox", "Navbox" та "Reflist", які використовуються для відображення додаткової інформації, таблиць та посилань(джерел) сторінки в додатку. Кожен з цих компонентів використовуватує функцію "useEffect", яка слугує для встановлення обробників подій для відповідних блоків («infobox», «navbox», «reflist»). Ці обробники викликають функцію "onLinkClick" при кліці на кнопку з назвою відповідного блоку на сторінці, виконується функція показу та приховання відповідного блоку. Кожен компонент отримує HTML-код через властивість "html" Вони використовуються для відображення відповідних блоків з HTML-кодом, дозволяючи вмісту містити активні посилання.

– Маршрут "/search" відповідає за відображення компоненту "Search" (Додаток В).

Механізм пошуку представляє собою сторінку пошуку, реалізовану з використанням бібліотеки React та пакета "react-router-dom". Він включає

HTML-форму, яка дозволяє користувачеві ввести термін пошуку, а також кнопку для пошуку.

У компоненті "SearchPage" використовуєьться хук "useState" для створення змінної "searchTerm", яка зберігає значення, введене в поле пошуку. Початкове значення "searchTerm" встановлюється як порожній рядок. Після введення терміну пошуку(ключового слова) та натискання кнопки "SEARCH", викликається функція "handleFormSubmit", яка перехоплює подію надсилання форми (event.preventDefault()) та навігує до шляху, який складається зі значення, введеного в поле пошуку "searchTerm".

Компонент "SearchPage" також включає функцію "handleInputChange", яка слухає зміни в полі введення пошуку і оновлює стан "searchTerm" залежно від введеного значення.

Маршрут "/support" відображається компонент "SupportPage" (Додаток В).

Згідно обраного маршруту відображається сторінка підтримки. На цій сторінці користувачі можуть знайти контактну інформацію для зв'язку з командою підтримки та отримати відповіді на деякі часто виникаючі питання.

Код даної сторінки написаний у створеному файлі "Support.jsx"з використанням бібліотеки React. Він починається з імпорту React та стилів для сторінки, які знаходяться в окремому файлі "h.module.css". Потім оголошується функціональний компонент "Support", який відповідає за відображення посилання на сторінку підтримки в меню. Компонент повертає відповідний HTML-код, де посилання знаходиться всередині елементу списку.

Окремо оголошується функціональний компонент "SupportPage", який відповідає за відображення вмісту сторінки підтримки. У цьому компоненті створюється HTML-структура з різними елементами, що містять інформацію про підтримку. Спочатку виводиться заголовок сторінки "Military Wikipedia Portal - Support" з використанням тега "strong". Далі йде короткий абзац, що пояснює, що користувачі можуть звернутися до команди підтримки щодо будь-яких запитів, проблем або відгуків щодо веб-порталу. Далі наведена

контактна інформація, яку можна використовувати для зв'язку з командою підтримки. Ця інформація відображається у вигляді списку з елементів, які містять адресу електронної пошти, номер телефону, обліковий запис Instagram та назву Discord-чату. Після цього наведені заголовки розділів "Frequently Asked Questions (FAQ)" і "Contact Information", після яких слідують списки з питаннями і відповідями. У списку питань і відповідей перераховані деякі загальні питання, що часто виникають у користувачів, наприклад, про навігацію по різних розділах веб-порталу, достовірність і актуальність інформації на веб-сайті, можливість внести свій внесок до порталу та наявність мобільного додатку. На кінці сторінки є ще один абзац, що закликає користувачів звертатися до команди підтримки за будь-якими іншими питаннями або потребою в додатковій допомозі.

Маршрут "/user/*" відображається компонент "FormRegLog", а також визначені дочірні маршрути "/registration", "/login" і "/dashboard", які відображають відповідні компоненти "RegistrationForm", "LoginForm" і "Dashboard".(Додаток В)

Розглянемо детальніше структуру файлу "FormRegLog.jsx":

Код, який наведено в лістингу (Додаток В), є сторінкою користувача веб-додатка. Розглянемо його функціонал починаючи з основних компонентів.

У коді використовується бібліотека React для побудови інтерфейсу користувача. Він також включає деякі компоненти з пакету "react-router-dom" для навігації між сторінками та визначення маршрутів.

Перші дві функції, "RegistrationForm" та "LoginForm", відповідають за відображення форм реєстрації та авторизації відповідно. В кожній з цих функцій використовуються станові змінні "useState" для зберігання введених користувачем даних (ім'я користувача та пароль). Функції також містять обробники подій для зміни цих значень та обробки подання форми. У разі успішної реєстрації або авторизації виконується перенаправлення на сторінку Home для подального користування додатком.

Наступна функція, "BookmarkItem", представляє елемент екрану закладки, який відображає посилання на сторінку закладки та кнопку для видалення відповідної закладки. Вона отримує дані про закладку та функцію обробки видалення із відповідного функціоналу бекенд-частини додатку, яка викликається при натисканні на кнопку видалення. Вона використовується для відображення списку закладок користувача. Спочатку вона ініціалізує станову змінну "userData" як "null". У тілі функції виконується ефект "useEffect", який викликає функцію "fetchUserData", що отримує дані про закладки користувача за допомогою запиту до сервера. Якщо отриманий статус відповіді 200, то дані про закладки зберігаються у становій змінній "userData". У разі помилки запиту, виводиться повідомлення про помилку. Функція також містить обробник "handleDeleteBookmark", який викликається при видаленні закладки. Після успішного видалення закладки з сервера, список закладок користувача оновлюється відповідно.

Остання функція «FormRegLog» відображає форму реєстрації/авторизації та панель керування користувача. Залежно від стану "isAuthenticated" (чи користувач автентифікований), відображаються відповідні компоненти. Якщо користувач не автентифікований, він може виконати реєстрацію або авторизацію, якщо він вже автентифікований, відображається панель керування користувача(екран закладок) та кнопка виходу.

3.3 Створення бекенд-частини додатку

На заключному етапі створення додатку була розроблена бекенд-частина, з фокусом на інтеграцію з розробленим фронтендом. В бекенді було реалізовано особистий кабінет користувача з таким функціоналом:

1. Реєстрація користувачів: було створено функцію, яка дозволяє користувачам зареєструватись у системі. Під час реєстрації користувачі

вводять свої дані, які зберігаються у базі даних MongoDB за допомогою бібліотеки mongoose.

2. Авторизація користувачів: було розроблено функціонал авторизації, що дозволяє користувачам увійти до свого особистого кабінету. Після введення вірних облікових даних, система перевіряє їх правильність та виділяє користувачу унікальний токен за допомогою бібліотеки bcrypt для подальшої ідентифікації.

3. Додавання та видалення статей: користувачі мають можливість додавати статті до свого списку обраних, а також видаляти їх. Для цього було реалізовано відповідні функції, які працюють з базою даних MongoDB за допомогою бібліотеки mongoose.

Розглянемо детальніше лістинг створеного коду бекенд-частини додатку:

Точкою входу та головним виконавчим є файл «index.js» який встановлює підключення до бази даних, ініціалізує та налаштовує Express-Додаток, використовуючи «authRouter» для обробки маршрутів. Він також стартує сервер на певному порті (Додаток В).

Наступний файл «authController.js» містить клас «authController», який виконує асинхронні функції обробки різних запитів, пов'язаних з аутентифікацією та керуванням користувачами (Додаток В).

Метод «registration(req, res)»: Цей метод обробляє POST-запит «/registration» для реєстрації нового користувача. Він перевіряє наявність помилок у вхідних даних, перевіряє, чи вже існує користувач з таким ім'ям, хешує пароль, створює нового користувача з роллю "USER" і зберігає його в базі даних. Після успішної реєстрації він повертає повідомлення про успішну реєстрацію у відповідь.

Метод «login»: Цей метод обробляє POST-запит «/login» для авторизації користувача. Він перевіряє наявність користувача з вказаним ім'ям, перевіряє відповідність введеного пароля збереженому хешованому паролю

користувача, створює JWT-токен з ідентифікатором користувача та його ролями, і повертає цей токен у відповідь.

Метод «getUsers»: Цей метод обробляє GET-запит «/users» для отримання списку всіх користувачів. Він отримує всіх користувачів з бази даних та повертає їх у відповідь.

Метод «addBookmark»: Цей метод обробляє POST-запит «/add-bookmark» для додавання закладки користувачеві. Він отримує інформацію про закладку(назву обраної користувачем сторінки) з тіла запиту та ідентифікатор користувача з «req.user.id». Потім він знаходить користувача в базі даних за ідентифікатором, додає закладку до списку закладок користувача та зберігає зміни. Після успішного додавання закладки він повертає повідомлення про успішне додавання у відповідь.

Метод «deleteBookmark»: Цей метод обробляє DELETE-запит «/delete-bookmark/:bookmarkId» для видалення закладки користувача за її ідентифікатором. Він отримує ідентифікатор закладки з параметра запиту та ідентифікатор користувача з «req.user.id». Потім він знаходить користувача в базі даних за ідентифікатором, знаходить індекс закладки у списку користувача, видаляє закладку та зберігає зміни. Після успішного видалення закладки він повертає повідомлення про успішне видалення у відповідь.

Метод «getBookmarks»: Цей метод обробляє GET-запит «/bookmarks» для отримання списку закладок користувача. Він отримує ідентифікатор користувача з «req.user.id», знаходить користувача в базі даних за ідентифікатором та повертає список закладок користувача у відповідь, далі в фронтенд-частині відповідний компонент виконує рендеринг списку закладок при його виклику.

Цей клас використовує моделі «User» та «Role», які імпортуються з файлу «./models/User.js» та «./models/Role.js» відповідно. Він також використовує модулі «bcryptjs» для хешування паролів, «jsonwebtoken» для створення та перевірки JWT-токенів, «express-validator» для перевірки

валідності даних, та об'єкт «secret» з файлу «./config» для підпису JWT-токенів.

Далі трохи детальніше про схеми ролей які використовуються :

Файл «Role.js» у папці «models» містить модель для ролей користувача. Він використовує модуль «mongoose» для визначення схеми та моделі даних.

Схема «Role» визначає поле «value» типу «String», яке є унікальним і має значення "USER" за замовчуванням. Це поле використовується для представлення ролі користувача.

Шлях «model(Role)» створює модель з назвою "Role", яка базується на схемі «Role» і експортується для використання в інших частинах програми.

Файл «User.js» у папці «models» містить модель для користувача. Він також використовує модуль «mongoose» для визначення схеми та моделі даних.

Схема «User» визначає поля:

– «username» - типу «String», яке є унікальним і обов'язковим для заповнення. Це поле представляє ім'я користувача.

– «password» - типу «String», яке є обов'язковим для заповнення. Це поле представляє пароль користувача.

– «roles» - масив строкових значень, що посилаються на модель "Role". Він використовується для збереження ролей, пов'язаних з користувачем.

– «bookmarks» - масив строкових значень, що представляють закладки користувача.

Шлях «model(User)» створює модель з назвою "User", яка базується на схемі «User» і експортується для використання в інших частинах програми.

Обидва файли використовуються для взаємодії з базою даних (MongoDB) та забезпечують валідацію та збереження даних про ролі користувачів.

Файл «authRouter.js» містить визначення маршрутів для різних запитів, які відправляються до «authController» (Додаток В):

– «/registration»: POST-маршрут, який викликає метод «registration» у «authController» для реєстрації нового користувача. Він також містить валідацію вхідних даних (перевірка наявності імені та пароля).

– «/login»: POST-маршрут, який викликає метод «login» у «authController» для авторизації користувача.

– «/users»: GET-маршрут, який викликає метод «getUsers» у «authController» для отримання списку користувачів. Використовує «roleMiddlewaree» для перевірки ролі користувача.

– «/add-bookmark»: POST-маршрут, який викликає метод «addBookmark» у «authController» для додавання закладки. Використовує «authMiddlewaree» для перевірки автентифікації користувача.

– «/bookmarks/:bookmarkId»: DELETE-маршрут, який викликає метод «deleteBookmark» у «authController» для видалення закладки. Використовує «authMiddlewaree» для перевірки автентифікації користувача.

– «/bookmarks»: GET-маршрут, який викликає метод «getBookmarks» у «authController» для отримання списку закладок користувача. Використовує «authMiddlewaree» для перевірки автентифікації користувача.

Файл «config.js» містить секретний ключ («secret»), який використовується для підпису та перевірки JWT-токенів (Додаток В).

Наступні файли « authMiddlewaree.js» та «roleMiddleware.js» виконують функцію проміжного програмного забезпечення (middleware) для автентифікації користувача « authMiddlewaree.js» та перевірки чи має користувач необхідні ролі доступу до захищених маршрутів «roleMiddleware.js» відповідно.

Файл «authMiddleware.js» виконує функцію проміжного програмного забезпечення (middleware) для автентифікації користувача. Перевіряє наявність і валідність токена авторизації, що передається через заголовок «Authorization». Ці файли перевіряють чи користувач має дійсний токен авторизації перед доступом до захищених маршрутів (Додаток В).

Функція middleware має параметри «req», «res» і «next», які представляють об'єкт запиту, об'єкт відповіді і функцію, яка викликає наступний middleware або обробляє запит. У першому рядку перевіряється метод запиту. Якщо це запит типу "OPTIONS", то викликається функція «next()», яка передає управління наступному middleware. Далі виконується спроба отримати токен авторизації з заголовка «Authorization». Якщо токен не знайдений, повертається відповідь зі статусом 403 і повідомленням про помилку. Якщо токен знайдений, він перевіряється за допомогою функції «jwt.verify()» з використанням секретного ключа «secret». Якщо перевірка успішна, розкодовані дані зберігаються у властивості «req.user». У кінці викликається функція «next()», щоб передати управління наступному middleware або обробнику запиту.

Файл «roleMiddleware.js» також виконує функцію проміжного програмного забезпечення (middleware). Він перевіряє, чи користувач має необхідні ролі для доступу до певних захищених маршрутів. Цей файл існує для забезпечення авторизації заснованої на ролях користувача. (Додаток В)

Функція middleware має параметри «roles», які вказують на необхідні ролі для доступу, а також «req», «res» і «next». Як і у попередньому файлі, перевіряється метод запиту. Якщо це запит типу "OPTIONS", то викликається функція «next()». Далі виконується спроба отримати токен авторизації з заголовка «Authorization». Якщо токен не знайдений, повертається відповідь зі статусом 403 і повідомленням про помилку. Після отримання токена він перевіряється за допомогою функції «jwt.verify()» з використанням секретного ключа «secret». Розкодовані дані містять властивість «roles», яка представляє ролі користувача. Далі перевіряється, чи користувач має хоча б одну з необхідних ролей. Якщо немає відповідної ролі, повертається відповідь зі статусом 403 і повідомленням про помилку доступу. У кінці викликається функція «next()», щоб передати управління наступному middleware або обробнику запиту.

3.4 Інструкція користувачу

При первинному запуску додатку «Military-енциклопедія» користувачу відобразиться головна сторінка навігації по розділам додатку (рис. 3.1).

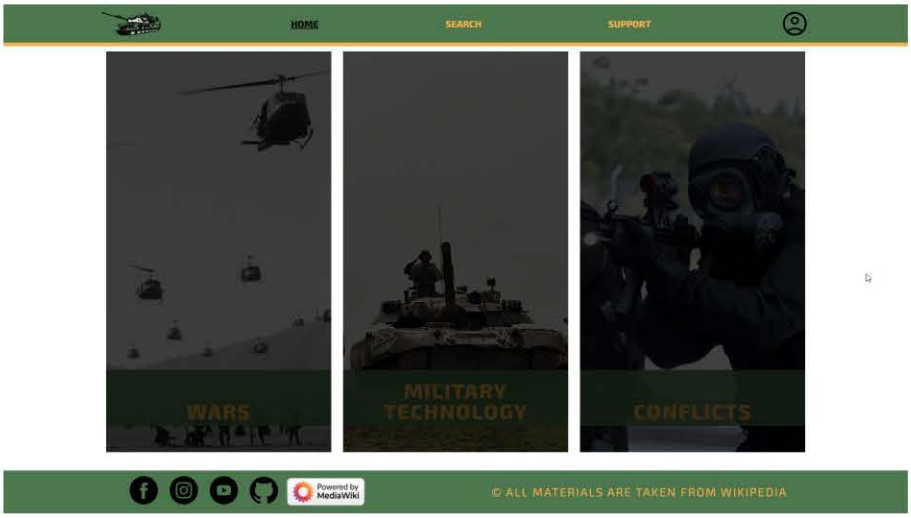

Рисунок 3.1 – Вигляд додатку при першому запуску

Розглянемо головне навігаційне меню додатку яке знаходиться у верхній частині екрану та має відповідні позначення в вигляді іконок та написів. При кліку на емблему сайту в верхньому лівому куті екрану користувачу буде відкрито так зване меню "Info" котре містить в собі опис призначення додатку та для кого його створено (рис.3.2).

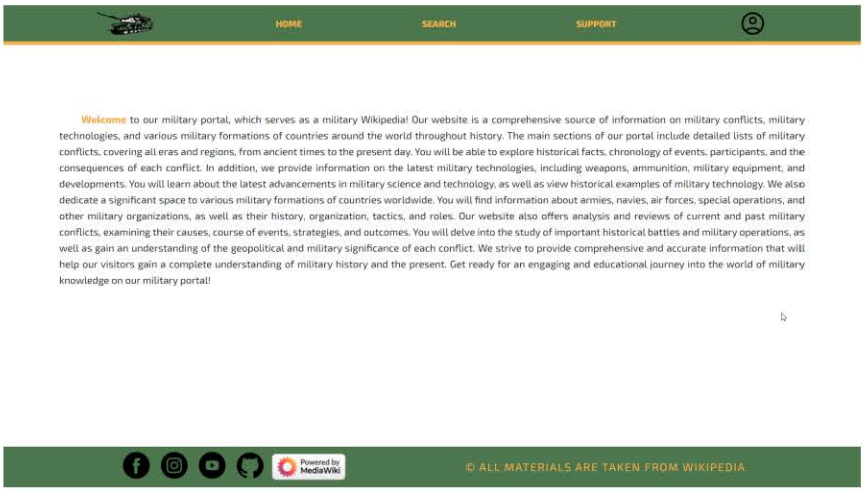

Рисунок 3.2 – Вигляд вкладки "Info"

Наступна вкладка "Home" при кліку повертає на головну навігаційну сторінку по відповідним розділам додатку із картками які мають підпис з назвами що відповідають за відповідний розділ енциклопедії "Wars", "Military Technology" та "Conflicts" (рис.3.1).

При виборі розділу "Wars" який містить в собі список війн до 1000 р.н.е. до списку воєн сучасності, список війн по регіонам, а також список війн по типу походження (рис.3.3).

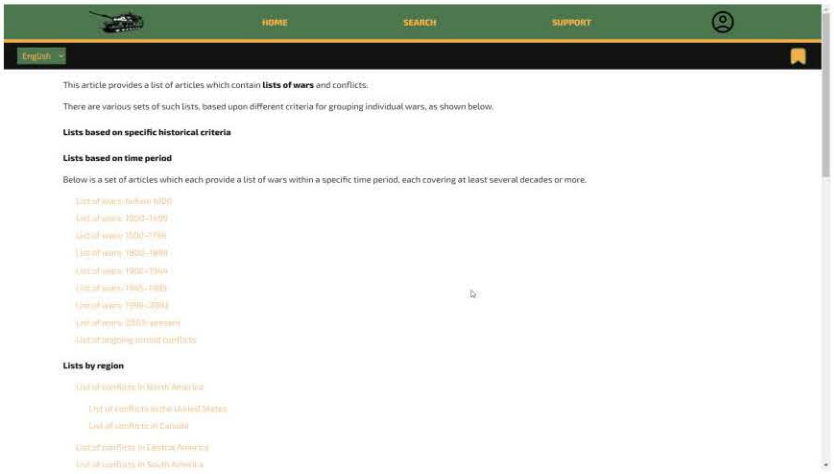

Рисунок 3.3 – Вигляд екрану при виборі "Wars"

При виборі розділу "Military Technology" який містить в собі список збройних сил за країнами , включаючи основні види сухопутних, військово-морських і повітряних сил (рис.3.4).

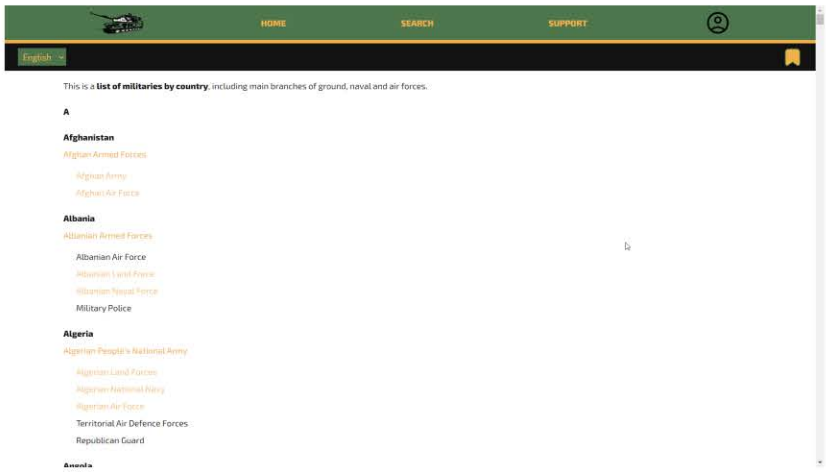

Рисунок 3.4 – Вигляд екрану при виборі " Military Technology "

При виборі розділу "Conflicts", який містить в собі список поточних збройних конфліктів визначає сучасні конфлікти та кількість загиблих, пов'язаних з кожним конфліктом (рис.3.5).

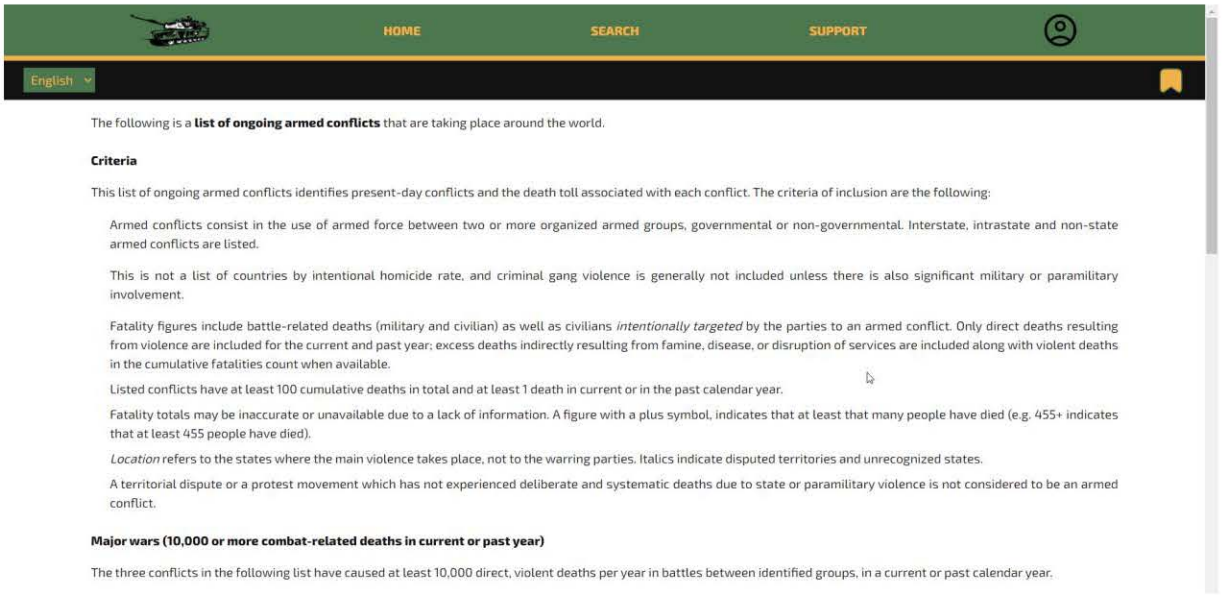

Рисунок 3.5 – Вигляд екрану при виборі "Conflicts"

Наступна вкладка "Search" при кліку здійснюється переадресація на сторінку пошку за ключовими словами. На екрані даної вкладки міститься поле для введення тексту та кнопка "Search" (рис.3.6). Введемо будь-яке ключове слово та натиснемо кнопку "Search".Після чого на екрані буде відображено відповідний до запиту результат (рис.3.7).

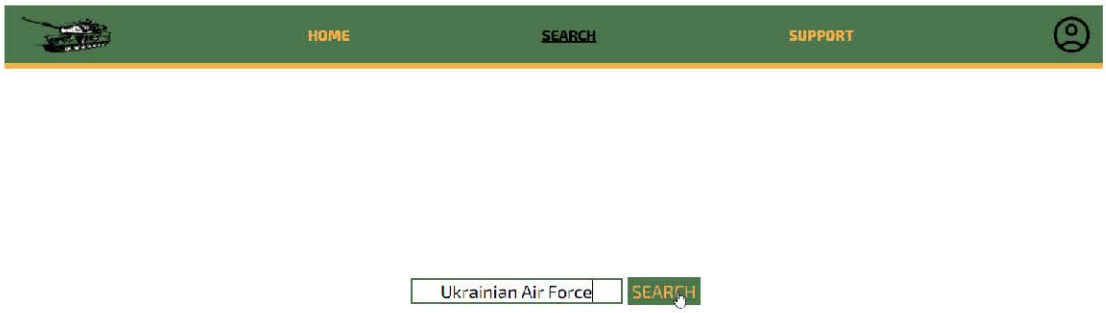

Рисунок 3.6 – Вигляд екрану пошуку

Далі демонстрація роботи пошукової системи. На якій можна побачити що в результаті роботи механізму пошуку відображено відповідну сторінку із вмістом (рис.3.7).

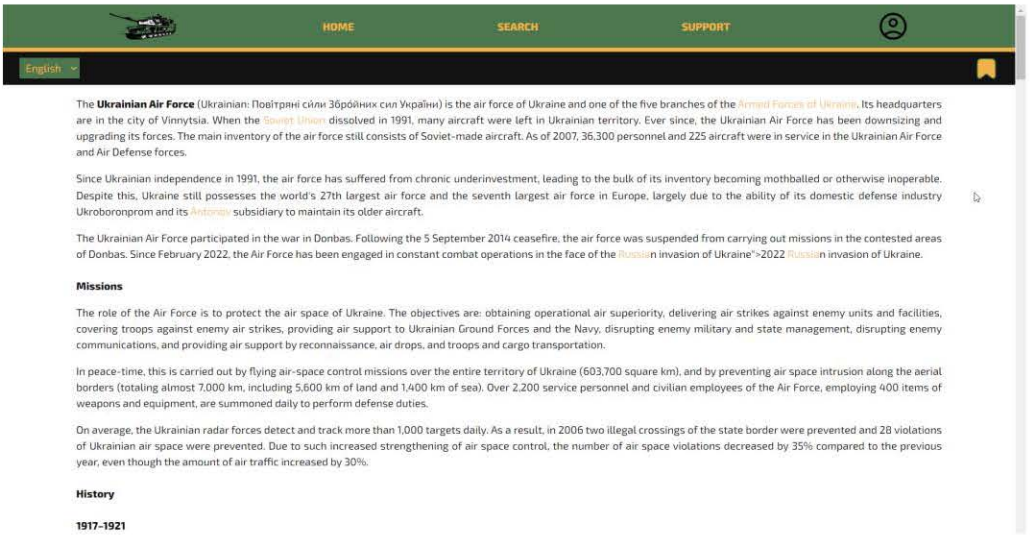

Рисунок 3.7 – Результат пошуку

Наступна вкладка "Support" при кліку здійснюється переадресація на сторінку підтримки користувача, тут знаходяться посилання на соціальні мережі проекту, пошта, та discord-чат проєкту за допомогою яких користувач може зв’язатися з тех-підтримкою на випадок пропозицій або зауважень щодо проєкту. Також відповіді на загальні питання котрі можуть виникнути у користувача (рис.3.8).

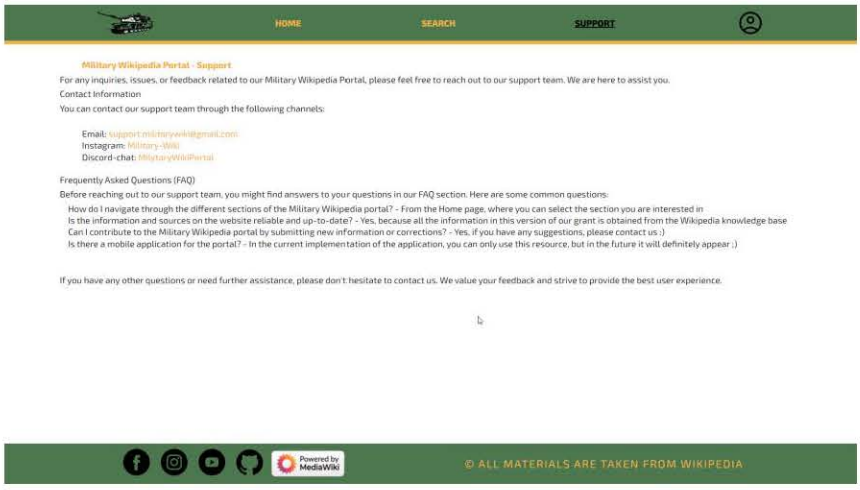

Рисунок 3.8 – Вигляд сторінки пітримки

Розглянемо головне меню користувача. Після кліку на відповідну кнопку в верхній правій частині екрану на кнопку із зображенням користувача. На даній сторінці відображено відповідний функціонал який надає можливість реєстрації та авторизації користувача. Зареєструємо нового користувача (рис.3.9).

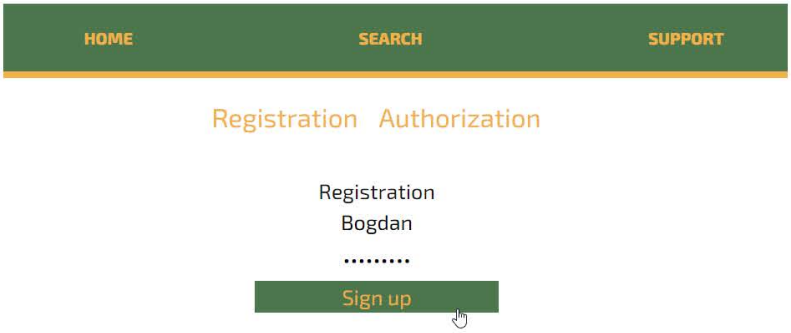

Рисунок 3.9 – Вигляд сторінки реєстрації та реєстрація

Після введення логіну та паролю та кліку по кнопці реєстрації "Sign up", користувача автоматично переадресує на сторінку "Home" головну сторінку навігації по розділам додатку.(рис. 3.1) Тепер після реєстрації проведемо авторизацію користувача в додатку за вказаними данними логіну та паролю.(рис.3.10)

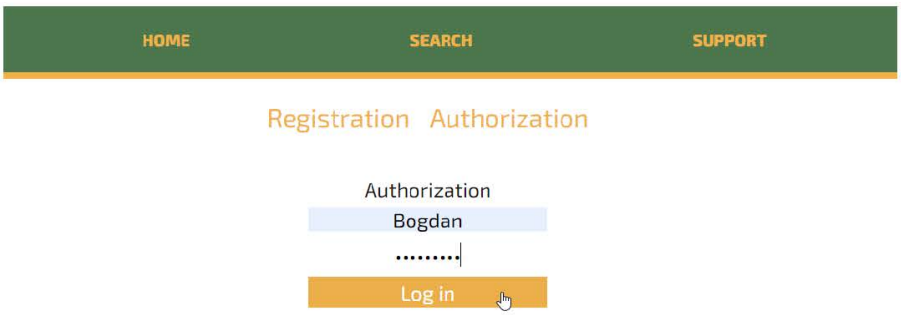

Рисунок 3.10 – Вигляд сторінки авторизації та авторизація

У разі вдалої авторизації користувача переадресує на сторінку "Home" головну сторінку навігації по розділам додатку (рис. 3.1).Далі перейдемо на

будь яку сторінку із статтею та додамо неї собі в збережені, за допомогою кнопки в верхньому правому куті. Додамо більше одної статті для демонстрації (рис.3.11).

Рисунок 3.11 – Випадкова стаття та додавання до вибраних

Тепер перейдемо в особистий кабінет та переглянемо список збережених статей. На даному рисунку можна побачити п’ять різних збережених сторінок на мілітарі тематику (рис.3.12)

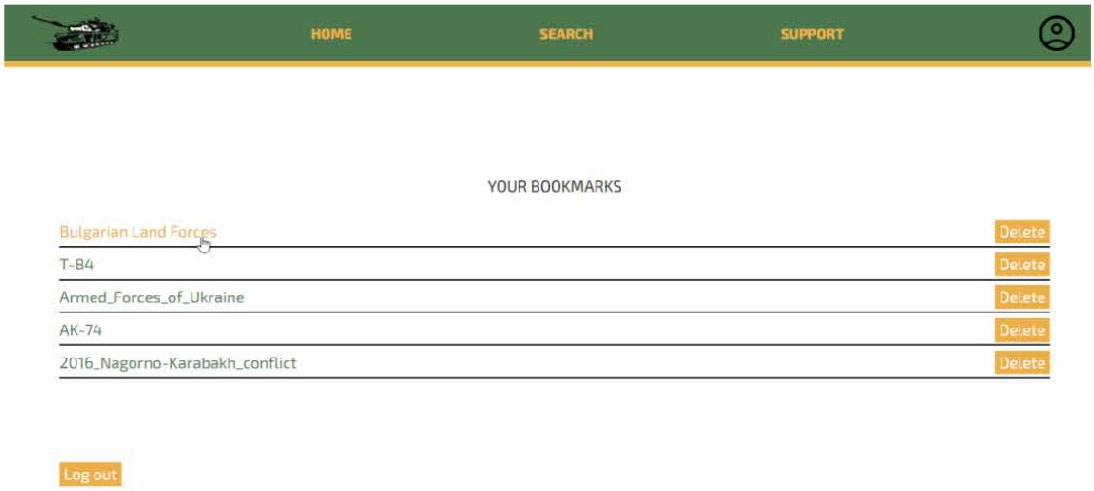

Рисунок 3.12 – Екран списку збережених статей користувача

З даного списку можна обрати будь-яку статтю та при кліку перейти по ній, а також за допомогою кнопки "Delete" можна видалити будь-яку.(рис.3.13)

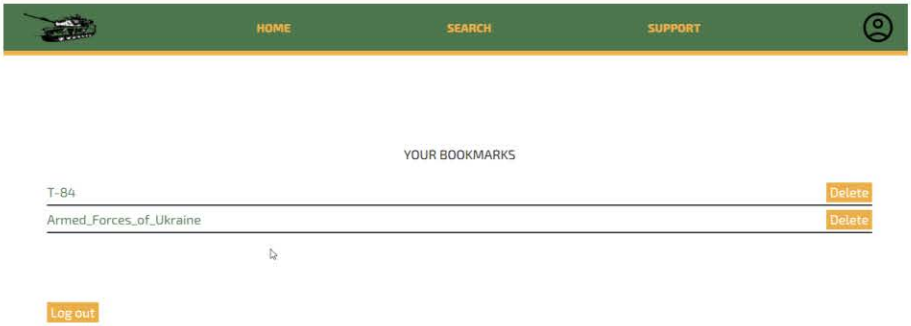

Рисунок 3.13 – Екран списку збережених статей користувача після виделення

Також додатково перейдемо на довільну статтю для демонстрації роботи кнопки "Прокрутити вгору" (рис.3.14).

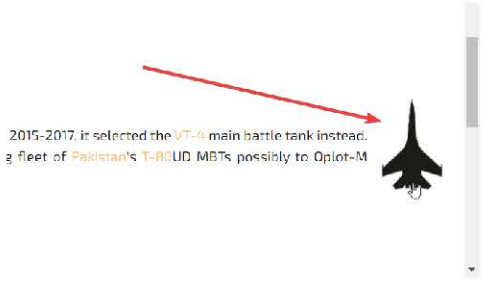

Рисунок 3.14 – Кнопка "Прокрутити вгору"

Наведемо курсор миші на дану кнопку, спрацює анімація, після кліку на дану кнопку та сторінка автоматично прокрутиться вгору (рис.3.15).

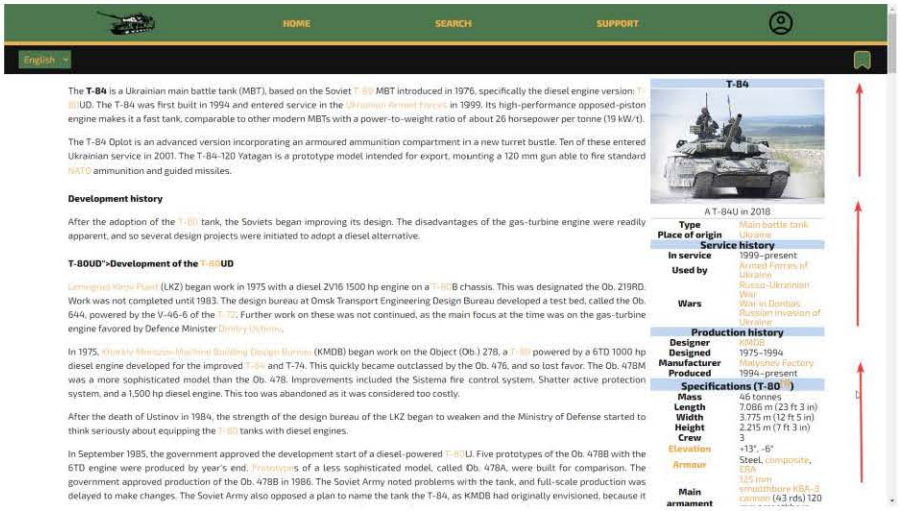

Рисунок 3.15 – Демонстрація кнопки "Прокрутити вгору"

Далі прокрутимо в самий низ сторінку і побачимо три заголовки функціональних блоків на фоні полоси зеленого кольору на всю ширину екрану " Show tables", "Show references" та "Images" (рис.3.16).

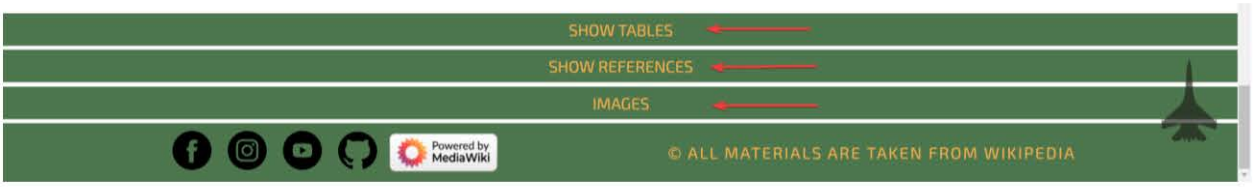

Рисунок 3.16 – Вигляд кнопок показати або приховати вміст

Дані заголовки є кнопками при кліку на які буде відображено відповідний вміст:

– Копка " Show tables" показано всі таблиці і посилання які відносяться до теми статті (рис.3.17).

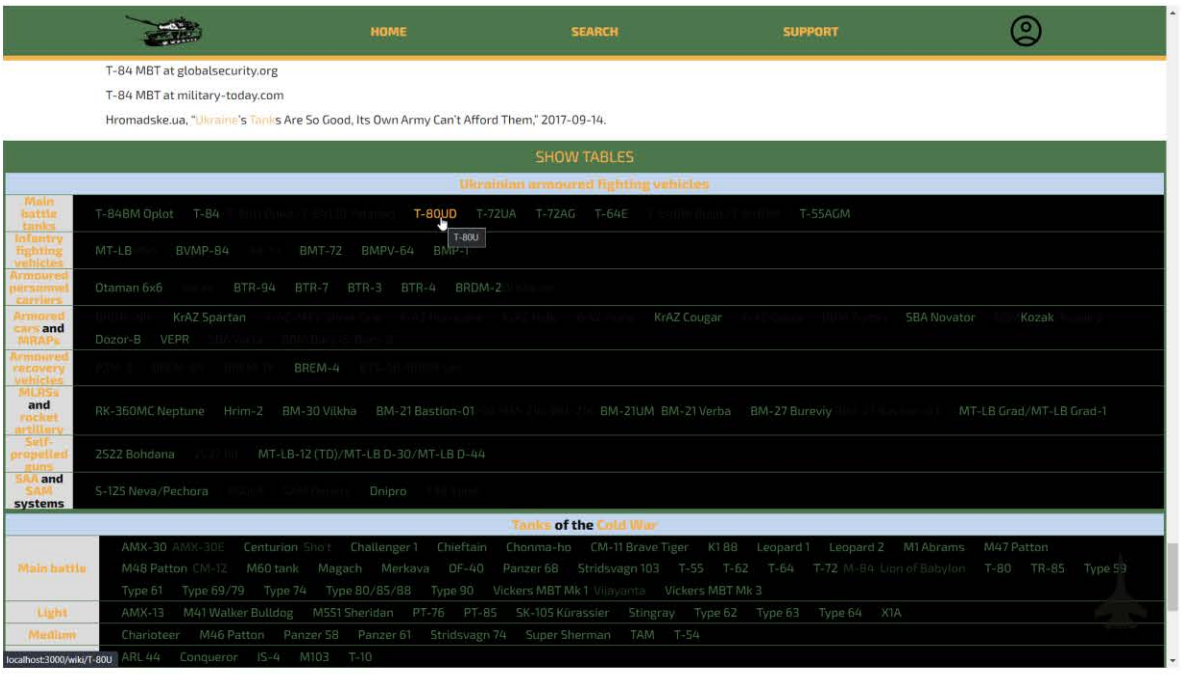

Рисунок 3.17 – Демонстрація вмісту екрану "Show tables"

Копка "Show references" показано всі посилання на зовнішні джерела які відносяться до теми статті (рис.3.18).

Рисунок 3.18 – Демонстрація вмісту екрану "Show references"

Копка "Images" показано всі зображення та схеми або інший медіа контент який відносяться до теми статті (рис.3.19).

Рисунок 3.19 – Демонстрація вмісту екрану " Images "

3.5 Висновки до третього розділу

У процесі реалізації дизайну було використано ряд технологій і інструментів. Зокрема, для розробки клієнтської частини додатку була використана бібліотека React JS, яка базується на об'єктно-орієнтованій мові програмування JavaScript. Для створення серверної частини додатку було використано фреймворк Express JS, що побудований на базі програмної платформи Node JS. Для зберігання та керування базами даних була використана документо-орієнтована система MongoDB. Це дозволило ефективно зберігати та отримувати інформацію про користувачів, що використовують додаток. Для наповнення додатку контентом та реалізації концепції достовірності та правдивості інформації щодо мілітарних тематик було використано API Wikipedia.

Після успішного завершення роботи над фронтенд та бекенд-частинами додатку, можна зробити такі висновки.

1. Створена верстка веб-додатку відповідає вимогам технічного завдання. Вона виконує функції, передбачені в дизайні та специфікаціях, і забезпечує зручний та привабливий користувацький інтерфейс.

2. Розробка фронтенд-частини на базі макету дозволила нам передати відповідність зовнішнього вигляду та поведінки додатку по задуманому дизайні. Взаємодія з додатком на ранніх етапах розробки допомогла виявити можливі проблеми та покращення, що можуть бути внесені для поліпшення користувацького досвіду. Також створення фронтенд-частини дало основу, яка відображає зовнішній вигляд та навігацію, і дала змогу перейти до наступних етапів розробки, а саме інтеграція з бекендом.

3. У бекенд-частині було успішно реалізовано функціонал особистого кабінету користувача, що дозволяє реєструватись, авторизовуватись та працювати зі списком обраних статей. Це важлива складова додатку, яка дозволяє користувачам зберігати свої персоналізовані дані.

ВИСНОВКИ

Згідно технічного завдання кваліфікаційної роботи було розроблено веб-додаток «Military-енциклопедія» з використанням сучасних технологій, інстументів та засобів розробки веб-додатків, використання метаданих API Wikipedia.

За результатами першого розділу було проведено аналіз можливих засобів для розробки додатку який дав вибір використання комбінації JavaScript або TypeScript для фронтенду з використанням фреймворку React.js або Vue, та Node.js з фреймворком Express.js для бекенду. У випадку бази даних був вибір між реляційною базою даних MySQL, або нереляційною базою даних MongoDB.

Розробка додатку здійснювалась за допомогою MERN-стеку, що складається з ReactJS, Node.js, Express.js та MongoDB. Використання цих технологій та компонентів гарантувало ефективну, швидку та безпечну реалізацію бекенд-частини та фронтенд-частини додатку з можливістю реєстрації, авторизації, додавання обраних статей та отримання інформації з API Wikipedia.

У процесі розробки веб-додатку «Military-енциклопедія» перед створенням базової файлової архітектури, було ретельно розроблене технічне завдання, яке визначало функціональні вимоги до додатку, його основні функції та очікувані результати. Також було проведено розробку макету додатку, що дозволило візуалізувати його інтерфейс та взаємодію елементів.

Створення технічного завдання та макету передувало розробці самого додатку, і вони були важливими кроками для забезпечення правильної організації роботи, уникнення непередбачених проблем та досягнення високої якості результуючого продукту.

У процесі створення, розробку було зосереджено на створенні зручного та привабливого користувацького інтерфейсу, що відповідає задачам і цілям додатку «Military-енциклопедія». З метою забезпечення інтуїтивної

зрозумілості інтерфейсу, було використано бібліотеку React JS на базі об'єктно-орієнтованої мови програмування JavaScript для реалізації функіоналу закладеного в технічному завданні та згідно макету. Також для реалізації функціоналу користувачів було використано фреймворк Express JS на базі платформи Node JS та документо-орієнтовану систему керування базами даних MongoDB для реалізації функціоналу особистого кабінету користувача, а також додатково, для отримання інформації використовується API Wikipedia як джерело для наповнення додатку контентом та реалізації концепції достовірності та правдивості інформації щодо мілітарі тематики.

Результати реалізації фронтенд-частини та бекенд-частини додатку показують, що верстка відповідає вимогам технічного завдання, фронтенд-частина відображає зовнішній вигляд та навігацію згідно з задуманим дизайном, а бекенд-частина успішно реалізує функціонал особистого кабінету користувача. Використання API Wikipedia дозволило отримувати актуальну та загальноприйняту інформацію з великої бази знань, що міститься в енциклопедії. Розроблений веб-додаток забезпечує зручний та привабливий користувацький інтерфейс, відповідаючи задачам і цілям проекту.

Висновок кваліфікаційної роботи підкреслює успішне виконання поставлених завдань, реалізацію функціонального дизайну та інтуїтивно зрозумілого інтерфейсу. Розроблений веб-додаток «Military-енциклопедія» задовольняє потреби користувачів у зручному та доступному способі отримання інформації про військову сферу, сприяючи підвищенню їхньої обізнаності та розуміння важливих аспектів військової історії та сучасності.

В цілому кваліфікаційна робота була успішно виконана, а розроблений веб-додаток відповідає всім поставленим завданням. Він має високу актуальність та відповідає потребам користувачів у сфері військових досліджень та освіти. Додаток є цікавим і корисним, і зможе поліпшити доступ до перевіреної інформації та покращити розуміння військової справи. В результаті роботи були розроблені зручний та привабливий користувацький інтерфейс, використано сучасні технології, такі як React JS, Express JS та

MongoDB, що дозволяють забезпечити мобільність, оперативність та ефективність додатку. Загалом, розроблений веб-додаток є цікавим доповненням у сфері військових досліджень та освіти або просто задоволення для любителів мілітарі тематики.

У плані подальшого розвитку веб-додатку «Military-енциклопедія» передбачається відмова від отримання інформації з API Wikipedia і створення власної обширної бази даних. Ця база буде заповнюватись редакторами та модераторами, які будуть перевіряти та додавати актуальну інформацію. Цей підхід дозволить забезпечити більшу надійність та контроль над інформацією, а також швидкість оновлення даних. Крім того, є ідея про розширення аудиторії додатку шляхом включення розділу з актуальними мілітарі новинами в режимі реального часу. Це дозволить користувачам бути в курсі останніх подій у військовій сфері та розширить функціонал додатку. В подальшому планується збільшення масштабів додатку, а також створення окремого великого додатку, який буде представляти собою форум для спілкування користувачів «Military-енциклопедія». Цей форум дозволить обговорювати теми військової справи, ділитись досвідом та інформацією, що сприятиме активному спілкуванню та обміну знаннями.

Крім того, є рання ідея розробки власної "Military-Social" соціальної мережі, за аналогією з відомими платформами, такими як Twitter та Pinterest. Це дозволить створити спеціалізовану соціальну платформу для військових та військово-інтересованих осіб, де вони зможуть обмінюватись інформацією, взаємодіяти та знаходити спільноту зі спільними інтересами.

Загалом, планується значне розширення функціоналу та масштабів веб-додатку «Military-енциклопедія», що надасть користувачам більше можливостей для отримання інформації, спілкування та взаємодії в сфері військових знань та спільнот.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Roznovsky A. Choosing a Technology Stack for Web Application Development./ Alexander Roznovsky – URL: https://light-it.net/blog/choosing-a-technology-stack-for-web-application-development/

2. JavaScript. – URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

3. Joshi M. Angular vs React vs Vue: Core Differences./ Mohit Joshi. – URL: https://www.browserstack.com/guide/angular-vs-react-vs-vue.

4. TypeScript for JavaScript Programmers. – URL: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html.

5. What Is Python?. – URL: https://aws.amazon.com/what-is/python/?nc1=h_ls.

6. Singh Khatri V. Flask vs Django: Which Python Web Framework to Use in 2023?./ V. Singh Khatri, R. Johns. – 2023. – URL: https://hackr.io/blog/flask-vs-django.

7. Веведення в C#. – URL: https://metanit.com/sharp/tutorial/1.1.php.

8. What is ASP.NET?. – URL: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet.

9. What is Java?. – URL: https://www.ibm.com/topics/java.

10. Spring Framework. – URL: https://www.techtarget.com/searchapparchitecture /definition/Spring-Framework.

11. What is PHP?. – URL: https://www.php.net/manual/en/intro-whatis.php.

12. Meet Laravel. – URL: https://laravel.com/docs/10.x#meet-laravel.

13. Olawanle J. What is a Framework? Software Frameworks Definition./ Joel Olawanle. – 2022. – URL: https://www.freecodecamp.org/news/what-is-a-framework-software-frameworks-definition/.

14. What Is Oracle Database. – URL: https://www.oracletutorial.com/getting-started/what-is-oracle-database/.

15. What is MySQL?. – URL: https://dev.mysql.com/doc /refman/8.0/en/what-is-mysql.html.

16. What is PostgreSQL?. – URL: https://www.postgresql.org/about/.

17. Taylor D. What is MongoDB?./ David Taylor – URL: https://www.guru99.com/what-is-mongodb.html.

18. What is CouchDB?. – URL: https://www.ibm.com/topics/couchdb.

19. Introduction to Redis. – URL: https://redis.io/docs/about/.

20. Color Hex Color Codes. – URL: https://www.color-hex.com/.

21. 4 AMAZING WEBSITE COLOR SCHEMES 2023 + COLOR MOOD BOARD. – URL: https://hookagency.com/blog/website-color-schemes/.

22. Google Fonts.– URL: https://fonts.google.com/specimen /Exo+2?preview.text=THIS%20IS%20TEST%20TEXT%20&preview.text_type=custom.

23. What Is An API (Application Programming Interface)? – URL: https://aws.amazon.com/what-is/api/.

24. Wikimedia REST API. – URL: https://www.mediawiki.org/wiki/Wikimedia_REST_API.

25. BasuMallick C. What Is a Single-Page Application? Architecture, Benefits, and Challenges./ Chiradeep BasuMallick. – URL: https://www.spiceworks.com/tech/devops/articles/what-is-single-page-application/.

26. Deshpande C. The Best Guide to Know What Is React./ Chinmayee Deshpande. – URL: https://www.simplilearn.com /tutorials/reactjs-tutorial/what-is-reactjs.

27. Your First Component. – URL: https://react.dev/learn/your-first-component.

28. Writing Markup with JSX. – URL: https://react.dev/learn/writing-markup-with-jsx.

29. Mittal A. How to use React Router v6 in React apps./ Aman Mittal. – URL: https://blog.logrocket.com/react-router-v6-guide/#what-react-router.

30. Liyana Gunawardhana P. What's New in React Router 6?./ Piumi Liyana Gunawardhana. – URL: https://enlear.academy/whats-new-in-react-router-6-e34e451e5285.

31. Jasraj J. ES6 Promises./ Jasraj Jasraj. – URL: https://www.geeksforgeeks.org/es6-promises/.

32. What is Axios?. – URL: https://axios-http.com/docs/intro.

33. Про Node.js. – URL: https://nodejs.org/uk/about.

34. Copes F. The V8 JavaScript Engine./ F. Copes, S. Smfoote, M. Borins. – URL: https://nodejs.dev/en/learn/the-v8-javascript-engine/.

35. Express web framework (Node.js/JavaScript). – URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/ Express_Nodejs.

36. Complete Guide to Express Middleware. – URL: https://reflectoring.io/express-middleware/#:~:text=Express%20middleware%20refers%20to%20a,processing%20to%20another%20middleware%20function..

37. Bcrypt. – URL: https://en.wikipedia.org/wiki/Bcrypt.

38. Dhruw D. ORM and ODM — A Brief Introduction / Deepanshu Dhruw. – URL: https://medium.com/spidernitt/orm-and-odm-a-brief-introduction-369046ec57eb#:~:text=What%20is%20ODM%3F,graph%20database%2C%20etc.

39. Hall J. Getting Started with MongoDB & Mongoose / Jesse Hall. – URL: https://www.mongodb.com/developer/languages/javascript/getting-started-with-mongodb-and-mongoose/.

40. Wikimedia REST API. – URL: https://www.mediawiki.org/wiki/Wikimedia_REST_API#:~:text=The%20REST%20API%20along%20with,org%2Fapi%2Frest_v1%2F..

41. Code editing. Redefined. – URL: https://code.visualstudio.com/.

42. What is Git. – URL: https://www.atlassian.com/git/tutorials/what-is-git#:~:text=Git%20is%20a%20mature%2C%20actively,as%20well%20as%20open%20source..

43. What Is GitHub? A Beginner's Introduction to GitHub. – 2022. – URL: https://kinsta.com/knowledgebase/what-is-github/.

ДОДАТОК А

Таблиця А.1

Таблиця порівняння мов програмування для фронтенду та бекенду

Мова програмування	**Об'єктно-орієнтованість**	**Синтаксис**	**Простота вивчення**	**Розширені можливості**	**Підтримка фреймворків**
JavaScript	Частково	Гнучкий	Висока	Широкі	React, Node.js, Angular, Vue.js
TypeScript	Так	Розширений	Помірна	Сильна типізація	Angular, React, Node.js
Python	Так	Читабельний	Висока	Багато бібліотек	Django, Flask
C#	Так	Схожий на C++	Середня	Міцна підтримка .NET	ASP.NET, .NET Core
Java	Так	Схожий на C++	Помірна	Багато фреймворків	Spring, JavaServer Faces, Hibernate
PHP	Частково	Простий	Висока	Розширені можливості	Laravel, Symfony

Таблиця А.2.

Порівняльна таблиця комбінацій мов програмування для фронтенду та бекенду

Комбінація	**Мова програмування для фронтенду**	**Мова програмування для бекенду**
JavaScript + Node.js	JavaScript	JavaScript
Python + Django	Python	Python
Ruby + Ruby on Rails	Ruby	Ruby
PHP + Laravel	PHP	PHP
Java + Spring	Java	Java
C# + ASP.NET	C#	C#

Таблиця А.3.

Порівняльна таблиця за об’єктно-орієнтованими можливостями та функціональними можливостями

База даних	**Об’єктно-орієнтовані можливості**	**Функціональні можливості**
Oracle	Обмежена підтримка	Широкий набір функцій
MySQL	Обмежена підтримка	Стандартні можливості
PostgreSQL	Повна підтримка	Розширені можливості
MongoDB	Повна підтримка	Гнучкість в зміні схеми
CouchDB	Повна підтримка	Легкість синхронізації
Redis	Відсутність	Висока швидкодія

ДОДАТОК Б

Рисунки для візуалізації макету дизайну додатку.

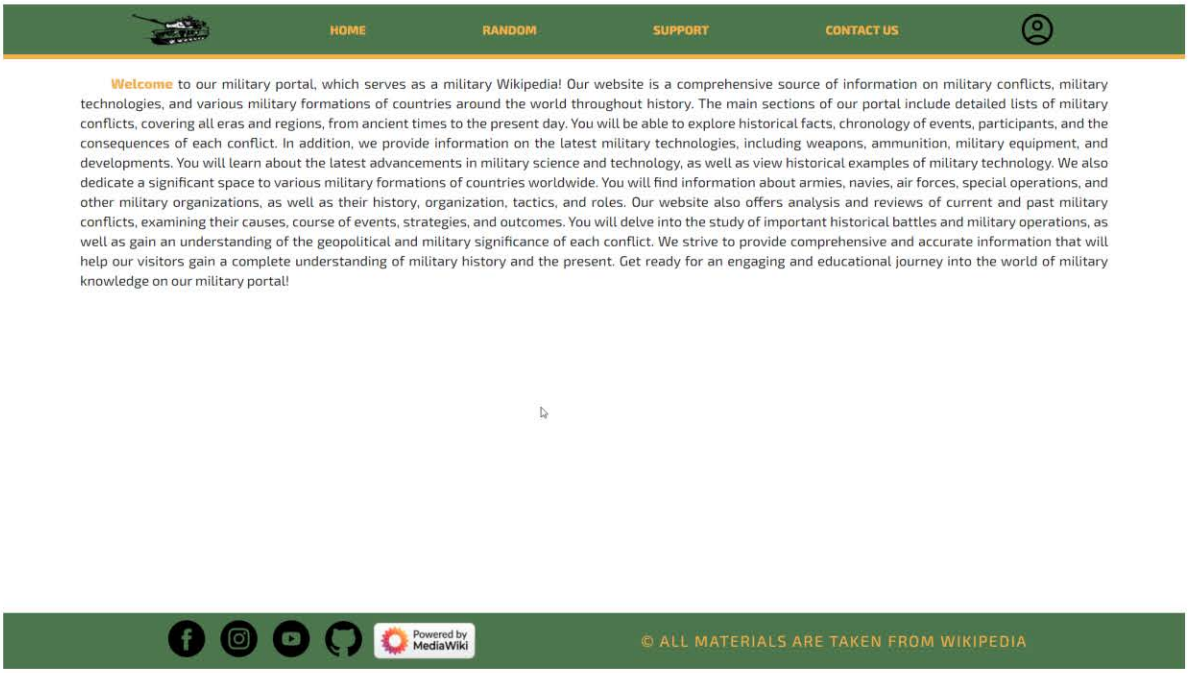

Рисунок Б.1 – Загальний вигляд додатку та меню Info

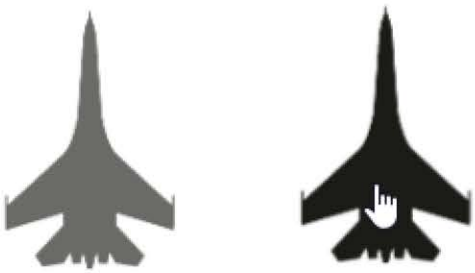

Рисунок Б.2 – Вигляд кнопки прокрутки

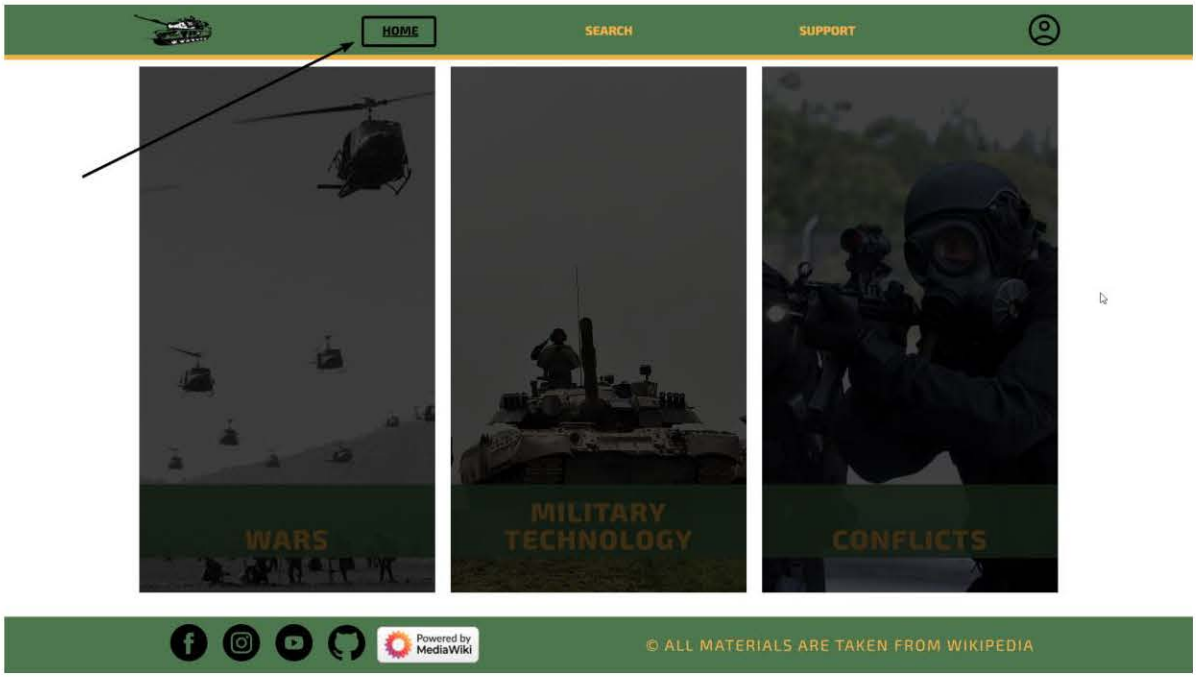

Рисунок Б.3 – Вигляд екрану меню HOME

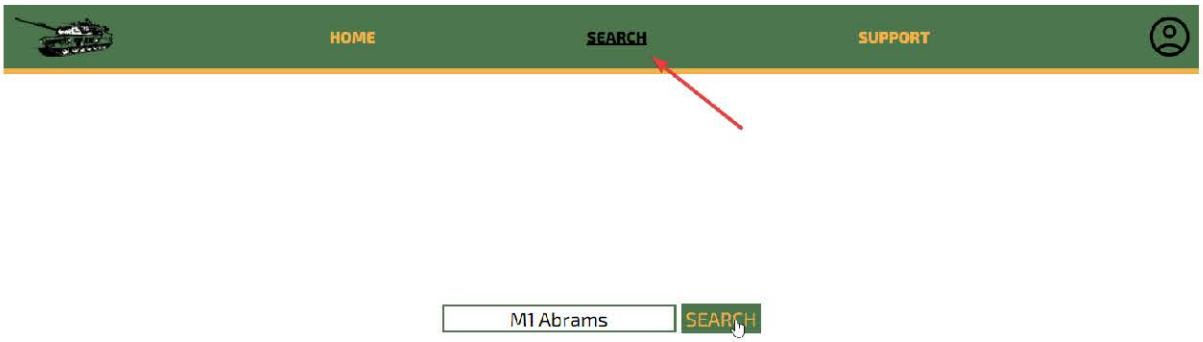

Рисунок Б.4 – Вигляд екрану меню SEARCH

Рисунок Б.5 – Вигляд екрану меню користувача та форма реєстрації

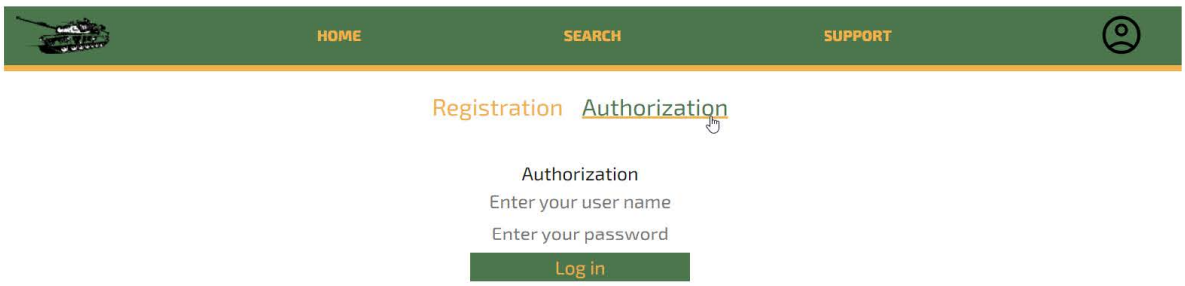

Рисунок Б.6 – Вигляд екрану меню користувача та форма авторизації

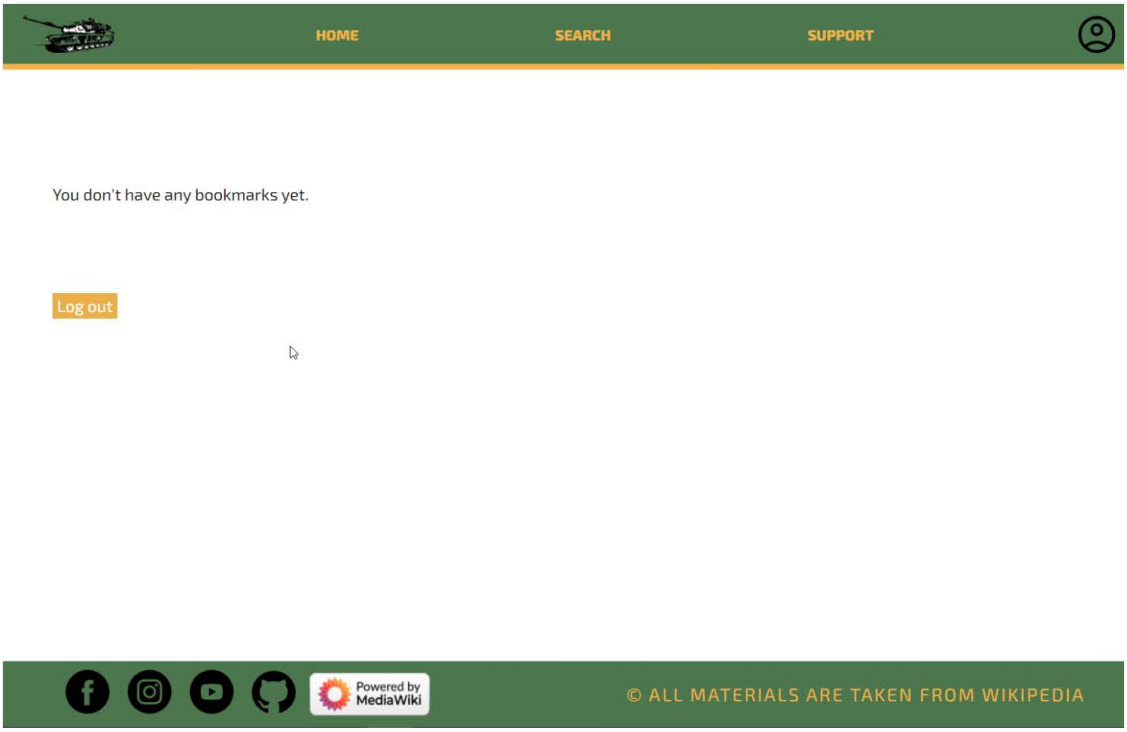

Рисунок Б.7 – Вигляд екрану закладок

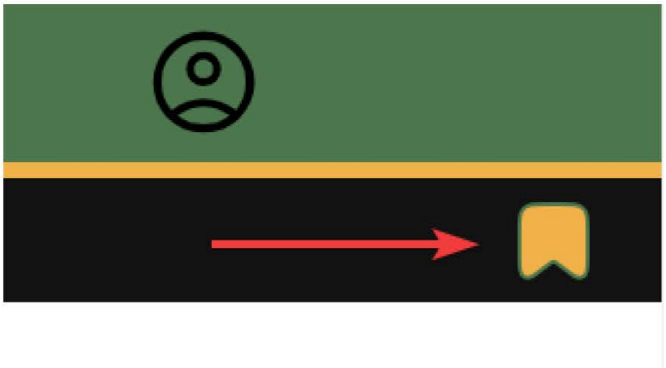

Рисунок Б.8 – Вигляд функціональної кнопки для додавання закладки

ДОДАТОК В

Лістинги фронтенд та бекенд-частин додатку.

В.1 Лістинг файлу "index.js"

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { BrowserRouter } from 'react-router-dom';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
<React.StrictMode>
<BrowserRouter>
<App />
</BrowserRouter>
</React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

В.2 Лістинг файлу "app.js"

```
import React from 'react';
import './App.css';
import { Routes, Router, Route, Link } from 'react-router-dom';

import { Layout } from './components/Layout';

import { WarsCard, MilitaryTechnologyCard, MilitaryConflictsCard
} from "./pages/main-page/main/cards/CardsComponents";
import { InfoPage } from './pages/main-page/header/Info';
import Home from "./pages/main-page/header/Home";
import Random from "./pages/main-page/header/Random";
import { SupportPage } from "./pages/main-page/header/Support";
import ContactUs from "./pages/main-page/header/ContactUs";
import { Article } from './API/Article';
import AppWiki from './API/AppWiki';
import { FormRegLog } from './pages/login-page/FormRegLog';
import { RegistrationForm } from './pages/login-page/FormRegLog';
import { LoginForm } from './pages/login-page/FormRegLog';
import { Dashboard } from './pages/login-page/FormRegLog';


function App() {

    return (
        <>
            <Routes>
```

```
26.                     <Route path='/' element={<Layout />}>
27.                         <Route index path='info' element={<InfoPage
/>} />
28.
29.                         <Route path='/' element={<Home />} >
30.                             <Route path="list_of_war"
element={<WarsCard />} />
31.                             <Route path="MilitaryTechnologyCard"
element={<MilitaryTechnologyCard />} />
32.                             <Route path="MilitaryConflictsCard"
element={<MilitaryConflictsCard />} />
33.                         </Route>
34.                         <Route path=':targetPage' element={<AppWiki
/>} />
35.
36.
37.                         <Route path='/*' element={<Article />} />
38.                         <Route path='random' element={<Random />} />
39.                         <Route path='support' element={<SupportPage
/>} />
40.                         <Route path='contact' element={<ContactUs />}
/>
41.
42.                         <Route path='user/*' element={<FormRegLog />}
>
43.                             <Route index element={<FormRegLog />} />
44.                             <Route path='registration'
element={<RegistrationForm />} />
45.                             <Route path='login' element={<LoginForm
/>} />
46.                             <Route path='dashboard'
element={<Dashboard />} />
47.                         </Route>
48.
49.                     </Route>
50.                 </Routes >
51.
52.             </>
53.
54.         );
55.     }
56.
57.     export default App;
```

В.3 Лістинг файлу "layout.jsx" компоненту "layout "

```
1.      import { React, useState } from "react";
2.      import { NavLink, Outlet, Route, Routes } from "react-router-
dom";
3.
4.      import BurgerMenu from "../pages/main-page/header/Burger-
btn/burgerMenu";
5.      import { Info } from "../pages/main-page/header/Info";
6.      import Footer from '../pages/main-page/footer/footer';
7.      import GoTopButton from '../pages/main-page/goTop/goTop';
8.
9.      import "../pages/main-page/header/Burger-btn/burger-btn.css";
10.     import s from "../pages/main-page/header/h.module.css";
11.     import "../pages/main-page/main/m.module.css"
12.
13.     const Layout = () => {
```

```
14.         const [menuActive, setMenuActive] = useState(false);
15.         return (
16.             <div className='wrapper'>
17.                 <header className={s.header}>
18.                     <div id="#" className={s.header__container}>
19.
20.                         <NavLink to="/info"
className={s.header__logo}>
21.                             <Info />
22.                         </NavLink>
23.
24.                         <div className={`${s.header__menu}
${s.menu}`}>
25.                             < nav>
26.                                 <div className="burger-btn"
onClick={() => setMenuActive(!menuActive)}>
27.                                     <span />
28.                                 </div >
29.                                 <ul className={s.menu__list}>
30.
31.                                     <NavLink className={s.menu__link}
to="/">Home</NavLink>
32.                                     <NavLink className={s.menu__link}
to="/random" >Search</NavLink>
33.                                     <NavLink className={s.menu__link}
to="/support" >Support</NavLink>
34.
35.                                 </ul>
36.                             </nav>
37.                             <BurgerMenu active={menuActive}
setMenuActive={setMenuActive} />
38.                         </div >
39.                         <div className={`${s.header__user}
${s.user}`}>
40.                             <NavLink to="/user"
className={s.header__user}>
41.                                 <a href="" className={s.user__icon}>
42.                                     <img src="../user-icon.svg"
alt="log in" title="LOG IN" placeholder="USER" />
43.                                 </a>
44.                             </NavLink>
45.                         </div>
46.                     </div >
47.                 </header >
48.
49.                 <main >
50.                     <Outlet />
51.                 </main>
52.                 <Footer />
53.                 <GoTopButton />
54.             </div>
55.         )
56.     }
57.     export { Layout }
```

В.4 Лістинг файлу "info.jsx"

```
1.     import React from "react";
2.     import '../main/cards/cardClickEvent/parseStyle.css'
3.
4.
5.     const Info = () => {
```

```
        return (
            <>
                <img src="../logo_12.png" placeholder="INFO" />
            </>
        )
    }

    const InfoPage = () => {
        return (
            <div className="info-page">

                <p>
                    <strong style={{ color: "#f1b24a", paddingLeft:
"40px" }}>Welcome</strong> to our military portal, which serves as a military
Wikipedia! Our website is a comprehensive source of information on military
conflicts, military technologies, and various military formations of
countries around the world throughout history.

                    The main sections of our portal include detailed
lists of military conflicts, covering all eras and regions, from ancient
times to the present day. You will be able to explore historical facts,
chronology of events, participants, and the consequences of each conflict.

                    In addition, we provide information on the latest
military technologies, including weapons, ammunition, military equipment, and
developments. You will learn about the latest advancements in military
science and technology, as well as view historical examples of military
technology.

                    We also dedicate a significant space to various
military formations of countries worldwide. You will find information about
armies, navies, air forces, special operations, and other military
organizations, as well as their history, organization, tactics, and roles.

                    Our website also offers analysis and reviews of
current and past military conflicts, examining their causes, course of
events, strategies, and outcomes. You will delve into the study of important
historical battles and military operations, as well as gain an understanding
of the geopolitical and military significance of each conflict.

                    We strive to provide comprehensive and accurate
information that will help our visitors gain a complete understanding of
military history and the present. Get ready for an engaging and educational
journey into the world of military knowledge on our military portal!
                </p>

            </div>
        )
    }

    export { Info, InfoPage };
```

B.5 Лістинг файлу "home.jsx"

```
    import React from "react";
    import "./h.module.css";
    import { WarsCard, MilitaryTechnologyCard, MilitaryConflictsCard
} from "../main/cards/CardsComponents";
    import { NavLink, Outlet } from "react-router-dom";

    export default function Home() {
```

```
7.
8.          return (
9.              <>
10.                 <div className="main_section_one">
11.
12.                     <NavLink to="Lists_of_wars">{<WarsCard
/>}</NavLink>
13.                     <NavLink to="List_of_militaries_by_country"
>{<MilitaryTechnologyCard />}</NavLink>
14.                     <NavLink to="List_of_ongoing_armed_conflicts"
>{<MilitaryConflictsCard />}</NavLink>
15.                 </div>
16.                 <Outlet />
17.             </>
18.         );
19.     }
```

В.6 Лістинг файлу "cardscomponents.jsx"

```
1.      import React from "react";
2.
3.      import s from "../m.module.css";
4.
5.      const WarsCard = () => {
6.          return (
7.              <div >
8.                  {/* Дизайн та вміст картки "WARS" */}
9.                  {<section className={«${s.main__chapter}, ${s.chapter
- 1}»} >
10.
11.                     <div className={s.chapter_block}>
12.                         <div className={s.chapter__overlay}></div>
13.                         <img src="/img/home-
page/main/menu/war image1.jpg" alt="WARS" />
14.                         <h4>WARS</h4>
15.                     </div>
16.                 </section >}
17.             </div>
18.
19.         );
20.     };
21.
22.     const MilitaryTechnologyCard = () => {
23.         return (
24.             <div >
25.                 {/* Дизайн та вміст картки "Military-technology" */}
26.                 {<section className={«${s.main__chapter}, ${s.chapter
- 3}»}>
27.                     <div className={
28.                         s.chapter_block}>
29.                         <div className={s.chapter__overlay}></div>
30.                         <img src="/img/home-page/main/menu/desktop-
wallpaper-tanks-soldiers-russian-three-3-army-russian-military.jpg"
31.                             alt="MILITARY-EQUIPMENT" />
32.                         <h4>Military<br /> technology</h4>
33.                     </div >
34.                 </section >}
35.             </div>
36.         );
37.     };
38.
39.     const MilitaryConflictsCard = () => {
40.         return (
```

```
            <div >
                {/* Дизайн та вміст картки "Military-conflicts" */}
                {<section className={«${s.main__chapter}, ${s.chapter
- 2}»}>
                    < div className={
                        s.chapter_block}>
                        <div className={s.chapter__overlay}></div>
                        <img src="/img/home-
page/main/menu/mc_image2.jpg " alt="MILITARY-CONFLICTS" />
                        <h4>CONFLICTS</h4>
                    </div >
                </section >}
            </div>
        );
    };

    export { WarsCard, MilitaryTechnologyCard, MilitaryConflictsCard
};
```

В.6 Лістинг файлу "appwiki.jsx"

```
    import React, { useState, useEffect } from "react";
    import './parseStyle.css';
    import { TakeTables, Infobox, Navbox, Reflist } from
"./TakeTables";
    import { BrowserRouter as Router, Link, Route, useParams, Routes,
BrowserRouter, useNavigate } from "react-router-dom";
    import { AddPageButton } from "../components/AddPageButton";


    // Компонент для відображення тексту статті
    const WikiText = ({ extract, onLinkClick }) => {
        const handleLinkClick = (event, linkUrl) => {
            event.preventDefault();
            onLinkClick(linkUrl);
        };

        const replaceLinks = (html) => {
            const tempElement = document.createElement("div");
            tempElement.innerHTML = html;

            const links = tempElement.getElementsByTagName("a");
            for (let i = 0; i < links.length; i++) {
                const link = links[i];
                const linkUrl = link.getAttribute("href");
                link.addEventListener("click", (event) =>
handleLinkClick(event, linkUrl));
            }

            return tempElement.innerHTML;
        };

        return (
            <div className="text" dangerouslySetInnerHTML={{ __html:
replaceLinks(extract) }} />
        );
    };

    // Компонент для відображення зображень
    const WikiImages = ({ images }) => {

        return (
            <>
```

```
39.                 <div className="images">
40.                     {images && <div className="div-wrap-img"
dangerouslySetInnerHTML={{ __html: images }} />}
41.                 </div >
42.             </>
43.         );
44.
45.     };
46.
47.     // Компонент для вибору мови
48.     const LanguageSelect = ({ language, onLanguageChange }) => {
49.         return (
50.             <div className="language">
51.                 <select className="lalanguage-select" id="language-
select" value={language} onChange={onLanguageChange}>
52.                     <option value="en">English</option>
53.                     <option value="fr">French</option>
54.                     <option value="es">Spanish</option>
55.                     <option value="de">German</option>
56.                     <option value="it">Italian</option>
57.                 </select>
58.             </div>
59.         );
60.     };
61.
62.     const AppWiki = () => {
63.
64.         //UseParams для відображення активного посилання сторінки
65.         const { targetPage } = useParams();
66.         const navigate = useNavigate();
67.
68.         // СЕТ ТАЙМ-АУТ НА 1 СЕКУНДУ ДЛЯ ЗАВАНТАЖЕННЯ СТОРІНКИ
69.         const [isLoading, setIsLoading] = useState(true);
70.         useEffect(() => {
71.             const timeoutId = setTimeout(() => {
72.                 setIsLoading(false);
73.             }, 1000);
74.
75.             return () => {
76.                 clearTimeout(timeoutId);
77.             };
78.         }, []);
79.
80.         const [language, setLanguage] = useState("en"); // Стан для
вибраної мови
81.         const [wikiText, setWikiText] = useState(""); // Стан для
тексту статті з Wikipedia
82.         const [wikiImages, setWikiImages] = useState([]); // Стан для
зображень з Wikipedia
83.         const [pageData, setPageData] = useState(null); //Стан для
даних сторінки з Wikipedia
84.         const [isImagesBlock, setIsImagesBlock] = useState(true);//
Стан для механізму приховати/показати блок зображень
85.         const [isTableHidden, setIsTableHidden] = useState(true);//
Стан для механізму приховати/показати блок таблиць
86.         const [isReflistHidden, setIsReflistHidden] =
useState(true);// Стан для механізму приховати/показати блок посилань
87.         const [infoboxHTML, setInfoboxHTML] = useState("");// Стан
для механізму блоку додадкової-короткої інформації
88.         const [navboxHTML, setNavboxHTML] = useState("");//
89.         const [reflistHTML, setReflistHTML] = useState("");
90.
91.
```

```
92.         const show_hide = () => {
93.             setIsImagesBlock(!isImagesBlock);
94.         };
95.
96.         const handleToggle_T = () => {
97.             setIsTableHidden(!isTableHidden);
98.         };
99.
100.        const handleToggle_R = () => {
101.            setIsReflistHidden(!isReflistHidden);
102.        };
103.
104.
105.        const getFirstPageExtract = (jsonResponse) => {
106.            const pageId = Object.keys(jsonResponse.query.pages)[0];
107.            const pageContent =
jsonResponse.query.pages[pageId].extract;
108.            const pageLinks = jsonResponse.query.pages[pageId].links;
109.
110.            let linkHTML = pageLinks;
111.            let pageHTML = pageContent;
112.            for (let i = 0; i < pageLinks.length; i++) {
113.                const link = pageLinks[i];
114.                linkHTML = «<a
href="${link.title}">${link.title}</a>«;
115.                pageHTML = pageHTML.replace(new RegExp(link.title,
"g"), linkHTML);
116.            }
117.            return pageHTML;
118.        };
119.
120.
121.        const getPageImages = jsonResponse => {
122.            // ... код функції getPageImages ...
123.            const pages = jsonResponse.query.pages;
124.            const pageIds = Object.keys(pages);
125.            const firstPageId = pageIds.length ? pageIds[0] : null;
126.            const images = pages[firstPageId]?.images || [];
127.
128.            const imageFiles = images.filter(image => (
129.                image?.title?.toLowerCase().endsWith('.jpg') ||
130.                image?.title?.toLowerCase().endsWith('.jpeg') ||
131.                image?.title?.toLowerCase().endsWith('.png') ||
132.                image?.title?.toLowerCase().endsWith('.gif')
133.            ));
134.
135.            const languageCode = language === 'en' ? 'commons' :
language; // Отримання коду обраної мови
136.
137.            const imageElements = imageFiles.map(image => {
138.                // Вилучаємо префікси "File:,
Datei:,Fichier:,Archivo:" и розширення ".jpg"
139.                const titleWithoutPrefix =
image.title.replace(/^File:|^Datei:|^Fichier:|^Archivo:/, '');
140.                const titleWithoutExtension =
titleWithoutPrefix.substring(0, titleWithoutPrefix.lastIndexOf('.'));
141.                return (
142.                    »<span>
143.                    <img
src="https://${languageCode}.wikipedia.org/wiki/Special:FilePath/${encodeURIC
omponent(image.title)}" />
144.                    <p>${titleWithoutExtension}</p>
145.                </span>«
```

```
146.                 );
147.             })
148.             return imageElements.join(``);
149.         };
150.
151.
152.         const getPageData = (jsonResponse, language) => {
153.             // ... код функції getPageData ...
154.             const pageId = Object.keys(jsonResponse.query.pages)[0];
155.             const pageData = jsonResponse.query.pages[pageId];
156.
157.             const languageCode = language === 'en' ? 'commons' :
language === 'ru' ? 'ru' : language === 'zh' ? 'zh-cn' : language; //
Исправлено
158.             return {
159.                 title: pageData.title,
160.                 extract: pageData.extract,
161.                 images: pageData.images.map((image) => ({
162.                     title: image.title,
163.                     url: image.url,
164.                 })),
165.                 links: pageData.links.map((link) => ({
166.                     title: link.title,
167.                     url:
«https://${languageCode}.wikipedia.org/wiki/${encodeURIComponent(
168.                         link.title
169.                     )}»,
170.                 })),
171.             };
172.         };
173.
174.         //Вибір мови статті
175.         const handleLanguageChange = (event) => {
176.             const selectedLanguage = event.target.value
177.             setLanguage(selectedLanguage);
178.
179.         };
180.
181.
182.         useEffect(() => {
183.             const url = «https://en.wikipedia.org/w/api.php?» +
184.                 new URLSearchParams({
185.                     action: "parse",
186.                     origin: "*",
187.                     page: targetPage,
188.                     format: "json",
189.                     prop: "text"
190.                 });
191.
192.             fetch(url)
193.                 .then(response => response.json())
194.                 .then(data => {
195.                     const pageContent = data.parse.text["*"];
196.
197.                     // Пошук та видалення атрибутів стилів з таблиць
198.                     const tableRegex =
/<table[^>]*>[\s\S]*?<\/table>/gi;
199.                     let tableMatches = pageContent.match(tableRegex);
200.                     let tableHTML = tableMatches ?
tableMatches.join("") : "";
201.                     tableHTML = tableHTML.replace(/style="[^"]*"/g,
"");
202.
```

```
203.                    // Пошук div-елементів з класами "infobox vcard" та "navbox"
204.                    const infoboxRegex = /<table[^>]*class="infobox vcard"[^>]*>[\s\S]*?<\/table>/gi;
205.                    const navboxRegex = /<div class="navbox[^"]*"(?:[\s\S](?!<div class="navbox)[^"]*)*<\/div>/gi;
206.                    const reflistRegex = /<div[^>]*class="reflist"[^>]*>[\s\S]*?<\/div>/gi;
207.
208.                    const infoboxMatches = pageContent.match(infoboxRegex);
209.                    const navboxMatches = pageContent.match(navboxRegex);
210.                    const reflistMatches = pageContent.match(reflistRegex);
211.
212.                    const infoboxHTML = infoboxMatches ? infoboxMatches.join("") : "";
213.                    const navboxHTML = navboxMatches ? navboxMatches.join("") : "";
214.                    const reflistHTML = reflistMatches ? reflistMatches.join("") : "";
215.
216.                    const cleanedNavboxHTML = navboxHTML.replace(/<ul[^>]*>(?=.*<abbr)[\s\S]*?<\/ul>/gi, "");
217.
218.                    setInfoboxHTML(infoboxHTML);
219.                    setNavboxHTML(cleanedNavboxHTML);
220.                    setReflistHTML(reflistHTML);
221.                })
222.                .catch(error => console.error(error));
223.
224.        }, [targetPage]);
225.
226.
227.        useEffect(() => {
228.            const url = «https://${language}.wikipedia.org/w/api.php?&origin=*&titles=${targetPage}&» +
229.                new URLSearchParams({
230.                    action: "query",
231.                    prop: "extracts|images|links|linkshere|pageimages",
232.                    format: "json",
233.                    imlimit: 500,
234.                    iiprop: "url",
235.                    pllimit: "max"
236.                });
237.
238.            const getCity = async () => {
239.                const response = await fetch(url);
240.                const jsonContent = await response.json();
241.                const pageData = getPageData(jsonContent, language);
242.                const pageHTML = getFirstPageExtract(jsonContent);
243.                const images = getPageImages(jsonContent);
244.                setWikiText(pageHTML);
245.                setWikiImages(images);
246.                setPageData(pageData);
247.            };
248.
249.            getCity();
250.
251.        }, [language, targetPage]);
```

```
252.
253.
254.        const handleLinkClick = (event) => {
255.            event.preventDefault();
256.            const linkUrl =
event.target.getAttribute("href").replace("/wiki/", "");
257.            navigate(«/${linkUrl}»);
258.            window.scrollTo(0, 0);
259.        };
260.
261.
262.        return (
263.
264.            <>
265.                {isLoading ? (
266.                    // Відображення інформації про стан завантаження
267.                    <div style={{ margin: "100px" }}>Loading...</div>
268.                ) : (
269.                    // Відображення вмісту сторінки
270.                    <div className="parse">
271.                        <div className="buttons_lang_adds">
272.                            <LanguageSelect language={language}
onLanguageChange={handleLanguageChange} />
273.                            <AddPageButton targetPage={targetPage} />
274.                        </div>
275.                        <Infobox html={infoboxHTML}
onLinkClick={handleLinkClick} />
276.                        <WikiText extract={wikiText}
onLinkClick={handleLinkClick} />
277.                        <div>
278.                            <h2 onClick={handleToggle_T}
className="show-tables-div">Show tables</h2>
279.                            {!isTableHidden && <Navbox
html={navboxHTML} onLinkClick={handleLinkClick} />}
280.                        </div>
281.                        <div>
282.                            <div>
283.                                <h2 onClick={handleToggle_R}
className="show-references-div">Show references</h2>
284.                                {!isReflistHidden && <Reflist
html={reflistHTML} onLinkClick={handleLinkClick} />}
285.                            </div>
286.
287.                        </div>
288.                        <div >
289.                            <h2 onClick={show_hide}
className="images-title">IMAGES</h2>
290.                            {!isImagesBlock && <WikiImages
images={wikiImages} />}
291.                        </div>
292.
293.
294.                    </div>
295.                )}
296.            </>
297.        );
298.    };
299.
300.    export default AppWiki;
```

В.7 Лістинг файлу "taketables.jsx"

```
1.  import React, { useState, useEffect, useRef } from "react";
2.  import "./parseStyle.css";
3.
4.  //Компонент для отримання додаткового інформаційного блоку
5.  const Infobox = ({ html, onLinkClick }) => {
6.      useEffect(() => {
7.          const links = document.querySelectorAll(".infobox");
8.          links.forEach((link) => {
9.              link.addEventListener("click", onLinkClick);
10.         });
11.
12.         return () => {
13.             links.forEach((link) => {
14.                 link.removeEventListener("click", onLinkClick);
15.             });
16.         };
17.     }, [html, onLinkClick]);
18.
19.     return <div className="infobox" dangerouslySetInnerHTML={{
__html: html }}></div>;
20. };
21. //Компонент для отримання таблиць сторніки
22. const Navbox = ({ html, onLinkClick }) => {
23.     useEffect(() => {
24.         const links = document.querySelectorAll(".navbox a");
25.         links.forEach((link) => {
26.             link.addEventListener("click", onLinkClick);
27.         });
28.
29.         return () => {
30.             links.forEach((link) => {
31.                 link.removeEventListener("click", onLinkClick);
32.             });
33.         };
34.     }, [html, onLinkClick]);
35.
36.     return <div className="navbox" dangerouslySetInnerHTML={{
__html: html }}></div>;
37. };
38. //Компонент для отримання посилань сторінки
39. const Reflist = ({ html, onLinkClick }) => {
40.     useEffect(() => {
41.         const links = document.querySelectorAll(".reflist a");
42.         links.forEach((link) => {
43.             link.addEventListener("click", onLinkClick);
44.         });
45.
46.         return () => {
47.             links.forEach((link) => {
48.                 link.removeEventListener("click", onLinkClick);
49.             });
50.         };
51.     }, [html, onLinkClick]);
52.
53.     return <div className="reflist" dangerouslySetInnerHTML={{
__html: html }}></div>;
54. };
55.
56. const TakeTables = () => { };
57.
58. export { TakeTables, Infobox, Navbox, Reflist };
```

В.8 Лістинг файлу "search.jsx"

```
import React, { useState } from "react";
import { useNavigate, Route, Routes } from "react-router-dom";
import '../random/SearchStyle.css'
import AppWiki from "../../../../API/AppWiki";

const SearchPage = () => {
    const [searchTerm, setSearchTerm] = useState("");
    const navigate = useNavigate();

    const handleInputChange = (event) => {
        setSearchTerm(event.target.value);
    };

    const handleFormSubmit = (event) => {
        event.preventDefault();
        navigate(`/${searchTerm}`);
    };

    return (
        <div>
            <form className="search_form"
onSubmit={handleFormSubmit}>
                <input
                    className="search-input"
                    type="text"
                    value={searchTerm}
                    onChange={handleInputChange}
                    placeholder="Enter search term"
                />
                <button className="search-button"
type="submit">SEARCH</button>
            </form>
            <Routes>
                <Route path="/:targetPage" element={<AppWiki />}
/>
            </Routes>
        </div>
    );
};

export default SearchPage;
```

В.9 Лістинг файлу "support.jsx"

```
import React from "react";
import "./h.module.css";

const Support = () => {

    return (
        <>
            <li className="menu__item">
                <a className="menu__link" >Support</a>
            </li>
        </>
    )
};

const SupportPage = () => {
    return (
        <div style={{ margin: "100px" }}>
```

```
18.
19.
20.                 <strong style={{ textAlign: "center", color:
"#F1B24A", paddingLeft: "40px" }}>Military Wikipedia Portal -
Support</strong>
21.                 <p style={{ margin: "10px 0px 10px 0px" }}>For any
inquiries, issues, or feedback related to our Military Wikipedia Portal,
please feel free to reach out to our support team. We are here to assist
you.</p>
22.
23.                 <h2 style={{ marginBottom: "10px" }}>Contact
Information</h2>
24.                 <p style={{ marginBottom: "30px" }}>You can contact
our support team through the following channels:</p>
25.
26.                 <ul style={{ paddingLeft: "40px", marginBottom:
"10px" }} >
27.                     <li style={{ marginBottom: "5px" }}>Email: <a
target="_blank"
href="https://mail.google.com">support.militarywiki@gmail.com</a></li>
28.                     <li style={{ marginBottom: "5px" }}>Instagram: <a
target=" blank" href="https://www.instagram.com/">Military-Wiki</a></li>
29.                     <li style={{ marginBottom: "25px" }}>Discord-
chat: <a target=" blank"
href="https://discord.com/">MilytaryWikiPortal</a></li>
30.                 </ul>
31.
32.                 <h2 style={{ marginBottom: "10px" }}>Frequently Asked
Questions (FAQ)</h2>
33.                 <p style={{ marginBottom: "10px" }}>Before reaching
out to our support team, you might find answers to your questions in our FAQ
section. Here are some common questions:</p>
34.
35.                 <ul style={{ marginBottom: "10px" }}>
36.                     <li style={{ marginBottom: "5px", paddingLeft:
"15px" }}>How do I navigate through the different sections of the Military
Wikipedia portal? - From the Home page, where you can select the section you
are interested in</li>
37.                     <li style={{ marginBottom: "5px", paddingLeft:
"15px" }}>Is the information and sources on the website reliable and up-to-
date? - Yes, because all the information in this version of our grant is
obtained from the Wikipedia knowledge base</li>
38.                     <li style={{ marginBottom: "5px", paddingLeft:
"15px" }}>Can I contribute to the Military Wikipedia portal by submitting new
information or corrections?
39.                         - Yes, if you have any suggestions, please
contact us :)</li>
40.                     <li style={{ marginBottom: "5px", paddingLeft:
"15px" }}>Is there a mobile application for the portal? - In the current
implementation of the application, you can only use this resource, but in the
future it will definitely appear ;)</li>
41.                 </ul>
42.
43.                 <p style={{ marginTop: "50px" }}>If you have any
other questions or need further assistance, please don't hesitate to contact
us. We value your feedback and strive to provide the best user
experience.</p>
44.             </div>
45.         );
46.     }
47.     export { Support, SupportPage }
```

В.10 Лістинг файлу "formreglog.jsx"

```
1.  import React, { useState, useEffect } from 'react';
2.  import { BrowserRouter as Router, Routes, Route, Link,
useNavigate, Navigate } from 'react-router-dom';
3.  import axios from 'axios';
4.  import '..//../API/parseStyle.css'
5.  import AppWiki from '../../API/AppWiki';
6.
7.  function RegistrationForm() {
8.      const [username, setUsername] = useState('');
9.      const [password, setPassword] = useState('');
10.     const navigate = useNavigate();
11.
12.     const handleUsernameChange = (e) => {
13.         setUsername(e.target.value);
14.     };
15.
16.     const handlePasswordChange = (e) => {
17.         setPassword(e.target.value);
18.     };
19.
20.     const handleSubmit = async (e) => {
21.         e.preventDefault();
22.
23.         try {
24.             const response = await fetch('/auth/registration', {
25.                 method: 'POST',
26.                 headers: {
27.                     'Content-Type': 'application/json',
28.                 },
29.                 body: JSON.stringify({ username, password }),
30.             });
31.
32.             const data = await response.json();
33.             console.log(data);
34.
35.             if (response.status === 200) {
36.                 localStorage.setItem('token', data.token);
37.                 navigate('/dashboard'); // Перенаправлення на
сторінку Dashboard після успішної авторизації
38.             }
39.         } catch (error) {
40.             console.error('Ошибка:', error);
41.         }
42.     };
43.
44.     return (
45.         <form className="form-login" onSubmit={handleSubmit}>
46.             <h2 className='form-login-
registaration'>Registration</h2>
47.             <div>
48.
49.                 <input className='enter-username' type="text"
id="username"
50.                     value={username}
onChange={handleUsernameChange} placeholder='Enter your user name' />
51.             </div>
52.             <div>
53.
54.                 <input className='enter-password' type="password"
id="password"
```

```
55.                         value={password}
onChange={handlePasswordChange} placeholder='Enter your password' />
56.                 </div>
57.                 <button className='button' type="submit">Sign
up</button>
58.
59.             </form>
60.         );
61.     }
62.
63.     function LoginForm() {
64.         const [username, setUsername] = useState('');
65.         const [password, setPassword] = useState('');
66.         const navigate = useNavigate();
67.
68.         const handleUsernameChange = (e) => {
69.             setUsername(e.target.value);
70.         };
71.
72.         const handlePasswordChange = (e) => {
73.             setPassword(e.target.value);
74.         };
75.
76.         const handleSubmit = async (e) => {
77.             e.preventDefault();
78.
79.             try {
80.                 const response = await fetch('/auth/login', {
81.                     method: 'POST',
82.                     headers: {
83.                         'Content-Type': 'application/json',
84.                     },
85.                     body: JSON.stringify({ username, password }),
86.                 });
87.
88.                 const data = await response.json();
89.                 console.log(data);
90.
91.                 if (response.status === 200) {
92.                     localStorage.setItem('token', data.token);
93.                     navigate('/'); // Перенаправлення на сторінку
Dashboard після успішної авторизації
94.                 }
95.             } catch (error) {
96.                 console.error('Ошибка:', error);
97.             }
98.         };
99.
100.        return (
101.            <form className="form-login" onSubmit={handleSubmit}>
102.
103.                <h2 className='form-login-
registaration'>Authorization</h2>
104.                <div>
105.                    <input className='enter-username' type="text"
id="username" value={username} onChange={handleUsernameChange}
placeholder='Enter your user name' />
106.                </div>
107.                <div>
108.                    <input className='enter-password' type="password"
id="password" value={password} onChange={handlePasswordChange}
placeholder='Enter your password' />
109.                </div>
```

```
      110.                <button className='button-r ' type="submit">Log
in</button>
      111.
      112.            </form>
      113.        );
      114.    }
      115.
      116.
      117.    function BookmarkItem({ bookmark, onDelete }) {
      118.        return (
      119.            <div className="bookmark-item">
      120.                <Link to={«/${bookmark}»} className="bookmark-
item__title">
      121.                    {bookmark}
      122.                </Link>
      123.                <button className="bookmark-item__delete-button"
onClick={() => onDelete(bookmark)}>
      124.                    Delete
      125.                </button>
      126.            </div>
      127.        );
      128.    }
      129.
      130.    function Dashboard() {
      131.        const [userData, setUserData] = useState(null);
      132.
      133.        useEffect(() => {
      134.            const fetchUserData = async () => {
      135.                try {
      136.                    const response = await
axios.get('/auth/bookmarks', {
      137.                        headers: {
      138.                            Authorization: «Bearer
${localStorage.getItem('token')}»,
      139.                        },
      140.                    });
      141.
      142.                    if (response.status === 200) {
      143.                        const data = response.data;
      144.                        setUserData(data);
      145.                    } else {
      146.                        setUserData(null);
      147.                    }
      148.                } catch (error) {
      149.                    console.error('Ошибка:', error);
      150.                }
      151.            };
      152.
      153.            fetchUserData();
      154.        }, []);
      155.
      156.        const handleDeleteBookmark = async (bookmark) => {
      157.            try {
      158.                const response = await
axios.delete(«/auth/bookmarks/${bookmark}», {
      159.                    headers: {
      160.                        Authorization: «Bearer
${localStorage.getItem('token')}»,
      161.                    },
      162.                });
      163.
      164.                if (response.status === 200) {
```

```
165.                     // Успешное удаление закладки, обновляем список
закладок пользователя
166.                     const updatedBookmarks =
userData.bookmarks.filter((b) => b !== bookmark.id);
167.
168.                     setUserData({ ...userData, bookmarks:
updatedBookmarks });
169.                 }
170.             } catch (error) {
171.                 console.error('Ошибка при удалении закладки:',
error);
172.             }
173.         };
174.
175.         if (!userData) {
176.             return <p>Loading user data...</p>;
177.         }
178.
179.         return (
180.             <div style={{ margin: '100px' }}>
181.
182.                 {userData.bookmarks.length > 0 ? (
183.                     <div>
184.                         <h3 style={{ marginBottom: "30px", textAlign:
"center", fontSize: "18px" }}>YOUR BOOKMARKS </h3>
185.                         {userData.bookmarks.map((bookmark) => (
186.                             <BookmarkItem key={bookmark.id}
bookmark={bookmark} onDelete={handleDeleteBookmark} />
187.                         ))}
188.                     </div>
189.                 ) : (
190.                     <p style={{ marginBottom: "30px", fontSize:
"18px" }}>You don't have any bookmarks yet.</p>
191.                 )}
192.             </div>
193.         );
194.     }
195.
196.
197.
198.     function FormRegLog() {
199.         const [isAuthenticated, setIsAuthenticated] =
useState(false);
200.
201.         useEffect(() => {
202.             const token = localStorage.getItem('token');
203.             if (token) {
204.                 setIsAuthenticated(true);
205.             }
206.         }, []);
207.
208.         const handleLogout = () => {
209.             localStorage.removeItem('token');
210.             setIsAuthenticated(false);
211.         };
212.
213.         return (
214.
215.             <div style={{ margin: '100px' }}>
216.                 <nav>
217.                     <ul className="reg-log">
218.                         {!isAuthenticated && (
219.                             <>
```

```
                                    <li>
                                        <Link
to="registration">Registration</Link>
                                    </li>
                                    <li>
                                        <Link
to="login">Authorization</Link>
                                    </li>
                                </>
                            )}
                        </ul>
                    </nav>

                    <Routes>
                        {!isAuthenticated && <Route path="registration"
element={<RegistrationForm />} />}
                        {!isAuthenticated && <Route path="login"
element={<LoginForm />} />}
                        {isAuthenticated && <Route path="/"
element={<Dashboard />} />}
                        {isAuthenticated && <Route path=":targetPage"
element={<AppWiki />} />}

                    </Routes>

                    {isAuthenticated && (
                        <div style={{ margin: '100px' }}>
                            <button className='user-logout'
onClick={handleLogout}>Log out</button>
                        </div>
                    )}
                </div>

            );
    }
    export { FormRegLog, RegistrationForm, LoginForm, Dashboard };
```

B.11 Лістинг файлу "index.js"

```
    const express = require('express')
    const mongoose = require('mongoose')
    const authRouter = require('./authRouter')
    const PORT = process.env.PORT || 3001


    const app = express()


    app.use(express.json())
    app.use("/auth", authRouter)

    app.get("/", (req, res) => {
        res.send("Працює, все ок!");
    });

    const start = async () => {

        try {
            await
mongoose.connect('mongodb+srv://user:user123@cluster0.br2rnxj.mongodb.net/pro
ject 1?retryWrites=true&w=majority')
```

```
        app.listen(PORT, () => console.log(«server started on port ${PORT}»))
    } catch (e) {
        console.log(e)
    }
}

start()
```

В.12 Лістинг файлу "authcontroller.js"

```
const User = require('./models/User');
const Role = require('./models/Role');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const { validationResult } = require('express-validator');
const { secret } = require("./config");

const generateAccessToken = (id, roles) => {
    const payload = {
        id,
        roles
    };
    return jwt.sign(payload, secret, { expiresIn: "24h" });
};

class authController {
    async registration(req, res) {
        try {
            const errors = validationResult(req);
            if (!errors.isEmpty()) {
                return res.status(400).json({ message: "Ошибка при регистрации", errors });
            }
            const { username, password } = req.body;
            const candidate = await User.findOne({ username });
            if (candidate) {
                return res.status(400).json({ message: "Пользователь с таким именем уже существует" });
            }
            const hashPassword = bcrypt.hashSync(password, 7);
            const userRole = await Role.findOne({ value: "USER" });
            const user = new User({ username, password: hashPassword, roles: [userRole.value] });
            await user.save();
            return res.json({ message: "Пользователь успешно зарегистрирован" });
        } catch (e) {
            console.log(e);
            res.status(400).json({ message: 'Registration error' });
        }
    }

    async login(req, res) {
```

```
40.             try {
41.                 const { username, password } = req.body;
42.                 const user = await User.findOne({ username });
43.                 if (!user) {
44.                     return res.status(400).json({ message:
«Пользователь ${username} не найден» });
45.                 }
46.                 const validPassword = bcrypt.compareSync(password,
user.password);
47.                 if (!validPassword) {
48.                     return res.status(400).json({ message: «Введен
неверный пароль» });
49.                 }
50.                 const token = generateAccessToken(user._id,
user.roles);
51.                 return res.json({ token });
52.             } catch (e) {
53.                 console.log(e);
54.                 res.status(400).json({ message: 'Login error' });
55.             }
56.         }
57.
58.         async getUsers(req, res) {
59.             try {
60.                 const users = await User.find();
61.                 res.json(users);
62.             } catch (e) {
63.                 console.log(e);
64.                 res.status(500).json({ message: 'Failed to retrieve
users' });
65.             }
66.         }
67.
68.         async addBookmark(req, res) {
69.             try {
70.                 const { bookmark } = req.body;
71.                 const userId = req.user.id;
72.                 // Пошук користувача по id
73.                 const user = await User.findById(userId);
74.                 if (!user) {
75.                     return res.status(404).json({ message: 'User not
found' });
76.                 }
77.
78.                 // Додавання закладки
79.                 user.bookmarks.push(bookmark);
80.                 await user.save();
81.
82.                 res.status(201).json({ message: 'Bookmark added
successfully' });
83.             } catch (error) {
84.                 res.status(500).json({ message: 'Failed to add
bookmark', error });
85.             }
86.         }
87.
88.         async deleteBookmark(req, res) {
89.             try {
90.                 const { bookmarkId } = req.params;
91.                 const userId = req.user.id;
92.
93.                 // Пошук користувача по id
94.                 const user = await User.findById(userId);
```

```
95.                 if (!user) {
96.                     return res.status(404).json({ message: 'User not found' });
97.                 }
98.
99.                 // Поиск закладки в списке пользователя
100.                const bookmarkIndex = user.bookmarks.findIndex((bookmark) => bookmark === bookmarkId);
101.
102.                if (bookmarkIndex === -1) {
103.                    return res.status(404).json({ message: 'Bookmark not found' });
104.                }
105.
106.                // Видалення закладки
107.                user.bookmarks.splice(bookmarkIndex, 1);
108.                await user.save();
109.
110.                res.status(200).json({ message: 'Bookmark deleted successfully' });
111.            } catch (error) {
112.                res.status(500).json({ message: 'Failed to delete bookmark', error });
113.            }
114.        }
115.
116.        async getBookmarks(req, res) {
117.            try {
118.                const userId = req.user.id;
119.
120.                // Пошук користувача по id
121.                const user = await User.findById(userId);
122.                if (!user) {
123.                    return res.status(404).json({ message: 'User not found' });
124.                }
125.
126.                // Повертання списку закладок користувача
127.                res.status(200).json({ bookmarks: user.bookmarks });
128.            } catch (error) {
129.                res.status(500).json({ message: 'Failed to retrieve bookmarks', error });
130.            }
131.        }
132.    }
133.
134.    module.exports = new authController();
```

В.13 ЛІСТИНГ ФАЙЛУ "AUTHROUTER.JS"

```
1.      const Router = require('express');
2.      const router = new Router();
3.      const controller = require('./authController');
4.      const { check } = require('express-validator');
5.      const authMiddlewaree = require('./middlewaree/authMiddlewaree');
6.      const roleMiddlewaree = require('./middlewaree/roleMiddleware');
7.
8.      router.post('/registration', [
9.          check('username', "name cannot be empty").notEmpty(),
10.         check('password', "min 4 max 10").isLength({ min: 4, max: 10 })
11.     ], controller.registration);
12.
```

```
13.    router.post('/login', controller.login);
14.    router.get('/users', roleMiddlewaree(['USER']),
controller.getUsers);
15.    router.post('/add-bookmark', authMiddlewaree,
controller.addBookmark);
16.    router.delete('/bookmarks/:bookmarkId', authMiddlewaree,
controller.deleteBookmark);
17.    router.get('/bookmarks', authMiddlewaree,
controller.getBookmarks);
18.
19.    module.exports = router;
```

В.14 ЛІСТИНГ ФАЙЛУ "CONFIG.JS"

```
1.    module.exports = {
2.        secret: "SECRET_KEY_RANDOM"
}
```

В.15 ЛІСТИНГ ФАЙЛУ "AUTHMIDDLEWAREE.JS"

```
1.    const jwt = require('jsonwebtoken')
2.    const { secret } = require('../config')
3.
4.    module.exports = function (req, res, next) {
5.        if (req.method === "OPTIONS") {
6.            next()
7.        }
8.
9.        try {
10.           const token = req.headers.authorization.split(' ')[1]
11.           if (!token) {
12.               return res.status(403).json({ message: "user is not
auth" })
13.           }
14.           const decodedData = jwt.verify(token, secret)
15.           req.user = decodedData
16.           next()
17.       } catch (e) {
18.           console.log(e)
19.           return res.status(403).json({ message: "user is not auth"
})
20.       }
21.   };
```

В.16 Лістинг файлу "rolemiddleware.js"

```
1.    const jwt = require('jsonwebtoken');
2.    const { secret } = require('../config');
3.
4.    module.exports = function (roles) {
5.        return function (req, res, next) {
6.            if (req.method === 'OPTIONS') {
7.                next();
8.            }
9.
10.           try {
11.               const token = req.headers.authorization.split('
')[1];
12.               if (!token) {
13.                   return res.status(403).json({ message: 'User is
not authorized' });
14.               }
15.               const decodedData = jwt.verify(token, secret);
16.               const { roles: userRoles } = decodedData;
```

```
17.
18.                 let hasRole = false;
19.                 for (const role of roles) {
20.                     if (userRoles.includes(role)) {
21.                         hasRole = true;
22.                         break;
23.                     }
24.                 }
25.
26.                 if (!hasRole) {
27.                     return res.status(403).json({ message: 'You do
not have access' });
28.                 }
29.
30.                 next();
31.             } catch (error) {
32.                 console.log(error);
33.                 return res.status(403).json({ message: 'User is not
authorized' });
34.             }
35.         };
36.     };
```