СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки

Кафедра інформаційних технологій та програмування

ПОЯСНЮВАЛЬНА ЗАПИСКА
до кваліфікаційної випускної роботи

освітній ступінь бакалавр

спеціальність 121 “Інженерія програмного забезпечення”

спеціалізація “Інженерія програмного забезпечення”

на тему “Веб-портал Генерального Штабу Збройних Сил України“

Виконав: студент групи ІПЗ-19д _____ М.Д. Панчищенко
(підпис) (ініціали і прізвище)

Керівник _____ ___В.О. Лифар___
(підпис) (ініціали і прізвище)

Завідувач кафедри _____ ___В.О. Лифар___
(підпис) (ініціали і прізвище)

Рецензент _____

Київ – 2023

ЛИСТ ПОГОДЖЕННЯ І ОЦІНЮВАННЯ
дипломної роботи студента гр. ІПЗ-19д Панчищенко М.Д.

Науковий керівник

Доцент, д.т.н. _____ Лифар В.О.

Оцінка наукового керівника: _____

Рецензент _____
ПІБ, місто роботи, посада

Оцінка рецензента: _____

Кінцева оцінка за результатами захисту:
_____

Голова ЕК

_____ Лифар В.О.
підпис

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки
Кафедра інформаційних технологій та програмування

Освітній ступінь бакалавр

спеціальність 121 “Інженерія програмного забезпечення”

спеціалізація “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ

Завідувач кафедри
ІТП, д.т.н., доцент Лифар В.О.
“____”_____2023 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ ВИПУСКНУ РОБОТУ СТУДЕНТУ

Панчищенко Максим Дмитрович
(прізвище, ім’я, по батькові)

1. Тема роботи “Веб-портал Генерального Штабу Збройних Сил України“

Керівник роботи доцент, д.т.н. Лифар Володимир Олексійович,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджений наказом університету від “26” 04 2023 року № 240/15-ОД

2. Строк подання студентом роботи 05.06.2023

3. Вихідні дані до роботи об’єктом дослідження є процес проектування та розробки веб-сайту

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
аналіз предметної області, вибір та обгрунтування технологій, створення веб-порталу, тестування веб-порталу, висновки

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслеників)
_____
_____
_____
_____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____24.03.2023_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання кваліфікаційної випускної роботи	Строк виконання етапів	Примітка
1	Ознайомлення з проблематикою досліджуваної галузі	25.03.2023 – 02.04.2023	
2	Аналіз закордонних веб-порталів Збройних Сил країн-членів НАТО	03.04.2023 — 06.04.2023	
3	Ознайомлення з необхідною літературою та інформаційними джерелами	07.04.2023 — 15.04.2023	
4	Розробка back-end частини проекту	16.04.2023 — 30.04.2023	
5	Розробка front-end частини проекту	02.05.2023 — 18.05.2023	
6	Тестування фінальної версії веб-порталу	21.05.2023 — 27.05.2023	
7	Оформлення пояснювальної записки	28.05.2023 — 01.06.2023	

Студент _____ _М.Д. Панчищенко_
(підпис) (ініціали і прізвище)

Керівник роботи _____ ___В.О. Лифар___
(підпис) (ініціали і прізвище)

РЕФЕРАТ

Текст – 56 с., рис. – 21, табл. – 1, додатків – 2, літературних джерел – 20

Метою роботи є дослідження методів створення прогресивних веб-додатків для розробки веб-сайту Збройних Сил України.

Були проаналізовані архітектурні шаблони проектування, порівняні різні сучасні технології для реалізації веб-сайтів, та обрано Vue.js для клієнтської частини, Node.js для серверної частини та MongoDB як база даних.

В результаті виконаної роботи, було створено адмін-панель для створення новин та лендінг для відображення даних новин.

Система задовольняє всім вимогам, пред'явленим в технічному завданні.

ЗМІСТ

ВСТУП

Актуальність. Інтернет здійснив революцію у світі комунікацій. Використання Інтернету забезпечує механізм розповсюдження інформації та забезпечує середовище для співпраці та взаємодії як між окремими особами, так і між різними групами людей.

Військовий веб-сайт використовується для створення та підтримання комунікації між Збройними Силами та громадськістю, для інформування суспільства щодо стану Збройних Сил, що в сучасних умовах війни є важливим як для більш ефективної діяльності Збройних Сил, так і для зменшення впливу ворожих ІПСО на окремих громадян.

Для забезпечення ефективності веб-сайту необхідно дотримуватись сучасних вимог до їх розробки та підтримки. Веб-сайт для Збройних Сил має виглядати однаково від сторінки до сторінки, а навігація має бути стандартною та передбачуваною по всьому сайту.

Також потрібно відмітити, що архітектурна структура військових веб-сайтів з кожним днем ускладнюється, що обумовлене задачами, які можуть розв’язуватись з використанням такого виду комунікації. Все це вимагає створення забезпечення ефективної розробки та управління військовим сайтом.

Сучасні підходи до розробки сайтів націлені на підвищення їх швидкості, гнучкості та надійності. Використання архітектурних шаблонів проектування націлено на полегшення роботи розробника, а застосування різних open source фреймворків та бібліотек спрощує написання коду та дозволяють вирішувати типові завдання з мінімальними витратами. Крім зазначених раніше переваг, у них вже реалізована стандартна логіка щодо взаємодії з HTTP-сервером та клієнтським API для побудови інтерфейсів, що також мінімізує можливість припуститися помилки через людський фактор.

Отже, задача реалізації сайту для Збройних Сил України на основі сучасних архітектурних рішень та з використанням фреймворків є задачею актуальною, оскільки забезпечить створення адаптивного уніфікованого простого в управлінні програмного продукту.

Об'єкт дослідження. Об’єктом дослідження є процес проектування та розробки веб-сайту.

Предмет дослідження. Предметом дослідження є засоби розробки веб-сайту для Збройних Сил України.

Мета дослідження. Метою роботи є дослідження методів створення прогресивних веб-додатків для розробки веб-сайту Збройних Сил України.

Задачі дослідження.

1. провести аналіз сучасних підходів до архітектури веб-додатків;
2. розглянути сучасні архітектурні шаблони проектування;
3. провести проектування веб-сайту для Збройних Сил України;
4. розробити веб-сайт для Збройних Сил України.

1. АНАЛІТИЧНИЙ ОГЛЯД

1.1. Використання веб-сайту як засобу комунікації в збройних силах України

Останні два десятиліття супроводжувались цифровою революцією та комунікаційним вибухом. Це значно змінило взаємодію, як між окремими людьми, так і між громадянами та владними структурами. У сучасному світі соціальні мережі стали невід'ємною частиною нашого повсякденного життя. Інтернет, фактично, дозволив людям висловлювати свої думки, формувати своє ставлення та бачення ключових моментів в країні, посилюючи їхні голоси [1].

Зазначимо також, що в сучасних умовах ворог може використовувати різні соціальні медіа, неперевірені сайти як засіб для обману, дискредитації, підриву та розчарування народу супротивника. В той же час збройні сили стали важливою зацікавленою стороною як основа безпеки як нації в цілому, так і окремих осіб. Тому для збройних сил важливо здійснювати вплив на суспільство, новобранців, молодих офіцерів через сучасні Інтернет-платформи [1].

Відмітимо, що зі зростанням кількості користувачів Інтернету збройні сили в усьому світі також відчули на собі його вплив як інструменту для зв'язків з громадськістю, неофіційної комунікації, або, навіть, для відстеження новин.

Збройні сили можуть використовувати веб-сайт через його ефективність в охопленні великої аудиторії по всьому світу. Офіційний сайт забезпечує інтерактивний досвід і можливість повідомляти та реагувати на події, коли вони відбуваються; інформаційна інтеграція та здатність синтезувати інформацію; широка аудиторія та можливість повідомлення офіційної інформації; миттєвий доступ до даних та інформації в режимі реального часу.

Для збройних сил соціальні медіа стають важливим засобом спілкування з широкою громадськістю, тобто як організація для просування своєї ролі в суспільстві, залучення новобранців і як платформа для трансляції офіційної інформації.

Враховуючи це, офіційний сайт Збройних Сил повинен мати логічний і простий інтерфейс, який забезпечує зручність використання. Важливою особливістю такого сайту повинна бути кросплатформеність, тобто можливість працювати з ним за допомогою будь-якого гаджету – десктопного комп'ютера, планшета, смартфона.

1.2. Адаптивний веб-дизайн на основі компонентів

Сьогодні графічний інтерфейс більшості веб-додатків складається з багаторазово використовуваних компонентів або модулів. Наприклад, на веб-сторінках можна знайти такі компоненти, як навігаційні елементи "хлібні крихти", контекстні меню, модальні вікна. Нестандартність означає, що вони не мають спеціального представлення в DOM (модель об'єкта документа). Розробник повинен створити ці компоненти за допомогою стандартних елементів DOM і самостійно написати для них логіку. І, якщо для невеликої кількості сторінок, рішення щодо написання таких компонент з нуля може бути правильним, то для сайту з великою кількістю сторінок таке рішення може призвести до породження цілого списку проблем:

– індивідуальне форматування, яке не орієнтується на структурі вмісту, призводить до непослідовного застосування стилів;

– використання стилів не контролюється глобально. Це може призвести до помилок відображення на різних гаджетах або ускладнити чи взагалі унеможливити адаптацію сайту в майбутньому;

– зі збільшенням кількості сторінок зі спеціальними макетами дизайн сайту виглядає все більш випадковим. Це негативно впливає на процес навігації користувача та може створити непотрібну плутанину [2].

Використання фреймворків, з іншого боку, передбачає, що сайт був розроблений з достатньою передбачливістю, щоб відповідати різноманітним авторським потребам і вимогам до вмісту. На відміну від інструментів, які дозволяють розробникам створювати власні сторінки та макети, фреймворки надають розробникам можливість структурувати вміст сторінки, вибирати модулі з бібліотеки багаторазово використовуваних компонентів і застосовувати метадані, які містять інструкції для створення динамічного вмісту та рішення щодо форматування шаблонів [10].

Модульна система має взаємозамінні частини, які називаються компонентами. У веб-розробці компонент – це лише загальний термін для будь-якого заздалегідь визначеного об’єкта, який використовувати на кількох сторінках [3].

2. ІНФОРМАЦІЙНА МОДЕЛЬ ПРЕДМЕТНОЇ ГАЛУЗІ.

2.1. Архітектурні шаблони проектування

Архітектурні шаблони проектування – конструкції, які передбачають розв’язання проблем, які часто виникають у межах певного контексту [4]. Зазвичай шаблон не є закінченою моделлю, яку відразу можна перетворити на код. Шаблон є одним із прикладів розв’язання задачі, який можна застосувати в різних умовах. Тому потрібно враховувати специфіку конкретного проекту. Шаблони високого рівня охоплюють архітектуру всієї програмної системи.

Розглянемо переваги та недоліки використання архітектурних шаблонів проектування.

Переваги використання шаблонів:

- шаблон визначає способи вирішення багатьох задач за рахунок готового набору абстракцій;
- різноманіття варіантів розв’язання;
- прискорення вивчення чужого коду;
- правильне використання шаблонів допомагає розробникам визначити потрібний вектор розвитку та уникнути багатьох проблем, які можуть виникнути в процесі розробки [4].

Сучасна концепція в розробці програмного забезпечення описується терміном відокремлення інтересів і передбачає досить жорстке розділення функціональності. Її реалізація здійснюється за допомогою трирівневої архітектуру. Відповідно до цього підходу існує три рівні (або шари) у програмній системі [5]:

- презентаційний рівень (або рівень представлення): інтерфейс користувача (UI), який показує дані користувачеві та представляє стани або різні форми даних;
- бізнес-рівень: ця частина стосується перевірки даних і бізнес правил та норм;
- рівень доступу до даних: механізм, який підключає програму до обраного середовища для зберігання даних.

Ці шари є способом інкапсуляції логіки, яку виконує кожна частина. Логіка відрізняється від даних, а самі дані відрізняються від інформації. Дані у формі необроблених елементів (наприклад, ціна продукту, операції з банківським рахунком або ставка дисконту) можуть переміщатися між шарами та з'являтися в будь-якому з них, і кожен шар може фіксувати дані та інтерпретувати їх у спосіб, який є зрозумілим для кожного шару. На цьому етапі дані переводяться в інформацію.

Обидва елементи (дані та інформація) можуть з'являтися в будь-якому з вищезгаданих шарів, залежно від функціональності шару (див. рис. 2.1). Логіку можна розглядати як матеріальну версію інформації або програмний код, який з'являється в шарах [6].

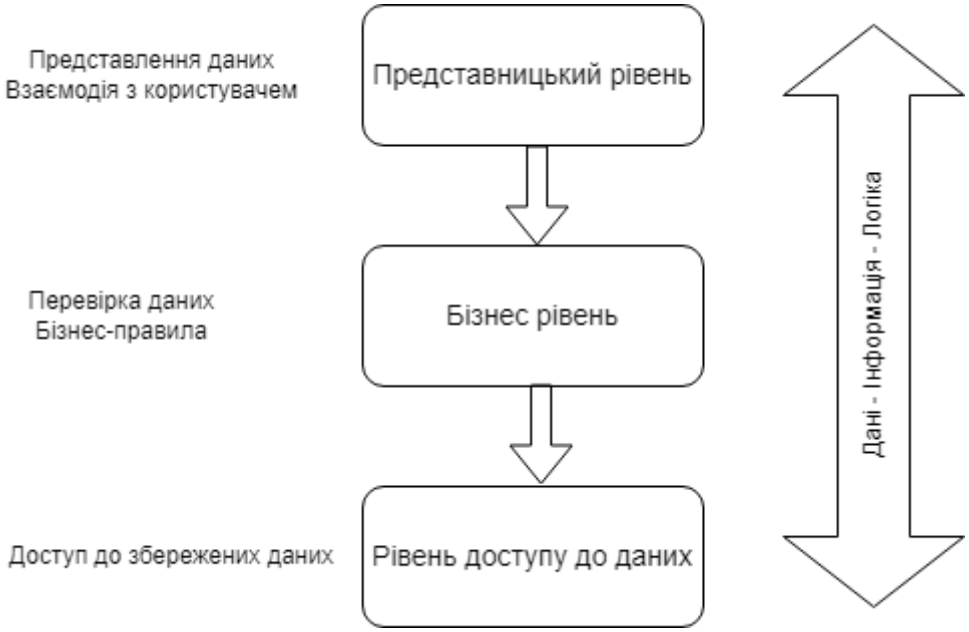

Рисунок 2.1 – Трирівнева структура додатку

Шаблони представлення формують різну сукупність завдань, які виконуються в програмі. Вони, як правило, ідентифікують чіткий інтерфейс, до якого отримує доступ користувач (Перегляд - View). Потім шаблони визначають частину програми, яка має справу з даними, інформацією і логікою програми. Цю частину зазвичай називають моделлю (Model).

Для реалізації зв'язку між View і Model, пропонується багато способів, які утворюють групу рішень, що називається сімейством шаблонів MV* [7].

2.2. Шаблон Model-View-Controller (MVC)

Патерн Model-View-Controller (MVC) включає три частини: модель, представлення та контролер. Перша концепція MVC описана в 1979 році. Тоді вона описувалась для іншого середовища. Поточний шаблон вважається спрощеною версією оригінального шаблону, призначеного для веб-розробки.

Основною ідеєю, закладеною в структуру шаблону MVC, є грамотний підхід до розподілу обов'язків і відповідальності програмного додатку. В патерні використовуються такі компоненти [7]:

– Model (модель), яка відповідає за обробку даних і логіка додатків;

– View (представлення) - представлення даних користувача в будь-якому підтримуваному форматі;

– Controller (контролер) – обробка запитів користувача та виклик відповідних ресурсів.

MVC широко використовується для створення розширених інтерфейсів користувача. Він досить популярний для розробки веб-додатків, Сам додаток поділяється на три основні компоненти, кожен з яких відповідає за різні завдання.

Контролер (Controller) керує запитами користувача (що отримуються у вигляді запитів HTTP GET або POST, коли користувач натискає на елементи інтерфейсу для виконання різних дій). Його основна функція – викликати і координувати дію необхідних ресурсів і об'єктів, потрібних для виконання дій, що задаються користувачем. Звичайний контролер викликає відповідну модель для задачі і вибирає відповідне представлення [6,8].

Модель (Model) – це дані та правила, які використовуються для роботи з даними, що представляють концепцію управління додатком. У будь-якому додатку вся структура моделюється як дані, які обробляються визначеним способом. Модель дає контролеру представлення запитуваних користувачем даних (повідомлення, сторінка книги, фотоальбом тощо). Модель даних буде однаковою, поза залежністю від того, як ми хочемо представляти їх користувачеві. Тому ми вибираємо будь-який доступний вид для відображення даних.

Модель містить найбільш важливу частину логіки додатка, яка вирішує задачу, з якою має справу користувач (форум, магазин, банк і тому подібне). Контролер міститься в основному організаційну логіку для самого додатка.

Представлення (View) забезпечує різні способи представлення даних, які отримані з моделей. Воно може бути шаблоном, який заповнює дані. Може бути кілька різних представлень, і контролер вибирає те, що найбільше підходить для поточної ситуації.

Веб-додаток зазвичай складається з набору контролерів, моделей і представлень. Контролер може бути влаштований як основний, який отримує всі запити та викликає інші контролери для виконання дій залежно від ситуації.

Структурна схема шаблону наведена на рис.2.2 [7].

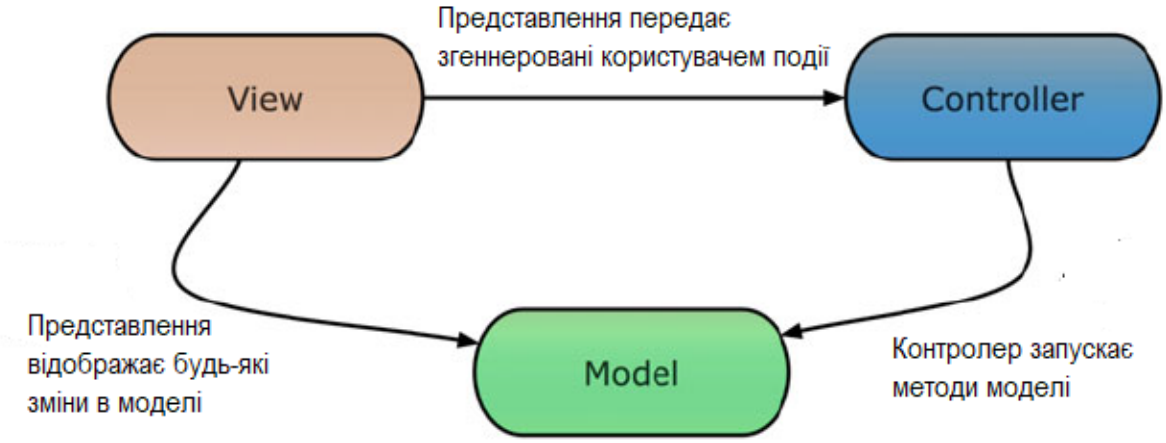

Рисунок 2.2 – Структурна схема шаблону MVC

Головною перевагою використання шаблону MVC є чіткий поділ логіки представлення (інтерфейсу користувача) та логіки програми.

Як вже зазначалось, однією з основних вимог до веб-додатків, є кросплатформеність, тобто можливість виконання на різних пристроях. Зрозуміло, що інтерфейс програми, що надається, повинен відрізнятися, якщо запит приходить з персонального комп’ютера або мобільного телефону. І, якщо модель повертає однакові дані, то контролер як раз і вибирає різні види виведення даних, тобто різні представлення.

Крім ізолювання представлення від логіки програми, шаблон MVC істотно зменшує складність великих додатків. Код виходить набагато структурованим, і, тим

самим, полегшується підтримка, тестування та повторне використання рішень. Введення моделі як компонента з нещільними зв’язками з іншими компонентами реалізує чіткий розподіл проблем. Розробники можуть тестувати представлення та контролери як окремі сутності.

В об’єктно-орієнтованому програмуванні використовується активна модель MVC, де модель – це не лише сукупність коду доступу до даних та СУБД, а й вся бізнес-логіка; також, моделі можуть інкапсулювати інші моделі. Контролери ж, як елементи інформаційної системи, відповідальні лише за:

– прийом запиту від користувача;

– аналіз запиту;

– вибір наступної дії системи відповідно до результатів аналізу (наприклад, передача запиту іншим елементам системи).

Однак взаємодія між представленням і контролером і їх зв’язок із моделлю розмивають поділ між представленням, станом програми та станом представлення. Наприклад, якщо потрібно змінити колір поля редагування через те, що користувач ввів неправильне значення, необхідно зв’язатися з контролером для введення користувача, спостерігати за моделлю для перевірки та реалізувати логіку програмування в представленні, щоб змінити колір поля на основі результатів перевірки. Цей «стан зору» охоплює різні шари шаблону і демонструє, що все ще між компонентами існують доволі щільні зв’язки. Отже, для створення альтернативного представлення потрібно створити новий контролер [7].

2.3. Шаблон Model - View - Presenter (MVP)

Model-View-Presenter (MVP) – шаблон проектування, похідний від MVC, який використовується в основному для побудови інтерфейсу користувача. Шаблон вперше з’явився в IBM, а потім використовувався в Taligent у 1990-х. MVP, на відміну MVC, має дещо інший підхід – представлення не тісно пов’язане з моделлю, як це було в MVC. У цьому підході контролер замінено на представника, а обов’язки, відповідальність і можливості кожної частини змінено. Тепер існує чітке

розмежування між представленням і моделлю, а синхронізацію виконує представник. Структурна схема архітектурного шаблону MVP представлена на рис.2.3 [7].

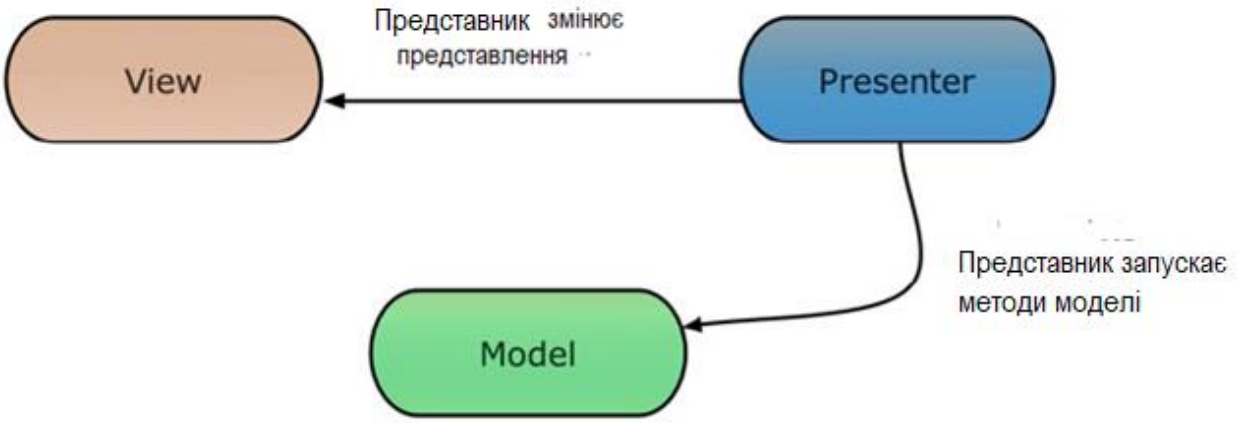

Рисунок 2.3 – Структурна схема шаблону MVP

Компоненти шаблону:

Model – надає дані для інтерфейсу користувача;

View – реалізує відображення даних (моделі) та маршрутизацію користувацьких команд або подій представнику;

Presenter – управляє моделлю та представленням. Наприклад, витягує дані з Моделі та форматує їх для відображення у представленні.

Елемент представник у цьому шаблоні бере на себе функціональність посередника (аналогічно контролеру в MVC) і відповідає за управління подіями інтерфейсу користувача.

Особливості представника в MVP:

- двостороння комунікація з представленням;
- представлення взаємодіє безпосередньо з представником, шляхом виклику відповідних функцій або подій екземпляра представника;
- представник взаємодіє з представленням шляхом використання спеціального інтерфейсу, реалізованого представленням;
- один екземпляр представника пов’язаний з одним представленням.

MVP є шаблоном проектування інтерфейсу користувача, який був розроблений для полегшення автоматичного модульного тестування і поліпшення розподілу відповідальності в презентаційній логіці відділення логіки від представлення.

Зазвичай екземпляр представлення створює екземпляр представника, передаючи посилання на себе. При цьому представник працює з представленням в абстрактному вигляді, через його інтерфейс. Коли викликається подія представлення, вона викликає конкретний метод представника, який не має ні параметрів, ні значення, що повертається. Представник отримує необхідні для роботи методу дані про стан інтерфейсу користувача через інтерфейс представлення і через нього ж передає в представлення дані з моделі та інші результати своєї роботи.

Існує велика кількість реалізацій MVP. Ці реалізації можна розділити за методом доставки даних у представлення на декілька категорій:

– Passive View: представлення містить мінімальну логіку відображення примітивних даних, а рештою займається представник;

– Presentation Model: у представлення можуть передаватися не лише примітивні дані, а й об’єкти;

– Supervising Controller: представленню відомо про моделі, і воно саме забирає із них дані.

Прикладом реалізації шаблону MVP є Windows Forms. Тут використається модифікація Passive View.

MVP має такі переваги у порівнянні з MVC:

– MVP забезпечує перевіреність стану та логіки представлення, переміщуючи їх у представник;

– представлення жорстко відокремлюється від моделі, при цьому зв’язок організується через представника. На відміну від MVC, MVP допускає повторне використання бізнес-логіки без безпосередньої зміни моделі.

Представник відповідає за більшу частину перевірки та зберігає більшу частину стану представлення. Три компоненти менш пов’язані між собою, і цей зв’язок базується на більш гнучких структурах (інтерфейсах). Таке розташування забезпечує кращу можливість тестування, оскільки модель і представлення можна замінити макетними одиницями або різними реалізаціями.

Однак з іншої сторони, інтерфейс користувача стає складнішим, існує потреба в додатковому коді. Більше коду означає більше можливостей для помилок і збільшення зусиль, необхідних для його підтримки.

2.4. Шаблон Model - View - ViewModel (MVVM)

MVVM став альтернативою шаблонам MVC і MVP. SmallTalk представив цю структуру в 1980-х роках, спочатку під назвою Application Model, а пізніше під назвою Presentation Model [9]. В остаточному вигляді він був представлений спільноті Джоном Госсманом в 2005 як модифікація шаблону Presentation Model. MVVM орієнтовано сучасні платформи розробки.

Як було показано, представлення і стан представлення у попередніх підходах (MVC/MVP) все ще взаємопов'язані з моделлю настільки, що провести індивідуальне тестування досить важко. Цей зв'язок суперечить загальному принципу модульного програмування.

Шаблон MVVM має три основні компоненти:

- Model: так само, як у класичному шаблоні MVC, модель є фундаментальними даними, необхідні для роботи програми (класи, структури);
- View: представлення відповідає за графічний інтерфейс, тобто вікно, кнопки тощо.
- ViewModel («Model of View»): модель представлення є з одного боку абстракцією представлення, тобто вона містить иодель, яка перетворена на представлення, а також містить команди, якими може користуватися представлення, щоб впливати на модель [9].

У шаблоні MVVM ViewModel замінює представника і контролера. Обов'язки ViewModel і View тепер різні. Шаблон MVVM прийнято представляти лінійним способом (рис. 2.4). Причина такого подання полягає в тому, щоб підкреслити зміну завдань, які виконуються кожною частиною шаблону, і вказати на потік даних та інформації. Модель залишається в основному такою ж, як і в дизайні MVP. Вона все ще відповідає за доступ до різних джерел даних (наприклад, баз даних, файлів або серверів). Як правило, модель має тенденцію бути дуже тонкою в реалізації MVVM. Представлення представляє дані у відповідному форматі (графічний/нерафічний), відображаючи стан даних, і збирає дії користувача та події. Як і у випадку з моделлю,

представлення в MVVM включають мінімальний код реалізації, лише те, що потрібно для роботи представлення та дозволу дій користувача.

Рисунок 2.4 – Структурна схема шаблону MVVM

Основна особливість MVVM полягає в тому, що вся поведінка виноситься з представлення до моделі представлення. MVVM зручно використовувати замість класичного MVC і йому подібних у тих випадках, коли в платформі, на якій ведеться розробка, є "зв’язування даних" [8].

Зв’язування даних – це процес, який встановлює з’єднання між інтерфейсом користувача і бізнес-логікою. Якщо налаштування та сповіщення встановлено правильно, дані відображають зміни, коли вони зроблені. Це може також означати, що коли інтерфейс користувача змінюється, дані, що лежать в його основі, будуть відображати ці зміни.

У шаблонах проектування MVC/MVP зміни в інтерфейсі користувача не впливають безпосередньо на модель, а попередньо йдуть через контролер або представника. Концепція "зв’язування даних" дозволяє зв’язувати дані з візуальними елементами в обидві сторони. Отже, при використанні цього прийому застосування моделі MVC стає вкрай незручним через те, що прив’язка даних до представлення безпосередньо не укладається в концепцію MVC/MVP.

У MVVM основна частина коду міститься в ViewModel. Концепція ViewModel полягає в тому, що цей компонент представляє спосіб, який визначає вигляд (стан перегляду) і поведінку під час взаємодії користувача (логіка перегляду). Цей компонент є моделлю представлення в тому сенсі, що він описує набір принципів і структур, які представляють конкретні дані, отримані за допомогою моделі. ViewModel обробляє зв’язок між представленням і моделлю, передаючи всі необхідні

дані від представлення до моделі у формі, яку модель може обробити. Перевірка виконується в компоненті ViewModel.

У цьому шаблоні компоненти працюють парами. Представлення знає про ViewModel, оновлює властивості ViewModel і відстежує будь-які зміни, які відбуваються в останній. ViewModel не знає про існування представлення. Подібним чином, модель не знає про модель представлення (або саме представлення), лише модель представлення має доступ до моделі. ViewModel передає події та дані в модель, оскільки вони надсилаються представленням у форми, які модель може інтерпретувати. ViewModel відстежує будь-які зміни, створені моделлю, і, отже, передає в представлення будь-які необхідні сигнали відповідно до представлення і бізнес-правил. Тобто представлення розглядає ViewModel як тунель, який візуалізує або виражає те, що відбувається в представленні. Крім того, представлення може виконувати фільтрацію або перетворення відповідно до необхідної логіки представлення.

Завдяки підходу MVVM розробник може зрозуміти, яка частина програми що робить, і створити слабкі зв’язки між обробкою даних та інформації і способом їх представлення користувачам. Крім того, даний шаблон дозволяє реалізовувати різні дизайни в своїх рамках. Це одна з сильних сторін підходу MVVM; це дозволяє розробити ViewModeli, які відповідають моделям і представленням програмного додатку, замість того, щоб налаштовувати представлення (і, можливо, моделі) відповідно до шаблону, як це часто буває з іншими шаблонами в домені MV*.

З точки зору трирівневої архітектури, тепер модель просунута глибше на рівень даних, оскільки вона здебільшого має справу з даними. Вона все ще охоплює аспекти бізнес-рівня, оскільки часто трансформація даних потрібна на бізнес-рівні. Представлення знаходиться на рівні презентації, як і раніше, а ViewModel тепер виконує ширший діапазон дій і, отже, займає як презентаційний, так і бізнес-рівень. ViewModel реалізує логіку та стан представлення, а бізнес-рівень відповідає будь-якій логіці, яка дозволяє маніпулювати даними способами, які служать логіці представлення.

Таким чином, шаблон MVVM усуває проблему щільних зв’язків між компонентами системи, що забезпечує швидку та надійну розробку з можливістю тестування окремих компонентів.

3. АНАЛІЗ ІНСТРУМЕНТАРІЮ ДЛЯ РОЗРОБКИ ВЕБ-САЙТУ

3.1. Сучасна архітектура веб-додатків

Ранні так звані веб-додатки 1.0 працювали як тонкі клієнти, оскільки на стороні клієнта не запускався код, і клієнт відображав розмітку HTML і команду формування, надану сервером. Як наслідок, додатки були, фактично, дуже повільними, і не передбачали взаємодії з користувачем, оскільки всі дії користувача призводили до перезавантаження всього веб-сайту [11].

Розвиток нових технологій сприяв появі програм, що працюють на стороні клієнта, тому підтримку Javascript було додано до веб-браузерів. Підтримка Javascript і технологія AJAX [12] були величезним кроком вперед. Код, який виконується на стороні клієнта, може ініціювати HTTP-запити та спілкуватися із сервером без перезавантаження сторони користувача після кожної взаємодії користувача. Перезавантаження сайту обмежувалося тими випадками, коли користувач відкривав зовсім інші функції.

Як подальше вдосконалення технології AJAX, були розроблені сучасні клієнтські інфраструктури. Їх основою є клієнт-серверна архітектура. Сервер більше не генерує переглядів будь-яким способом, він надсилає лише необроблені дані JSON/XML і шаблони HTML (частково) до клієнта на вимогу. Клієнтська сторона програми несе повну відповідальність за створення користувацького інтерфейсу з використанням отриманих від сервера даних. Іншими словами, вся логіка інтерфейсу користувача переміщена з серверної на клієнтську сторону програми [11,13].

Сьогодні існують різні способи розробки та створення веб-додатків. Однак будь-яка розробка веб-сайту починається з розуміння того, яку архітектуру вибрати – статичну чи динамічну, і подальшого визначення інструменту розробки [14].

Статичний веб-сайт складається з кількох сторінок HTML, CSS, які пов’язані між собою гіперпосиланнями. Динамічний веб-сайт містить контент, який знаходиться в базі даних і відображається "на льоту", безпосередньо за запитами користувача [15]. За винятком сторінок HTML і CSS, статичний веб-сайт може містити деякі зовнішні сценарії Java, тоді як динамічний веб-сайт містить внутрішні сценарії [15].

Front-end сценарії – це сценарії на стороні клієнта, які виконуються браузером. Що стосується мови, сценаріями на стороні клієнта є JavaScript, тоді як у сценаріях серверної частини використовуються JavaScript, PHP, Python та багато інших [11]. Сценарії на стороні серверу створюють основу для доступу сайту до його бази даних, усіх механізмів, які організовують і забезпечують роботу веб-сайту. А код на стороні клієнта обробляє те, що бачить користувач [4]. Статична архітектура веб-сайту наведена на рис.3.1. На статичному веб-сайті, коли користувач хоче перейти на сторінку, браузер надсилає HTTP-запит "GET" із зазначенням URL-адреси. Сервер отримує запитаний документ зі своєї файлової системи та повертає відповідь HTTP, що містить документ і статус успіху. Якщо з якоїсь причини файл не вдається отримати, повертається статус помилки [14].

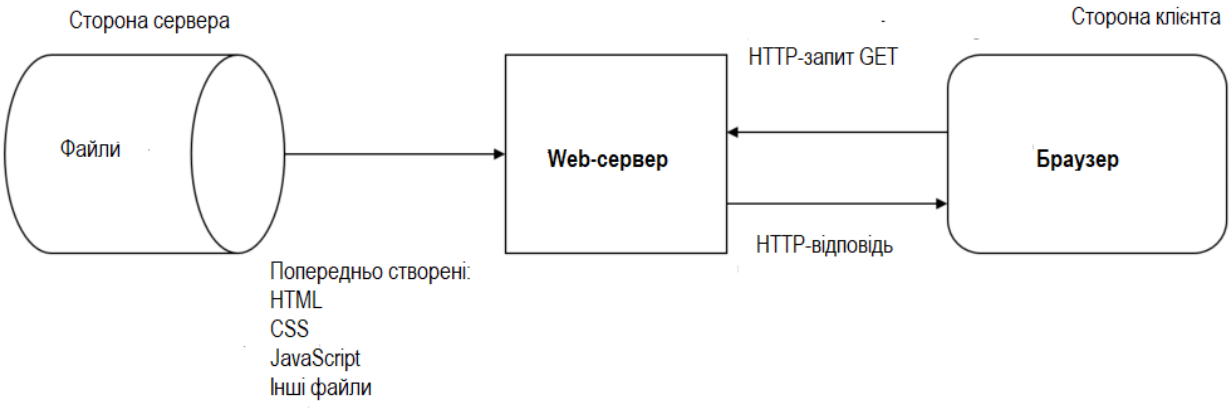

Рисунок 3.1 – Схема статичної архітектури сайту

На рисунку 3.2 показана проста архітектура динамічного веб-сайту. Як і на попередній діаграмі, браузери надсилають HTTP-запити на сервер, потім сервер обробляє запити та повертає відповідні HTTP-відповіді [4].

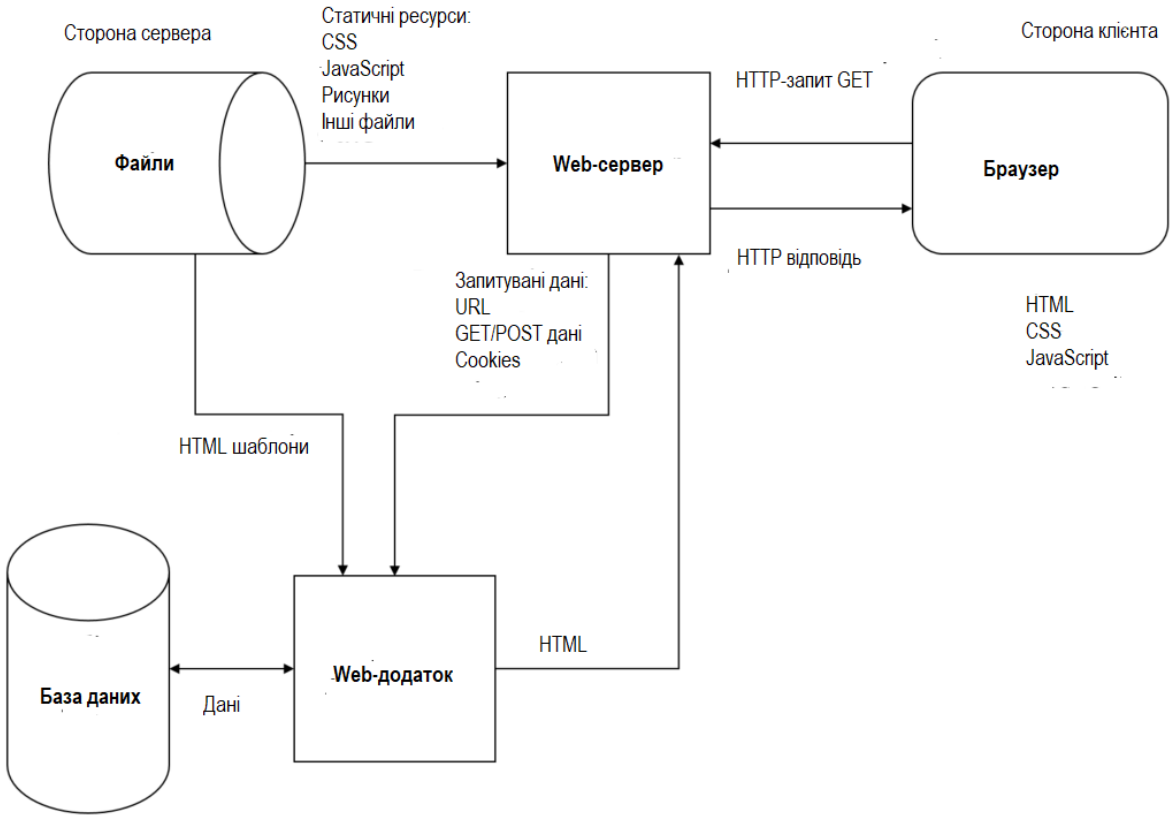

Рисунок 3.2 – Схема динамічної архітектури веб-сайту

Зараз більшість додатків розробляються як динамічні сайти з використанням одного з клієнтських фреймворків. Популярні фреймворки, такі як React, AngularJS і Vue.js займають перші місця серед найкращих фреймворків JavaScript. На рис.3.3 наведені 8 найкращих фреймворків JavaScript для інтерфейсної розробки у 2023 році [16].

Попри всю різноманітність, більшість front-end фреймворків веб-додатків містять спільні функції [16].

В більшості з них підготовлена система шаблонів для розробників. Існує два типи таких систем.

Перший тип системи шаблонів базується на HTML. Цей тип системи шаблонів використовується ASP.NET MVC, Angular, React JSX, Vue.js тощо. Ці типи системи шаблонів подібні, прості у вивченні та використанні, але потребують більше процесорного часу для обробки вхідних даних шаблону та роботи аналізатора шаблонів. Деякі фреймворки компілюють шаблон під час створення веб-додатку.

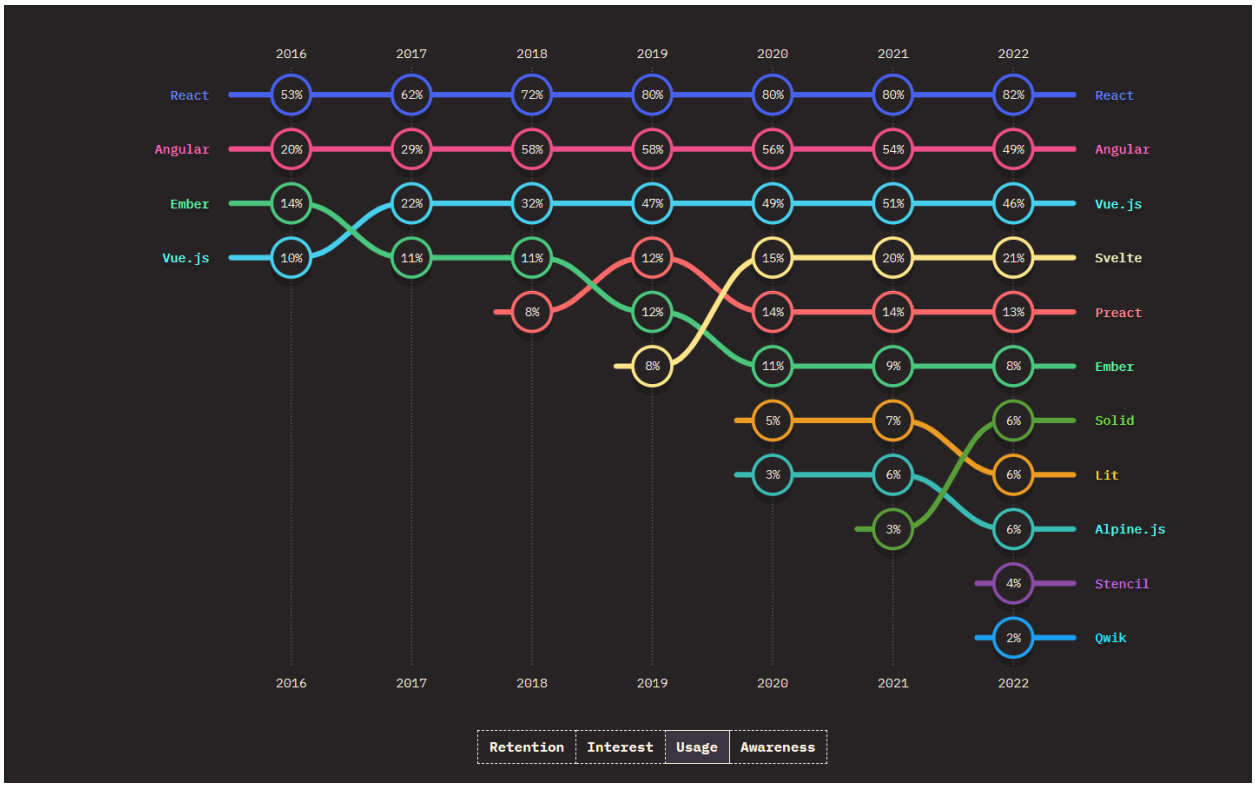

Рисунок 3.3 – Найбільш популярні фреймворки у 2023 році [16]

Другий тип системи шаблонів створює весь HTML з іншої мови. Ці типи систем шаблонів створюють HTML-сторінки з нуля. Вивчення мови для цих систем є складним, оскільки вони мають різний синтаксис. Для створення шаблону використовуються такі мови, як Slim, Haml, Pug тощо. Ці мови є полегшеною версією HTML і дозволяють швидко створити сторінку HTML. Назви тегів HTML зберігаються, але атрибути та вміст тегів записуються у спрощеній формі.

Компоненти являють собою один блок, який містить шаблони. Під час візуалізації сторінки HTML усі спеціальні теги обробляються та замінюються відповідним шаблоном. Компонент визначає спеціальний тег HTML, який можна використовувати в інших шаблонах. Деякі фреймворки дозволяють приєднати до компонента набір стилів, представлених каскадними таблицями стилів (CSS) або мовою, скомпільованою в CSS. Іншою важливою частиною компонента є логіка, пов’язана з компонентом. Логіку можна написати багатьма мовами. Компонент можна вкладати в інший компонент, щоб створити більший функціональний блок.

Маршрутизація – це технологія, яка відображає шлях URL-адреси до сторінки. У багатьох фреймворках маршрутизація має можливість відображати частини сегмента URL-адреси як властивість або змінну.

Сервіси містять функцію, подію, властивості, константи та інші об’єкти, які потрібні веб-додатку. Сервіс може представляти підключення до бази даних, API-клієнт, сховище стану та багато іншого. Сервіси надаються спільно через весь веб-додаток для легкого зв’язку між будь-якими компонентами програми. Будь-який компонент у додатку може підключитися до події, яка надається сервісом. Деякі фреймворки надають послуги через впровадження залежностей. Інші фреймворки використовують сховище служб, яке надає сервіси компоненту.

3.2. Клієнтська частина

Технологія Angular має два покоління. Перше покоління було angularJS. Друге покоління формально відоме як Angular2 [8]. Цей фреймворк розроблений компанією Google INC і є найпопулярнішим фреймворком на стороні клієнта для розробки SPA (односторінкових додатків), розробленого для великих веб-додатків. Angular написаний мовою TypeScript. Архітектура цієї структури розділена на компоненти, модулі, служби та маршрутизацію. Компонент в Angular містить функціональні можливості та шаблони перегляду, які є комбінацією HTML і розмітки Angular. Angular markup забезпечує зв’язування даних між HTML і функціональними можливостями, сценарієм яких є TypeScript. Angular також містить модулі, які групують кілька компонентів в одну функціональну одиницю. Модулі можуть містити компонент, служби та інші модулі в одному блоці. Контекст компіляції оголошення модуля Angular містить усі компоненти та сервіс [17].

Angular – це потужний інструмент і фреймворк, який можна використовувати для створення додатків як Gmail або Google Docs, але, з іншого боку, він має багато функцій, які розробник повинен знати та вивчати, перш ніж почати використовувати цю технологію. Крім того, Angular має складну архітектуру, з якою важко працювати вручну.

Фреймворк Angular активно підтримується – оновлення відбуваються досить часто.

AngularJS не тільки дозволяє створювати добре структуровані, сучасні та чуйні додатки, але також надає засоби для їх розширення різними бібліотеками JavaScript.

React – це бібліотека JavaScript для створення інтерфейсу користувача (UI). React розроблено компанією Facebook Inc. Частиною React є можливість використовувати кращу систему шаблонів під назвою JSX. JSX – це XML-подібний синтаксис, який використовується у файлі JavaScript. У багатьох відношеннях React схожий на Angular як компонентний, розроблений для великих програм і під час використання JSX потребує попередньої обробки файлу JavaScript [18].

React можна використовувати як основу для розробки односторінкових або мобільних додатків. Однак React займається відтворенням даних у DOM, тому для створення додатків React зазвичай потрібні додаткові бібліотеки для керування станом, маршрутизації та взаємодії з API.

React також відомий своєю високою продуктивністю завдяки використанню віртуальної DOM, яка скорочує час, необхідний для повторного рендерингу перегляду, коли стан змінюється. Крім того, React пропонує підтримку візуалізації на стороні сервера, яка може покращити продуктивність веб-додатків, дозволяючи отримати дані до того, як перегляд буде відтворено на стороні клієнта.

Переваги React визначаються наступним:

– Фреймворк вирішує проблеми, коли HTML не є динамічним шаблоном за допомогою JSX.

– React дозволяє повторно використовувати компоненти.

– Функціональний компонент React схожий на функції JavaScript, що полегшує навчання.

– За допомогою рендеринга на стороні сервера, який пропонує React.js, документ створюється на сервері за запитом і надсилається клієнту, що зменшує час завантаження та дозволяє сканерам Google легко сканувати, що покращує SEO.

Недоліками React є можлива крута крива навчання JSX, а також складність JSX, хуків та зв’язування даних.

Основними характеристики React.js є:

1. Віртуальний DOM: кожен створений компонент React перетворюється на об’єктну модель віртуального документа для адаптації всіх нових змін до того, як фактичні зміни будуть внесені в оригінальний DOM.

2. Кросплатформенність: React native – це спеціальний рендерер React, який використовує рідні компоненти для створення інтерфейсів користувача на мобільних пристроях (iOS та Android) замість веб-компонентів.

3. React Flux: React створив архітектуру під назвою flux, яка ефективно обробляє оновлення перегляду (компонентів React) на стороні клієнта та дотримується принципу єдиного зв'язування даних.

Vue.js – це адаптована екосистема, яка масштабується між бібліотекою та повнофункціональним фреймворком [19]. Ця проста прогресивна структура підходить для створення реактивного інтерфейсу користувача. Цей фреймворк використовує найкращі практики інших фреймворків – просте створення шаблону, інтерфейс користувача розділений на компоненти та має віртуальний DOM. Vue.js можна використовувати без node.js і будь-якої компіляції на стороні сервера або клієнта. Розробка веб-додатків за допомогою vue.js набагато простіша, ніж з використанням Angular і React. Цей фреймворк можна поступово додавати до існуючих фреймворків. Розробник-початківець, який хоче використовувати Vue.js, повинен знати лише HTML. CSS і JS.

Фреймворк був створений Еваном Ю і зараз підтримується командою основних розробників. Vue.js має архітектуру, що поступово адаптується, зосереджену на декларативному рендерингу та композиції компонентів. Він складається з набору основних бібліотек, які розроблено таким чином, щоб їх було легко підібрати та використовувати, а також розширювати та налаштовувати.

Основна бібліотека зосереджена лише на рівні перегляду, її легко підібрати та інтегрувати з іншими бібліотеками чи існуючими проектами. Модель одностороннього потоку даних, яка використовується у Vue.js, дозволяє дуже просто представляти додаток, а малий розмір робить її високопродуктивною [19].

Крім того, Vue.js використовує синтаксис шаблону на основі HTML для прив'язки даних до об'єктної моделі візуалізованого документа, і це робиться шляхом компіляції шаблонів у високооптимізований код JavaScript. Усі формати/шаблони Vue є дійсним HTML, який можна проаналізувати браузерами, сумісними зі специфікаціями, і аналізаторами HTML.

Для забезпечення MVVM Vue.js використовує тип реактивності на основі зв’язування даних. Він використовує здатність JavaScript для визначення властивостей. Для частин ViewModel Vue.js визначає пари getter/setter на етапі ініціалізації. Коли відбуваються зміни в ViewModel, фреймворк повідомляють про них і може оновити своє віртуальне дерево DOM. Віртуальний DOM зберігає стан вузлів Vue.js, які не є дійсними вузлами DOM. Синхронізація між DOM і віртуальною DOM є асинхронною щодо змін у віртуальній DOM. Віртуальний DOM накопичує зміни, які потім можна записати в DOM [20].

Vue.js фокусується на односторонньому зв’язуванні даних, він синхронізує DOM з ViewModel. Для забезпечення зворотної синхронізації можна використовувати події. Винятком є поле введення, яке надається з двостороннім зв’язуванням даних Vue.js із коробки.

Перевагами Vue.js є:

– маленький розмір. Vue.js, якщо його завантажити як ZIP, має лише 18k порівняно з іншими фреймворками;

– простий у вивченні та легко адаптований для новачків, оскільки вони більше знайомі зі сценарієм HTML і JS, як правило, в одному файлі;

– реактивність у Vue3 досягається за допомогою Composition API. Composition API надає взаємозамінний спосіб налаштування реактивності, для якого не потрібні декоратори.

Недоліками Vue.js є:

– невелика бібліотечна система плагінів.

– Vue має мовний бар’єр, оскільки деякі з його найбільших спільнот знаходяться в Китаї, тому китайські розробники створюють плагіни та пишуть документацію рідною мовою, що ускладнює її читання іншим розробникам;

– завелика гнучкість фреймворку Vue є недоліком через неузгоджену кодову базу.

Основні характеристики Vue.js:

1. API композиції: використовуючи API композиції, можна застосовувати Redux-подібне керування станом у своєму компоненті без необхідності зовнішніх залежностей. Це також призводить до більш інтуїтивно зрозумілої структури коду, що

полегшує розуміння програми. Крім того, Composition API забезпечує кращу продуктивність, уникаючи необхідності створювати непотрібне повторне відтворення.

2. Загалом Composition API забезпечує більш гнучкий і ефективний спосіб керування реактивністю у Vue3.

3. Однофайловий компонент (SFC): може бути складно керувати кодом, коли він інтерполюється з трьох окремих файлів (HTML, CSS і JavaScript), але за допомогою SFC усе це можна записати в один файл і використовувати в програмі.

4. Використання CSS в одному компоненті файлу також полегшує збереження стилів CSS лише цим компонентом.

5. Перехід/анімація: Vue.js надає вбудований компонент під назвою перехід, який дозволяє застосовувати анімацію до документів HTML, коли вони додаються або видаляються з об'єктної моделі документа DOM. Компонент переходу доступний у кожному компоненті програми Vue.

6. Обчислені властивості: ця функція використовується для оновлення інтерфейсу користувача відповідно до реактивних даних. Доцільно використовувати обчислену властивість під час написання будь-яких обчислюваних функцій, на яких має базуватися інтерфейс користувача, оскільки ми, ймовірно, не захочемо повторюватися, якщо нам знадобиться повторно використовувати функцію в іншому місці нашого компонента, рекомендованого для використання обчисленої властивості.

Отже, і Vue.js, і React використовують віртуальну DOM, надають реактивні та компоновані компоненти, зберігають фокус на основній бібліотеці, а такі проблеми, як маршрутизація та глобальне керування станом, обробляються супутніми бібліотеками. React має значно більшу екосистему, ніж Vue.js, оскільки він старший і популярніший, але екосистема Vue.js також продовжує активно розвиватися. Області продуктивності Vue.js може бути кращим, оскільки він відстежує залежності під час процесу рендерингу, тож система точно знає, які компоненти насправді потребують повторного рендерингу, коли змінюється стан. На відміну від цього, у React, коли змінюється стан компонента, він запускає повторний рендер усього піддерева компонента, починаючи з цього компонента як кореня.

Vue.js використовує класичні веб-технології та базується на них. У React усі компоненти виражають свій UI у функціях візуалізації за допомогою JSX. У Vue.js

також є функції рендерингу і навіть підтримка JSX, однак найзручніше використовувати шаблони. При початковому використанні у випадку Vue.js потрібно лише включити один тег сценарію, а потім можна почати писати код.

Якщо порівняти Angular і Vue.js, то вони мають досить схожий синтаксис. Angular надихнув розробників Vue.js. Angular є доволі жорстким відносно структурованості програми, тоді як Vue.js є більш гнучким, модульним рішенням. Angular використовує двостороннє зв'язування між областями, тоді як Vue.js забезпечує односторонній потік даних між компонентами. Vue.js має більш чіткий розподіл між директивами та компонентами, тоді як в Angular директиви роблять усе, а компоненти – це лише певний тип директив. Vue.js має кращу продуктивність і його набагато, набагато легше оптимізувати, оскільки він не використовує брудну перевірку.

Також Vue.js пропонує офіційну підтримку різноманітних систем побудови, без обмежень щодо структури програми.

На даному етапі були проаналізовані засоби для розробки клієнтської частини та обрано найбільш Vue. Далі потрібно проаналізувати та обрати мову програмування для серверної частини.

3.3. Серверна частина

PHP - це поширена мова програмування загального призначення з відкритим вихідним кодом. PHP спеціально сконструйований для веб-розробок і його код може впроваджуватися безпосередньо в HTML.

Переваги PHP:

- Основи PHP можна швидко вивчити завдяки простому синтаксису. Мова пропонує легкість у вивченні, яку немає жодна інша мова.
- Активна підтримка: завдяки великій спільноті розробників PHP можна отримати швидкі рішення та підтримку.
- Численні фреймворки: PHP може похвалитися низкою фреймворків, які полегшують створення веб-додатків, включаючи Yii, Laravel, CodeIgniter і Symfony.

- Розробка веб-додатків є типовим застосуванням PHP, і вона широко використовується людьми, які створюють веб-сайти.

Недоліки PHP:

- Під час великих навантажень PHP відстає від інших мов програмування, що може призвести до зниження продуктивності.
- Підтримка та розвиток застарілих проектів на PHP може бути складною через його архаїчну архітектуру та код. Значна кількість існуючих PHP-проектів страждає від цієї проблеми.
- Безпека PHP ненадійна, оскільки містить уразливості, якими можуть скористатися хакери.

Node.js - це середовище виконання JavaScript на стороні сервера, яке дозволяє розробникам створювати масштабовані мережеві додатки.

Переваги Node.js:

- Продуктивність: Оскільки Node.js використовує однопотокову асинхронну модель, він може ефективно обробляти багатопотокові операції і забезпечувати високу продуктивність.
- Спільне використання коду: Оскільки Node.js дозволяє використовувати JavaScript як на стороні клієнта, так і на стороні сервера, повторне використання коду є простим і дозволяє розробляти повностекові JavaScript-додатки.
- Комплексне пакування Node.js має комплексну бібліотеку пакетів (npm) з тисячами високоякісних модулів для спрощення розробки додатків.
- Масштабованість: Node.js підходить для розробки масштабованих додатків, особливо при використанні моделі архітектури реактивного програмування.

Недоліки Node.js:

- Високі вимоги до ресурсів: Node.js вимагає більше оперативної пам'яті та процесорної потужності, ніж інші технології.
- Складність в управлінні багатопотоковими процесами: враховуючи асинхронну природу Node.js, керування багатопотоковими процесами може бути складним для розробників.

- Відсутність стандартних бібліотек: У порівнянні з іншими мовами, Node.js не має повного набору стандартних бібліотек, тому розробникам, можливо, доведеться шукати сторонні модулі.

Java - це мова програмування, яка відома своєю переносимістю та використовується для розробки різноманітних застосунків.

Переваги Java:

- Переносимість: Код на Java може працювати на будь-якій платформі, що підтримує віртуальну машину Java (JVM), забезпечуючи високу мобільність додатків.
- Широка спільнота: Java має велику та активну спільноту розробників, які сприяють швидкому виявленню, підтримці та розробці рішень.
- Універсальність: Java надає безліч вбудованих функцій і бібліотек для розробки різних типів додатків, таких як веб-додатки, мобільні додатки та додатки для вбудованих систем.
- Висока надійність: Java має такі механізми, як управління пам'яттю та обробка виключень, які гарантують стабільну роботу додатків.

Недоліки Java:

- Великі обсяги коду: розробка додатків на Java може вимагати більше коду, ніж на інших мовах, що може збільшити час розробки.
- Повільний запуск: Java-додатки можуть запускатися довше, ніж інші мови програмування.
- Високі вимоги до ресурсів: Java-додатки можуть вимагати більше оперативної пам'яті та процесорної потужності.

Проаналізувавши кожну мову програмування, було обрано Node.js для розробки серверної частини. Далі потрібно проаналізувати та обрати базу даних.

3.4. База даних

Проведемо аналіз наступних баз даних: ієрархічну, реляційну та обє’ктно-орієнтовану.

Ієрархічна база даних може бути представлена у вигляді дерева об'єктів на різних рівнях. Між об'єктами існує зв'язок "предок-нащадок". При цьому об'єкт може

не мати дочірніх об'єктів або мати декілька дочірніх об'єктів, в той час як дочірній об'єкт завжди має тільки одного предка.

Характеристики ієрархічної бази даних: дані представлені у вигляді дерева з багатьма рівнями, кожен вузол може мати лише одного батька, але може мати багато дочірніх.

Реляційні бази даних зберігають дані у вигляді таблиць. Найпоширеніші СУБД використовують реляційну модель даних.

Характеристики реляційної бази даних: організована у вигляді таблиць і зв'язків між таблицями, кожна таблиця - це набір рядків і стовпців, де кожен рядок представляє запис, а кожен стовпець - атрибут запису, зв'язки між таблицями визначаються за допомогою ключів.

Об'єктно-орієнтовані. У цьому типі баз даних дані представлені у вигляді об'єктної моделі.

Характеристики об’єктно-орієнтованої бази даних: структурована навколо об'єктів з властивостями (атрибутами) і методами (функціями, які можна викликати), може зберігати інформацію у вигляді об'єктів, які можна розширювати, успадковувати та поліморфізувати, підтримує такі концепції ООП, як успадкування, поліморфізм та інкапсуляція.

Для веб-платформи було обрано об’єктно-орієнтовану базу даних, а саме MongoDB.

MongoDB - одна з найпопулярніших нереляційних баз даних, призначена для зберігання, організації та обробки великих обсягів даних. Основним принципом MongoDB є концепція документно-орієнтованої моделі даних, де дані зберігаються у вигляді документів у форматі BSON (Binary JSON), а Дані зберігаються у вигляді документів у форматі BSON (Binary JSON), що полегшує роботу з даними, які нагадують структури JSON.

Основні переваги MongoDB:

– Гнучкість і швидкість: MongoDB забезпечує гнучкість у роботі зі зміною структур даних. Нові поля можна додавати в документи без зміни схеми бази даних. Це дуже корисно для проектів, де структури даних повинні постійно змінюватися.

MongoDB також дуже швидка завдяки своїй розподіленій архітектурі, яка дозволяє легко розширювати базу даних по горизонталі.

- Простота використання: MongoDB має простий інтерфейс запитів, заснований на мові запитів MongoDB (MQL). Це робить його простим у використанні, особливо для розробників, які знайомі з синтаксисом JSON.
- Масштабованість: MongoDB дозволяє легко масштабувати дані за допомогою горизонтального масштабування. Дані можуть бути розподілені між декількома серверами, а нові сервери можуть бути додані в міру зростання навантаження. Це дозволяє обробляти великі обсяги даних і забезпечувати високу доступність.
- Географічна реплікація MongoDB підтримує географічну реплікацію, яка дозволяє створювати резервні копії бази даних у різних місцях. Це забезпечує високу доступність і надійність даних у разі збою системи або стихійного лиха.
- Повнотекстовий пошук: MongoDB має вбудовану підтримку повнотекстового пошуку, що забезпечує ефективні текстові запити на основі індексів. Це особливо корисно для додатків, яким потрібно отримувати та аналізувати великі обсяги текстової інформації.
- Спільнота та екосистема: MongoDB має активну спільноту користувачів та розробників і надає багато корисних плагінів, бібліотек та інструментів. Документація, навчальні посібники та приклади легко доступні для прискорення розробки проекту.

Проаналізувавши кожну базу даних, було обрано MongoDB.

3.5 Взаємодія клієнтської та серверної частини

Архітектура клієнт-сервер - це модель розподіленої архітектури, яка використовується в багатьох комп'ютерних системах. У цій моделі система складається з двох основних компонентів, клієнта і сервера, які взаємодіють один з одним через мережеве з'єднання, наприклад, Інтернет.

Схему клієнт-сервер показано на рисунку 3.4.

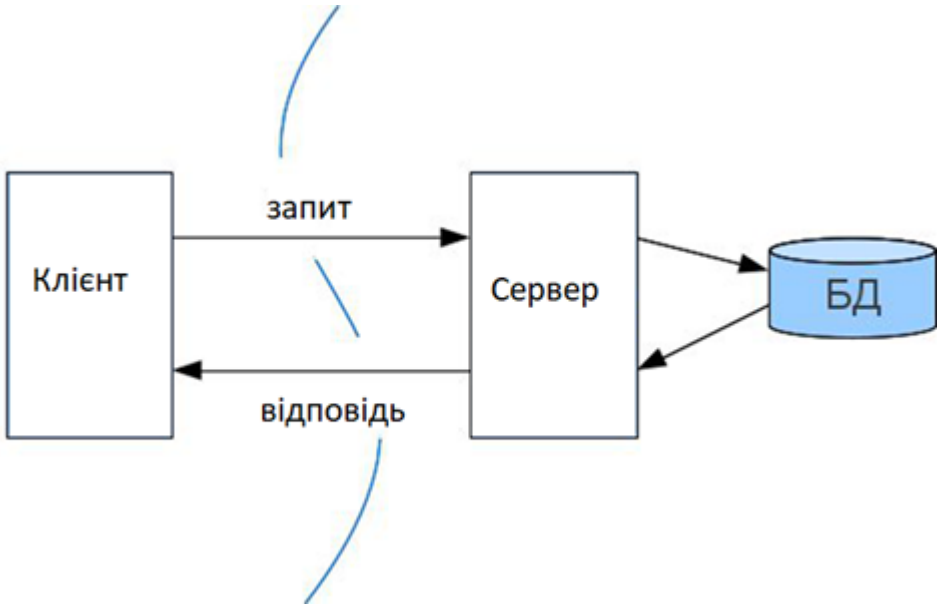

Рисунок 3.4– Схема архітектури клієнт-сервер

Основні особливості архітектури клієнт-сервер

Розподіл ролей. У цій архітектурі клієнтська і серверна частини мають різні ролі і виконують різні завдання. Клієнтська частина взаємодіє з сервером, надсилаючи запити та отримуючи відповіді, як правило, в інтерфейсі користувача або додатку. Серверна частина відповідає за обробку цих запитів і надання необхідних ресурсів та послуг.

Мережева комунікація. В архітектурі клієнт-сервер клієнт і сервер взаємодіють через мережеве з'єднання. Клієнт надсилає запити на сервер, який обробляє ці запити і повертає відповіді клієнту. Це розподіляє завдання між клієнтом і сервером і забезпечує ефективну комунікацію.

Розподіл функцій. Системні функції розподіляються між клієнтом і сервером. Зазвичай клієнт відповідає за відображення даних і взаємодію з користувачем, тоді як сервер відповідає за обробку даних, зберігання інформації, доступ до баз даних і виконання конкретних обчислювальних завдань.

Переваги клієнт-серверної архітектури:

Масштабованість. Архітектура клієнт-сервер дозволяє розширювати систему шляхом додавання нових клієнтів і серверів до мережі. Це дозволяє системі розширити свої можливості та обслуговувати більше користувачів.

Централізоване управління. Сервер централізовано керує обробкою даних і системними ресурсами. Це спрощує управління та підвищує безпеку завдяки централізованому доступу до даних і ресурсів.

Зменшення навантаження на клієнта. В архітектурі клієнт-сервер клієнти зазвичай є простими пристроями з обмеженими ресурсами, такими як персональні комп'ютери та мобільні пристрої. Складні операції та обробку даних виконує сервер, що зменшує навантаження на клієнтський пристрій і забезпечує оптимальну продуктивність.

Безпека. Архітектура клієнт-сервер дозволяє реалізувати різні заходи безпеки на рівні сервера, спрощуючи контроль доступу до даних і запобігаючи несанкціонованим діям. Централізоване управління також дозволяє використовувати механізми резервного копіювання та відновлення даних.

Загалом, архітектура клієнт-сервер є ефективним і поширеним підходом до розробки розподілених систем, що дозволяє розподіляти завдання між клієнтами і серверами, збільшуючи масштабованість і забезпечуючи ефективну комунікацію.

3.6. Середовище розробки Visual Code

Visual Studio Code (VS Code) - це безкоштовний текстовий редактор, розроблений компанією Microsoft. Відкритий, простий у використанні та з великою спільнотою активних користувачів, VS Code набув популярності серед розробників завдяки широкому функціоналу, розширюваності та підтримці різних мов програмування.

На рисунку 3.5 наведено зовнішній вигляд вікна програми Visual Code.

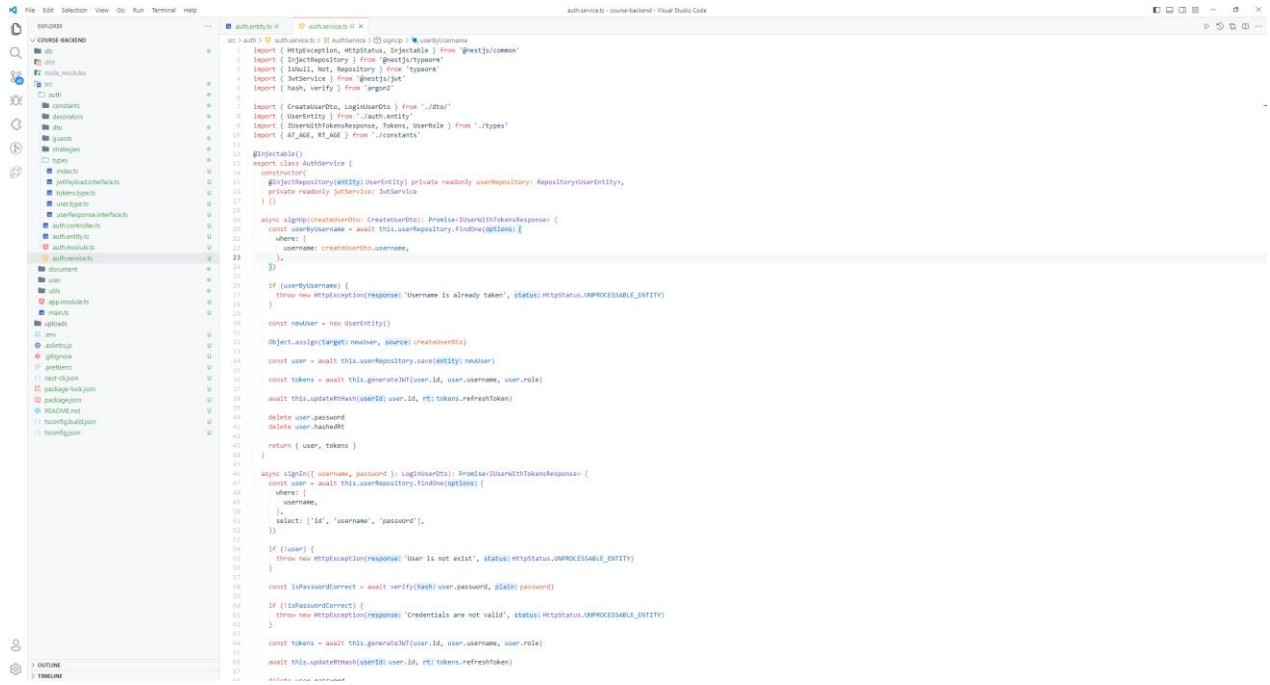

Рисунок 3.5– Зовнішній вигляд вікна програми Visual Code

Ключові особливості середовища розробки Visual Studio Code:

– Розширюваність: VS Code пропонує розширювану архітектуру, яка дозволяє розробникам створювати і використовувати розширення для розширення функціональності редактора. Доступна низка розширень, що охоплюють різні аспекти розробки, такі як підтримка мов програмування, інструменти маніпулювання базами даних, системи контролю версій та структури проектів.

– Підтримка мов програмування VS Code має вбудовану підтримку багатьох популярних мов програмування, включаючи JavaScript, Python, C++, Java та PHP. Він пропонує автозавершення, налагодження коду, пропуски визначення функцій, управління пакетами та багато іншого.

– Інтеграція з інструментами розробки VS Code надає можливість інтегруватися з низкою інструментів розробки, включаючи системи контролю версій Git, інструменти Docker, інструменти тестування та системи збірки проектів. Це робить робочі процеси більш зручними та ефективними.

– Вбудовані функції: VS Code пропонує ряд корисних вбудованих функцій, які полегшують роботу розробників. До них відносяться відмінне підсвічування синтаксису, швидкий пошук і заміна, управління закладками, розділення і розмітка

екрану, а також зручний текстовий редактор з можливістю налаштування колірних схем.

- Віртуальний термінал: VS Code має вбудований віртуальний термінал, який дозволяє виконувати команди безпосередньо в редакторі. Це дозволяє запускати скрипти, виконувати консольні команди та взаємодіяти з іншими інструментами без необхідності перемикатися на зовнішній термінал.
- Портативність: VS Code доступний для широкого спектру операційних систем, включаючи Windows, macOS та Linux. Тому розробники можуть працювати з редактором у своєму улюбленому середовищі.

Середовище розробки Visual Studio Code надає розробникам широкий спектр можливостей для комфортної та ефективної роботи над проектами будь-якого розміру та складності.

4. СТВОРЕННЯ ВЕБ-ПЛАТФОРМИ

В попередньому розділі було описані технології для розробки серверної та клієнської частини та описано середовище розробки.

4.1. Розробка серверної частини

API (Application Programming Interface) - це набір правил і протоколів, які дозволяють програмам спілкуватися між собою; API визначає, які запити можна робити до певної частини програмного забезпечення, які дані можна отримувати і які дії можна виконувати.

Основними функціями API є:

Отримання даних. API дозволяють програмам запитувати та отримувати дані з інших джерел, таких як веб-сервери, бази даних або інші програми. Наприклад, соціальні мережі надають API, які дозволяють розробникам отримувати дані про користувачів, їхні пости, фотографії тощо.

Надання функціональних можливостей. API дозволяють додаткам викликати функції та методи, доступні у віддаленому програмному забезпеченні. Це дозволяє їм використовувати функціональність інших додатків у своїх власних додатках. Наприклад, платіжні шлюзи надають API для здійснення платежів.

Оновлення та зберігання даних. API дозволяють програмам створювати, оновлювати та зберігати дані для інших програм і сервісів. Наприклад, API бази даних дозволяють додаткам створювати нові записи, здійснювати пошук за певними критеріями або оновлювати наявні дані.

Інтегрувати та розширювати функціональність. API дозволяють додаткам спілкуватися та взаємодіяти з іншими додатками, сервісами або пристроями. Це дозволяє використовувати зовнішні ресурси для розширення функціональності програми. Наприклад, API Google Maps можна використовувати для додавання карт і геолокації на веб-сайт або мобільний додаток.

Стандартизація. API встановлює стандарти і правила, які дозволяють різним додаткам взаємодіяти один з одним. Це дозволяє розробникам створювати сумісні додатки незалежно від мови програмування або платформи.

Автоматизація. API можна використовувати для автоматизації процесів і взаємодії додатків без безпосереднього втручання людини. Наприклад, API для автоматичного надсилання електронної пошти дозволяє програмі автоматично надсилати електронні листи без ручного введення даних.

Набір методів, за допомогою яких взаємодіють серверний та клієнтський додатки, наведено в таблиці 3.1.

Таблиця 3.1 – Набір методів сервера

Метод	HTTP Запит	Опис
POST	/auth/ register	Реєстрація користувача
POST	/auth/ login	Логін
GET	/auth/me	Отримати інформацію про авторизованого користувача
POST	/auth/refresh	Оновлення токені користувача
POST	/auth/logout	Очищення токенів користувача
GET	/news/ /:limit/:page	Отримання новин
GET	/news/:id	Отримання однієї новини з необхідної інформацією
GET	/news /list/:limit/:skip	Отримання останніх новин
POST	/news	Створення новини
PUT	/news/:id	Редагування новини
DELETE	/news/:id	Видалення новини
POST	/news/upload	Завантаження фотографій

Для розробки серверної частини було обрано фреймворк Express.js

Express - один з найпопулярніших фреймворків для розробки веб-додатків на мові JavaScript; він базується на Node.js і надає простий і легкий спосіб створення серверних додатків і API.

Ось деякі з переваг Express:

Мінімалізм: Express забезпечує легкий і мінімалістичний спосіб створення веб-додатків. Він не накладає суворих архітектурних правил, дозволяючи розробникам вибирати і розгортати компоненти по-своєму.

Висока продуктивність: Express має низькі витрати пам'яті та часу на обробку запитів, що дозволяє створювати високопродуктивні веб-додатки.

Розширюваність: завдяки великій кількості пакетів проміжного програмного забезпечення, доступних в екосистемі Node.js, Express можна легко розширити для виконання широкого спектру завдань. Наприклад, можна додати проміжне програмне забезпечення для обробки автентифікації, ведення журналів, обробки помилок тощо.

Швидка розробка: Express надає простий API маршрутизації для спрощення розробки шляхів запитів та обробки параметрів. Це дозволяє швидко створювати функціональні веб-додатки з меншою кількістю коду.

Широкі можливості співпраці: Express є одним з найпопулярніших JavaScript-фреймворків для розробки бекенд-версій веб-додатків. Це означає, що вам доступна велика кількість ресурсів, документації та підтримки спільноти, які допоможуть вирішити проблеми, що виникають під час розробки.

Легкий у вивченні: завдяки дружньому API, Express легко освоїти навіть новачкам. Він не вимагає складної конфігурації, тому ви можете швидко почати розробку веб-додатків.

Для авторизації використовується JWT, а саме реалізовано refresh та access токени.

JWT (JSON Web Token) - це відкритий стандарт (RFC 7519), який використовується для безпечної передачі інформації між сторонами у вигляді JSON-об'єктів. JWT складається з трьох частин: заголовка, корисного навантаження та підпису.

Аутентифікація JWT зазвичай використовується в розподілених системах для перевірки ідентичності користувача та доступу. У процесі автентифікації використовуються два типи токенів: токени доступу та токени оновлення.

Токени доступу - це короткострокові токени, що видаються після успішної автентифікації користувача. Цей токен зазвичай має обмежений термін дії і використовується для доступу до захищених ресурсів і API. Він містить інформацію про користувача (наприклад, ідентифікатор користувача, роль тощо) і підписується сервером для перевірки його автентичності.

Токен оновлення - це довготривалий токен, який видається разом з токеном доступу. Його основне призначення - оновити токен доступу, коли термін його дії закінчується. Токен поновлення зберігається в безпечному місці, наприклад, у файлі cookie або в локальному сховищі браузера. Коли термін дії токена доступу закінчується, клієнтська програма може використовувати токен оновлення для отримання нового токена доступу без необхідності повторної автентифікації користувача.

Процес автентифікації за допомогою JWT і токена оновлення включає наступну послідовність дій

1. Користувач надає автентифікаційну інформацію (наприклад, ім'я користувача та пароль).
2. Сервер перевіряє ці дані і, якщо вони правильні, видає токен доступу і токен оновлення.
3. Клієнт зберігає токен оновлення в безпечному місці (наприклад, у файлі cookie або в локальній пам'яті браузера).
4. Клієнт використовує токен доступу для доступу до захищеного ресурсу або API, передаючи його в заголовку запиту або параметрі запиту.
5. Якщо токен доступу стає недійсним (наприклад, через закінчення терміну дії), клієнт може використати токен оновлення для отримання нового токену доступу.
6. Сервер перевіряє дійсність токена оновлення і видає новий токен доступу, якщо він дійсний.

7. Клієнт використовує новий токен доступу для подальшого доступу до ресурсу.

Цей процес може повторюватися (наприклад, через певну кількість разів або певний період часу), доки токен оновлення також не стане недійсним. У цьому випадку користувач повинен пройти повторну автентифікацію та отримати новий токен поновлення.

Використання токену оновлення допомагає забезпечити безпеку та зменшити навантаження на сервер, оскільки не вимагає повторної автентифікації кожного разу, коли термін дії токену доступу закінчується.

Для завантаження фотографій використовується проміжне програмне забезпечення multer.

Multer - це проміжне програмне забезпечення для обробки завантаження файлів на Express.js, яке дозволяє легко та ефективно обробляти файли, надіслані через форми та API-запити. Він може отримувати файли від користувачів і зберігати їх на сервері або використовувати для подальшої обробки.

Нижче наведено деякі з переваг використання Multer в Express.js додатках

Легка інтеграція: Multer легко інтегрується з Express.js і діє як проміжне програмне забезпечення; його можна використовувати разом з існуючими маршрутами і проміжним програмним забезпеченням в Express-додатках.

Підтримує багатофайлове завантаження: Multer може завантажувати кілька файлів одночасно, а не тільки один файл; його можна налаштувати на обробку одного або декількох файлів в одному запиті; його можна налаштувати на одночасне завантаження декількох файлів; його можна налаштувати на одночасне завантаження декількох файлів; і його можна налаштувати на одночасне завантаження декількох файлів.

Керування розміром файлів: Multer надає можливість обмежити розмір завантажуваних файлів. Встановивши максимальний розмір файлу, який можна завантажити, ви можете забезпечити безпеку і уникнути великих файлів, які можуть вплинути на продуктивність сервера.

Збереження файлів. Multer дозволяє легко зберігати завантажені файли на сервері. Ви можете визначити шляхи для зберігання файлів і налаштувати імена файлів, щоб забезпечити унікальність і впорядкованість файлової структури.

Робота з різними типами файлів: Multer підтримує завантаження різних типів файлів, таких як зображення, відео, аудіо та документи. Ви можете обмежити типи файлів, які можна завантажувати, або встановити перевірку типу файлу перед збереженням.

Доступ до метаданих файлів: Multer надає зручний спосіб доступу до метаданих завантажених файлів. Ви можете отримати оригінальне ім'я файлу, тип MIME, розмір файлу та іншу інформацію про файл для подальшої обробки або збереження.

Після того як користувач завантаже файл, у відповіді він отримає посилання на файл.

Використовуючи можливості статичних файлів у express, ми отримаємо наступну логіку:

1. завантажуємо файл, також генеруємо префікс для кожного файлу, аби гарантувати, що не буду дублікату;
2. express перевіряє кожний запит на наявність відповідного файлу в цьому каталозі. Якщо файл знайдений, він автоматично відправляється як відповідь на запит.

На даному етапі було описано реалізацію серверної частини.

4.2. Розробка бази даних

Для бази даних було обрано Mongo DB. Структуру бази даних наведено на рисунку 4.1.

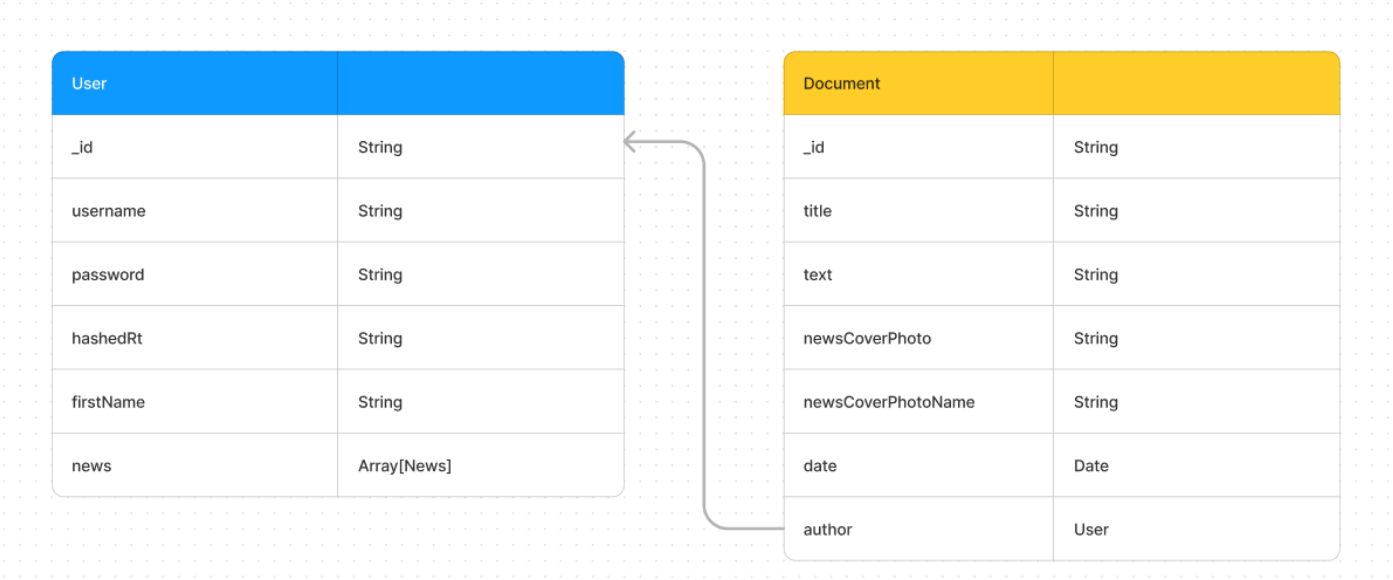

Рисунок 4.1 – Структура бази даних

Таблиця User включає в себе:

- _id: string, ідентифікатор користувача;
- username: string, логін користувача;
- пароль: string, пароль користувача;
- hashedRt: string, токен оновлення користувача;
- firstName: string, ім’я користувача;
- lastName: string, прізвище користувача;
- news: [News]: зовнішній ключ.

Таблиця News включає в себе:

- _id: string, ідентифікатор новини;
- title: string, заголовок новини;
- text: string, текст новини;
- newsCoverPhoto: string, посилання на фотографію новини;
- newsCoverPhotoName: string, ім’я фотографії новини;
- date: Date, дата створення новини;
- author: string, зовнішній ключ.

На даному етапі було описано структуру бази даних.

4.3. Розробка адмін-панелі

Панель адміністрування була розроблена для того, щоб дозволити адміністраторам зручно створювати, переглядати та редагувати новини. Панель адміністрування була розроблена з урахуванням оптимальної взаємодії з користувачем і адаптована для використання на різних мобільних пристроях.

Функції панелі адміністрування дозволяють адміністраторам легко створювати новини, вказуючи необхідну інформацію, таку як назва, зміст та обкладинка новини. Зручний інтерфейс панелі надає швидкий доступ до необхідних полів та опцій, що дозволяє адміністраторам швидко створювати новини.

Панель також дозволяє адміністраторам переглядати існуючі новини у зручному форматі, дозволяючи користувачам швидко та ефективно знаходити та відображати необхідну інформацію.

Редагування новин також здійснюється за допомогою зручного інтерфейсу на екрані адміністрування. Адміністратори можуть змінювати заголовок, зміст та інші деталі новини. Це дає адміністраторам повний контроль над своїми новинами і дозволяє їм оновлювати інформацію в режимі реального часу.

Адмін-панель надає адміністраторам зручний і потужний інструмент для створення, перегляду та редагування новин. Його функціональність, зручний інтерфейс і сумісність з мобільними пристроями роблять роботу з новинами ефективною і приємною для адміністраторів.

На рисунку 4.2 наведено форму головну сторінку адмін-панелі

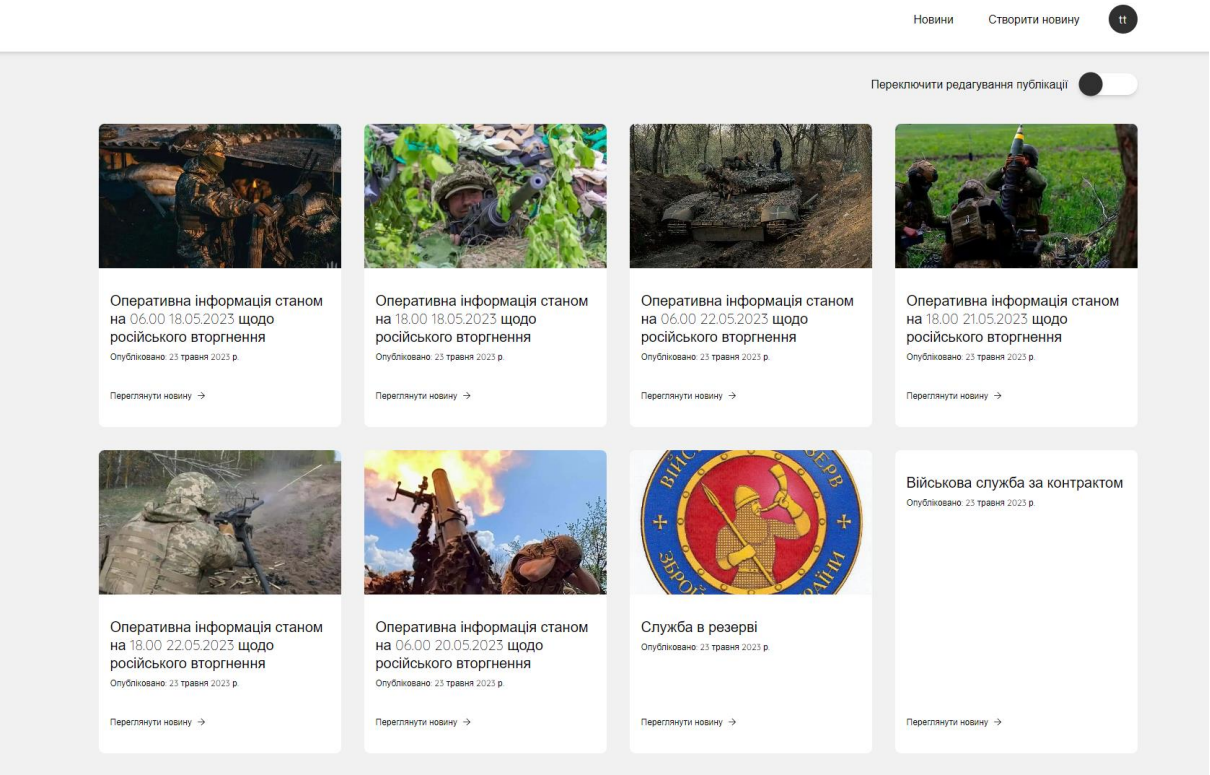

Рисунок 4.2 – Головна сторінка

На головному екрані відображено останні опубліковані новини з відображенням назви новини та обкладинки.

На головному екрані також представлена кнопка для редагування, натиснувши її ми можем обрати, які новини ми хочемо видалити або редагувати. Даний функціонал наведено на рисунку 4.3

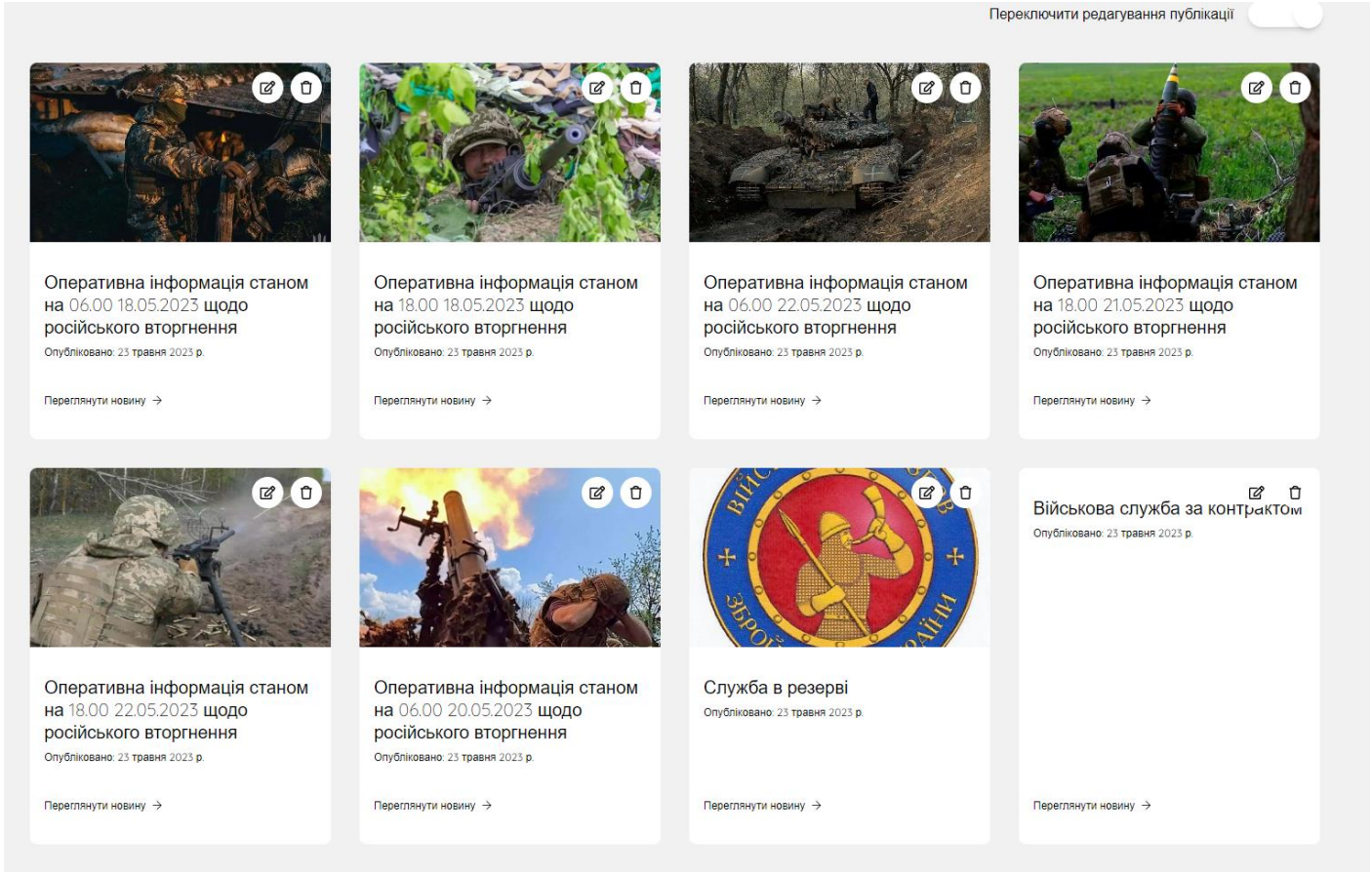

Рисунок 4.3 – Редагування новини

В навігації ми можемо перейти на сторінку створення новини. Де можна ввести назву новини, завантажити обкладинку, вписати текст новини, побачити попередній вигляд новини та опублікувати новину. Сторінку створення новини наведена на рисунку 4.4

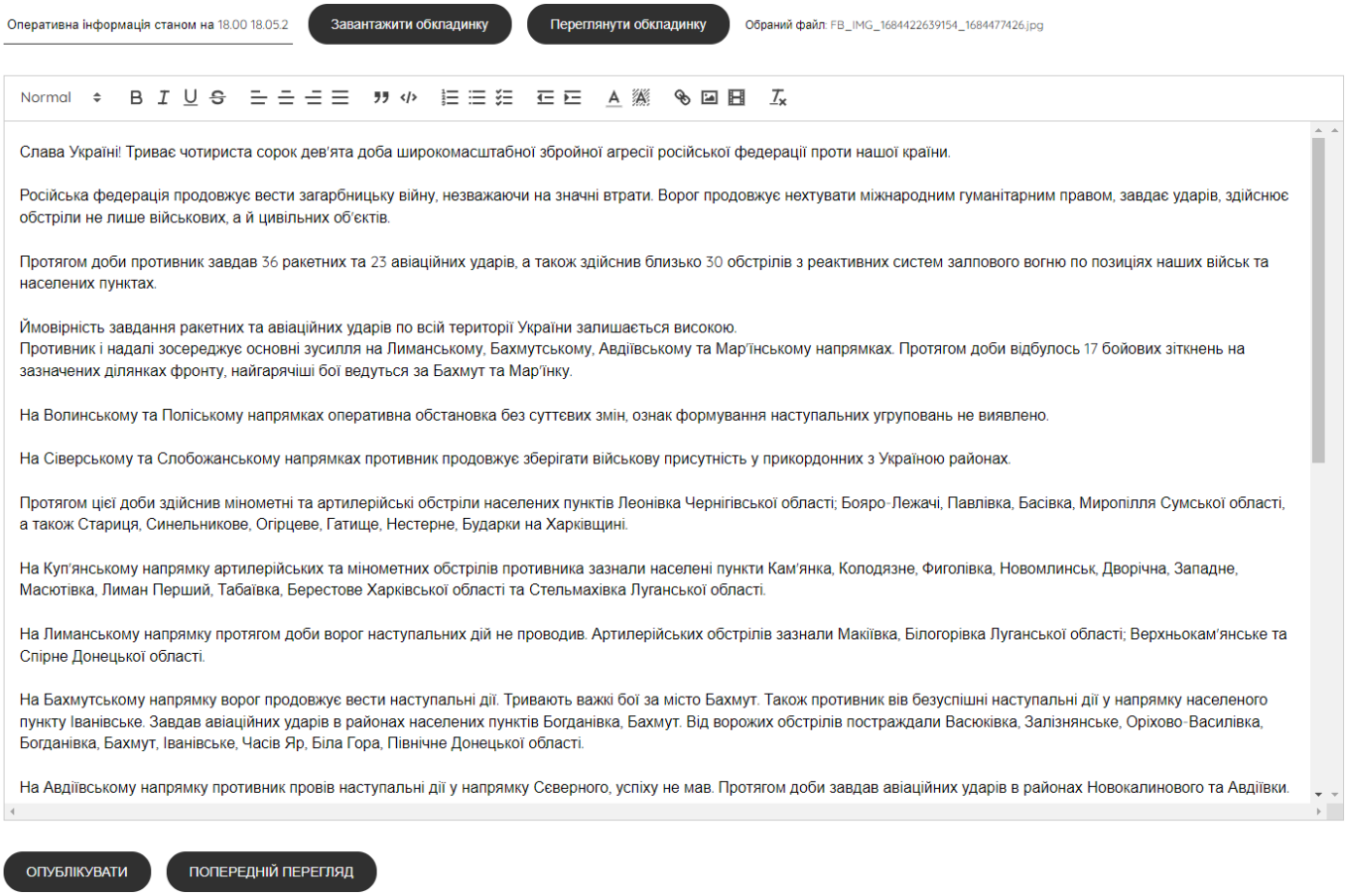

Рисунок 4.4 – Створення новини

Особливістю є те, що він сумісний з мобільними пристроями. Це означає, що адміністратори можуть зручно користуватися дашбордом на своїх смартфонах і планшетах, незалежно від місця і часу. Інтерфейс дашборду автоматично адаптується до розміру екрану мобільних пристроїв, що дозволяє зручно та ефективно працювати з новинами навіть на невеликих екранах. На рисунку 4.5 наведено головну сторінку з мобільного пристрою.

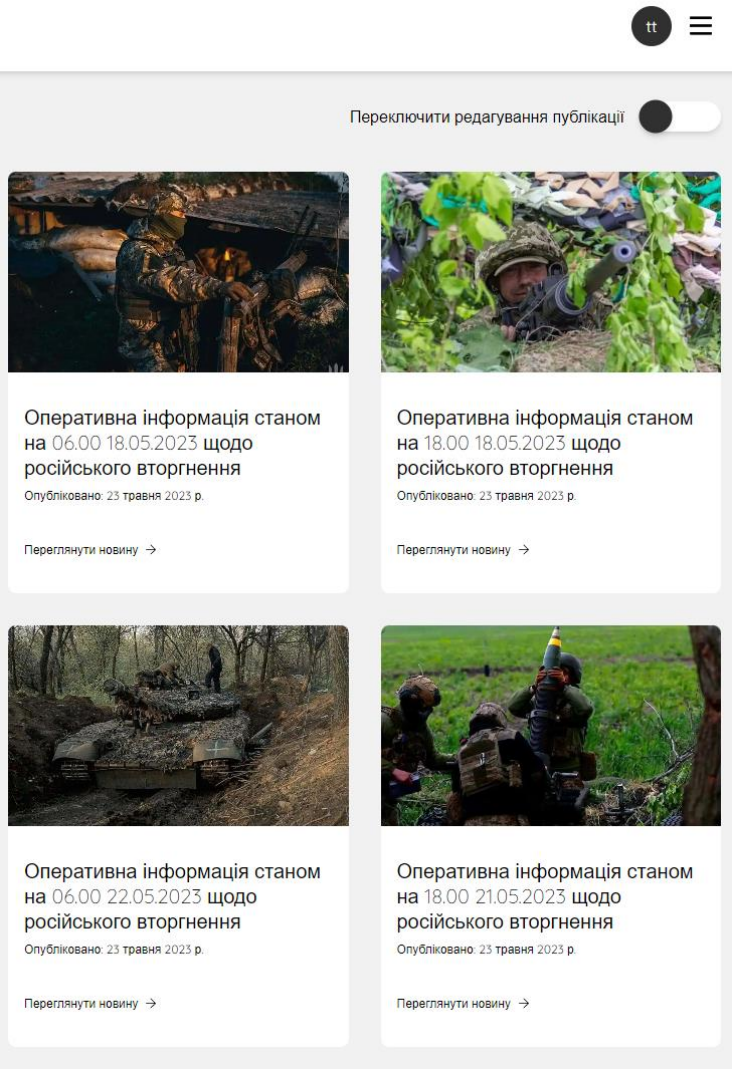

Рисунок 4.5 – Головна сторінка з мобільного пристрою

В даному етапі було описано реалізацію адмін панелі. В наступному етапі розглянемо реалізацію лендінгу.

4.4. Розробка лендінгу

На головній сторінці користувачі можуть переглянути останні новини та бути в курсі поточних подій. Головна сторінка також пропонує користувачам можливість дізнатися про рід військ або командування збройних сил, перейшовши на відповідну сторінку.

Однією з головних переваг цільової сторінки є її гарний дизайн. Кожна сторінка, включаючи сторінку новин, має привабливий дизайн, який виглядає професійно і справляє на користувача приємне враження.

Головна сторінку лендінгу наведена на рисунку 4.6.

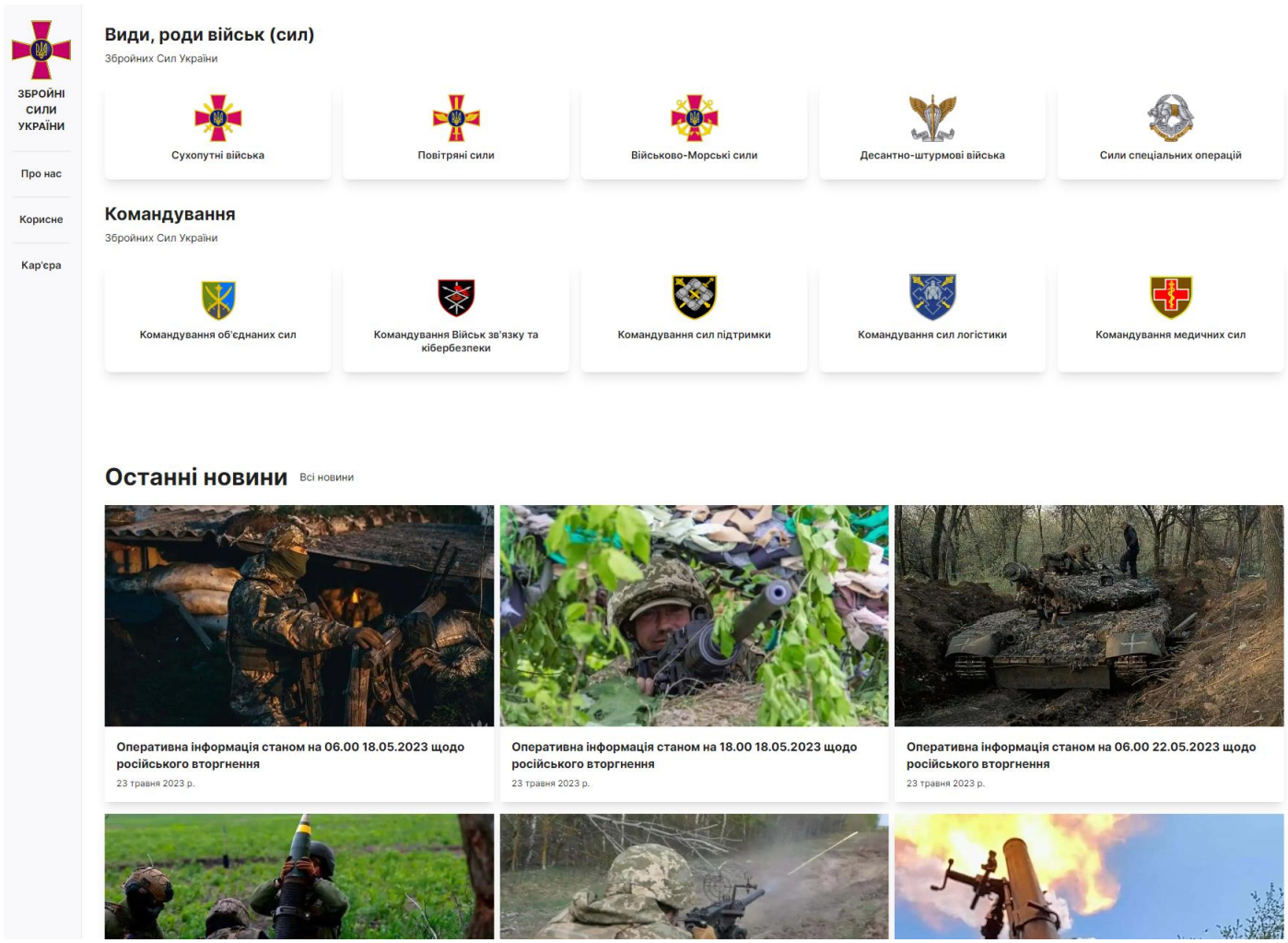

Рисунок 4.6 – Головна сторінка лендінгу

Новини представлені у зручному форматі, що дозволяє користувачам швидко прочитати текст статті, ознайомитися з заголовком та переглянути схожі новини, які їх цікавлять. Сторінка новини наведена на рисунку 4.7.

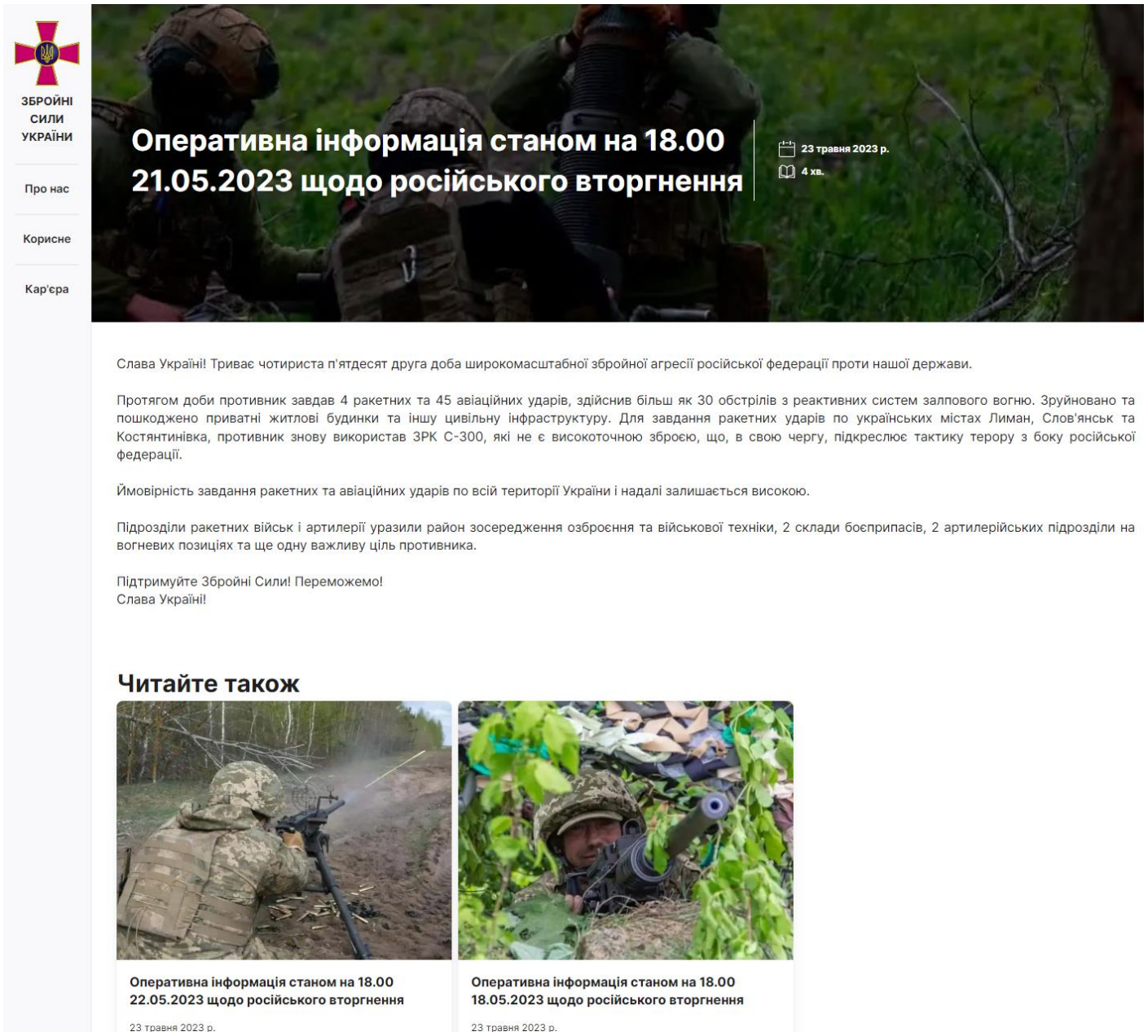

Рисунок 4.7 – Сторінка новини

Інтерфейс лендінгу автоматично адаптується до розміру екрану мобільних пристроїв, що дозволяє зручно переглядати новини навіть на невеликих екранах. На рисунку 4.8 наведено сторінку новини з мобільного пристрою.

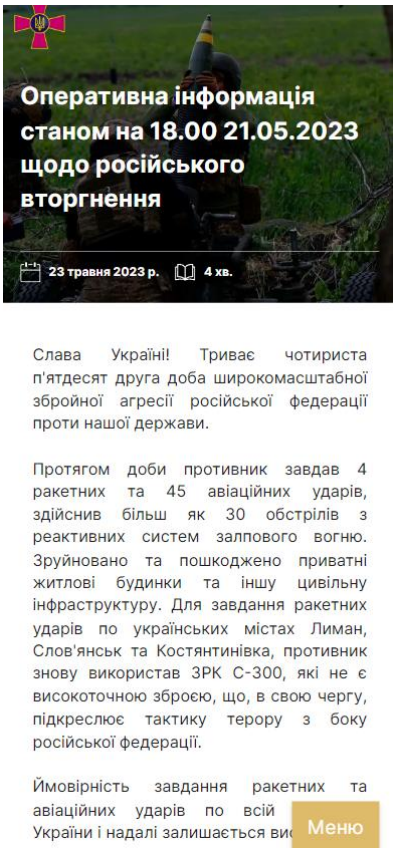

Рисунок 4.8 – Сторінка новини з мобільного пристрою

Загалом, розроблений лендінг забезпечує зручний доступ до широкого спектру інформації, включаючи новини, види військ та командування. Зрозумілий та зручний інтерфейс сприяє комфортній навігації для користувача та дозволяє йому швидко та ефективно знаходити потрібну інформацію.

5. ТЕСТУВАННЯ ВЕБ-ПЛАТФОРМИ

Тестування - це процес перевірки та оцінки функціональності, якості та надійності програмного продукту. Це важлива частина розробки програмного забезпечення, де можна виявити помилки, дефекти та недоліки і переконатися, що додаток працює відповідно до вимог та очікувань.

5.1. Тестування за допомогою Postman

Одним з найпоширеніших інструментів для тестування API (інтерфейсу прикладного програмування) є Postman - широко використовуваний інструмент для розробки, тестування і документування API; він виконує HTTP-запити до веб-сервісу і надає зручне середовище для виконання HTTP-запитів до веб-сервісу і тестування відповідей.

Postman дозволяє створювати і виконувати різні типи запитів, такі як GET, POST, PUT і DELETE, і відстежувати відповідь від сервера. Postman дозволяє відправляти запити, що містять різні параметри, заголовки і тіла запитів, і може перевіряти, чи повертається відповідь з очікуваними даними.

Postman також надає можливість створювати автоматизовані тестові кейси, які виконують серію запитів і перевіряють результати. Це дозволяє автоматизувати процес тестування і швидко тестувати різні сценарії взаємодії з API.

Таким чином, тестування за допомогою Postman дозволяє розробникам і тестувальникам перевіряти і валідувати API, визначати коректність запитів і перевіряти, чи повертається відповідь, як очікувалося. Використання Postman спрощує процес тестування API, легко виявляти помилки і проблеми в програмному забезпеченні.

На рисунку 5.1 наведено результат тестування логіну.

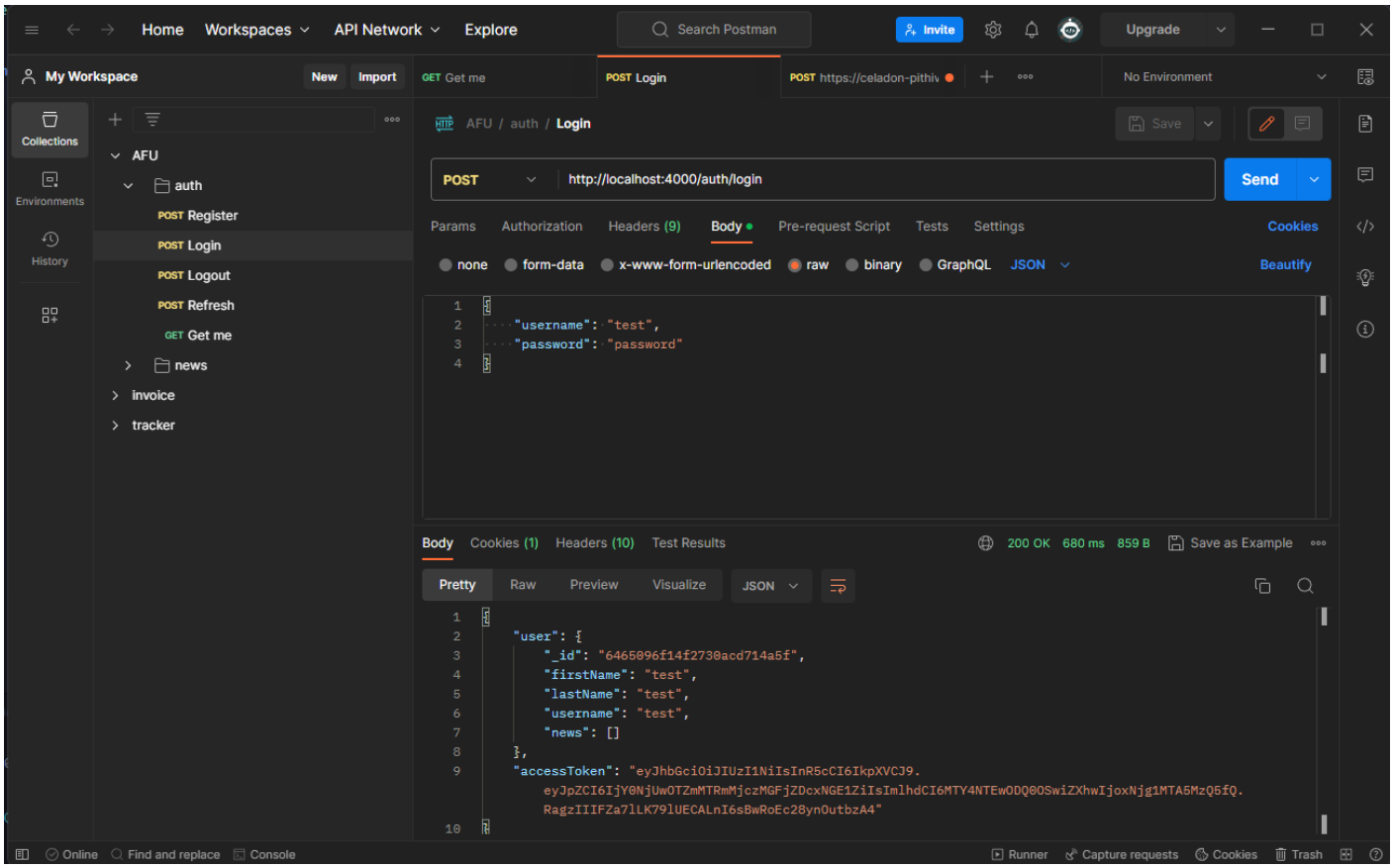

Рисунок 5.1 – Результат тестування логіну

В результаті тестування логіну, отримано очікуваний результат, а саме ідентифікатор, логін, ім’я, прізвище, новини, які створив користувач та токен доступу.

На рисунку 5.2 наведено результат тестування отримання новин.

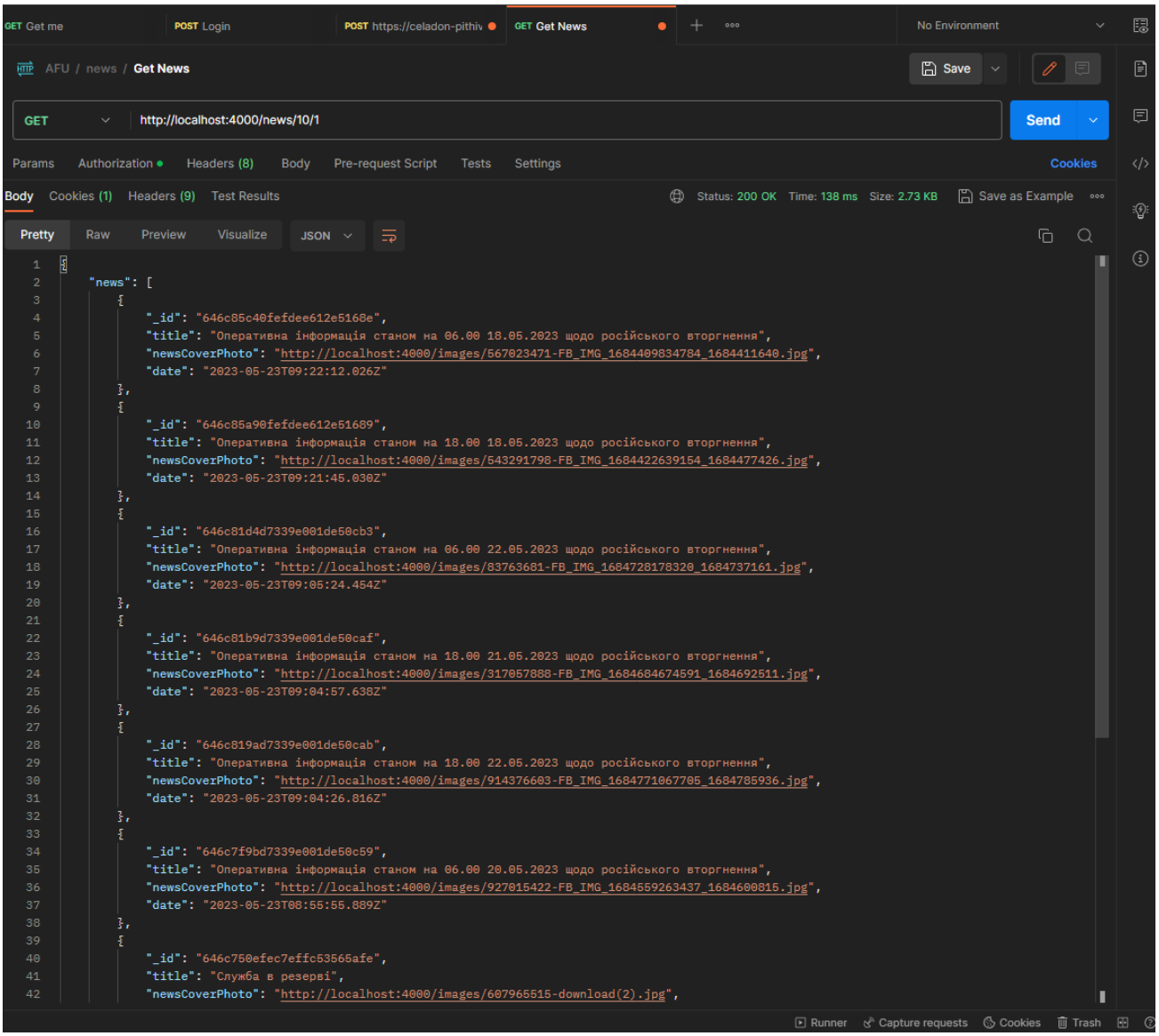

Рисунок 5.2 – Результат тестування отримання новин

В результаті ми отримати очікуваний результат, а саме масив новин, де кожна новина містить ідентифікатор, заголовок, текст новини та дата створення.

Далі протестуємо отримання специфічної новини, результат наведено на рисунку 5.3

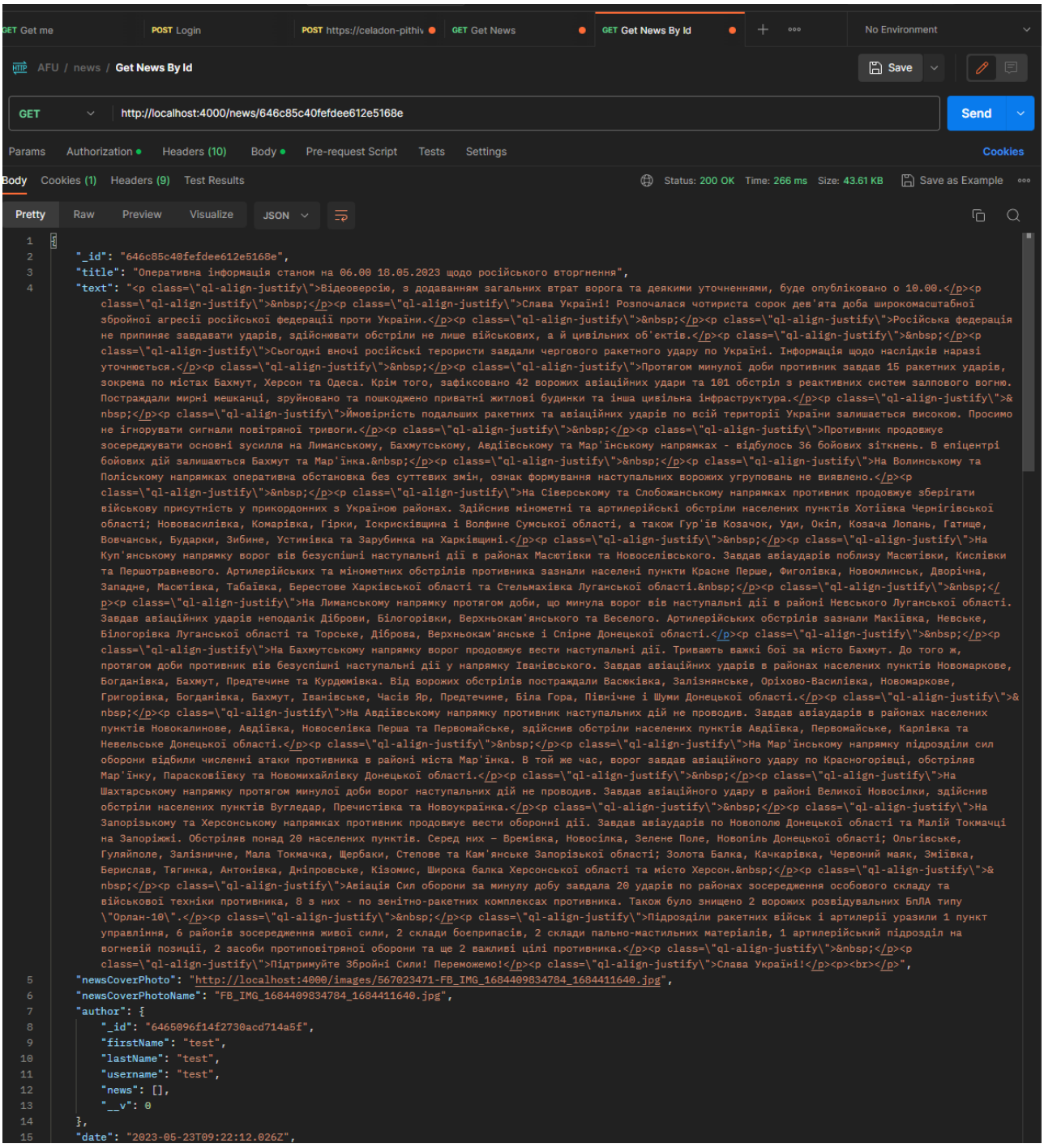

Рисунок 5.3 – Результат тестування отримання новини

В результаті ми отримали очікуваний результат, а саме ідентифікатор новини, заголовок новини, текст новини, обкладинку новини, назву обкладинки новини, інформацію про автора, дату створення новини, час прочитання новини та масив схожих новин.

5.2. Створення документації за допомогою Postman

Документація є важливою частиною розробки програмного забезпечення та інших технічних продуктів. Вона надає чітку інформацію про продукт, його функціональність, використання, конфігурацію та API, а також допомагає розробникам, користувачам та іншим зацікавленим сторонам зрозуміти, як працювати з продуктом.

Основна мета документації - надати чітку, повну і точну інформацію, щоб допомогти користувачам зрозуміти, як використовувати продукт і API, як його налагоджувати і як вирішувати проблеми, що можуть виникнути.

Документація може включати таку інформацію, як

Вступ: вступ: опис продукту, призначення, основні функції та переваги.

Встановлення та налаштування: інструкції щодо встановлення та налаштування продукту.

Використання: детальні інструкції щодо використання продукту, включно з процедурами, сценаріями та прикладами.

Документація API: якщо продукт має API, документація з описом доступних кінцевих точок, параметрів, типів даних, прикладів запитів і відповідей.

Конфігурація та налагодження: описує, як змінювати налаштування та конфігурацію продукту, включаючи базові та розширені налаштування.

Приклади: практичні приклади, що ілюструють різні сценарії використання продукту.

Відповіді на запитання та проблеми: відповіді на поширені запитання та інформація про вирішення проблем, з якими можуть зіткнутися користувачі.

Посилання на ресурси: посилання на інші ресурси, які можуть допомогти вам зрозуміти продукт.

На рисунку 5.4 наведено документацію веб-платформи

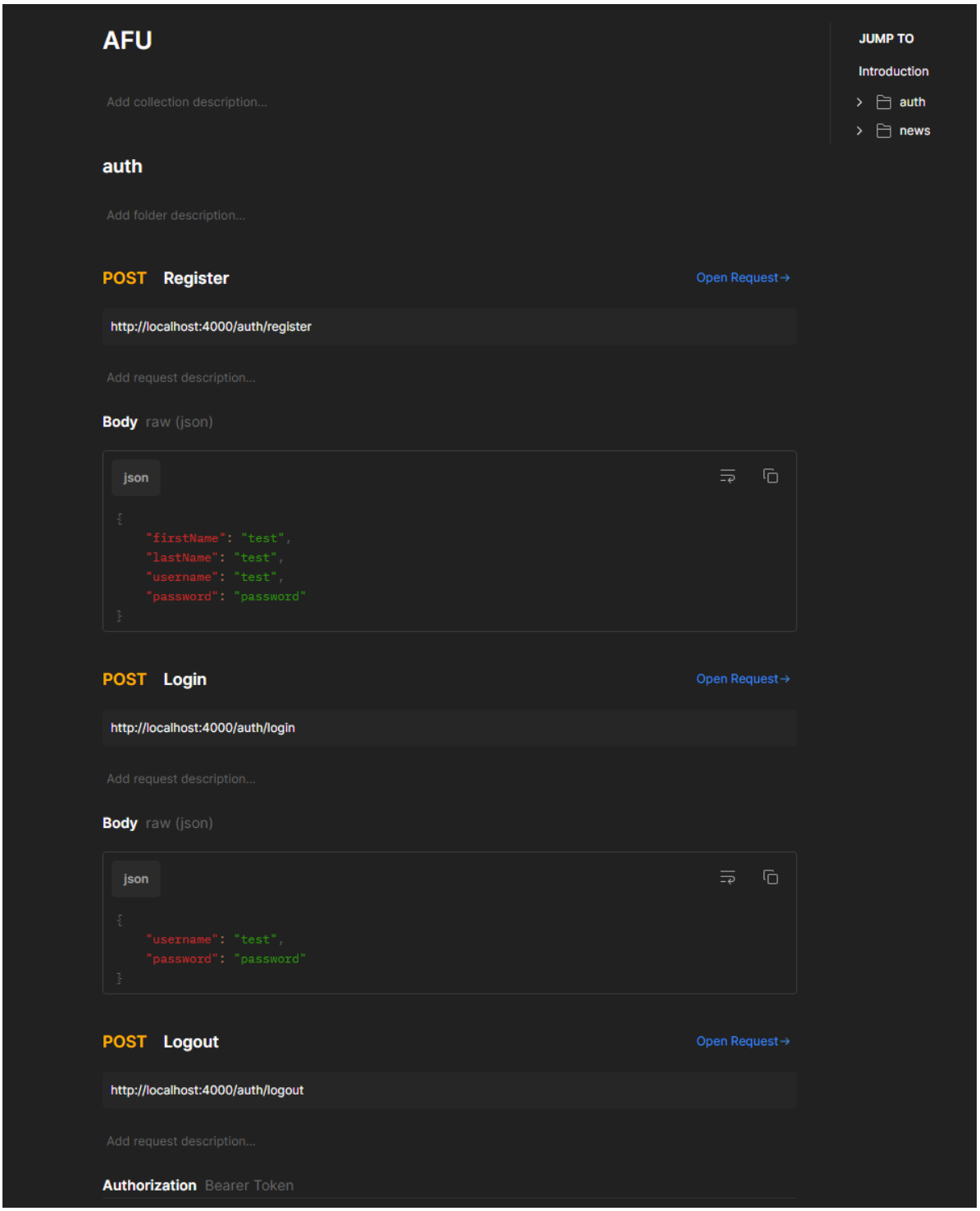

Рисунок 5.4 – Документація веб-платформи

ВИСНОВКИ

В даній дипломній роботі було розроблено веб-сайт для Збройних Сил України з використанням сучасних методів та архітектурних рішень. Для досягнення поставленої мети були сформульовані та вирішені наступні завдання

1. Аналіз сучасних підходів до архітектури веб-додатків та визначення найбільш ефективних та популярних методів розробки.
2. Проаналізовані сучасні шаблони архітектурного дизайну визначено найбільш підходящі для даного веб-сайту.
3. Розроблено веб-сайт для Збройних Сил України. Це включало в себе створення єдиної та передбачуваної навігації, адаптивності та забезпечення єдиного зовнішнього вигляду на всіх сторінках.
4. Веб-сайт Збройних сил України був розроблений з використанням новітніх фреймворків та інструментів для забезпечення швидкості, гнучкості та надійності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Brigadier Mick Ryan et al. Social Media in the Military: Opportunities, Perils and a Safe Middle Path. Grounded Curiosity (blog), 20 August 2016. URL: https://groundedcuriosity.com/social-media-in-the-military-opportunities-perils-and-a-safe-middle-path/#.ZBt-kHbP3IU (дата звернення 20.03.2023).
2. Introduction to client-side Frameworks - learn web development: MDN. URL: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing /Client-side_JavaScript_frameworks/Introduction (дата звернення 20.03.2023).
3. Kardys D. Modular Web Design: Designing With Components. URL: https://blog.wsol.com/modular-web-design-designing-with-components (дата звернення 20.03.2023).
4. Petitit F. The new Web application architectures and their impacts for enterprises – Part 1 URL: https://blog.octo.com/en/new-web-application-architectures-and-impacts-for-enterprises-1/ (дата звернення 20.03.2023).
5. Fauler M. Patterns of Enterprise Application Architecture. Pearson Education. 17-th edition, 2011. 520 p.
6. Dhaduk H. An Ultimate Guide of Web Application Architecture. URL: https ://www.simform. com / blog / web -application-architecture/ (дата звернення 20.03.2023).
7. Syromiatnikov A. A Journey Through the Land of Model-View-* Design Patterns, 2014.
8. Moskala M. MVC vs MVP vs MVVM vs MVI. URL: https://academy.realm.io/posts/mvc-vs-mvp-vs-mvvm-vs-mvi-mobilization-moskala/ (дата звернення 20.03.2023)
9. Vice R., Siddique M.S. MVVM Survival Guide for Enterprise Architecture in Silverlight and WPF. Packt Publishing. 2012.
10. Minar I. MVC vs MVVM vs MVP. What a controversial topic that many developers can spend hours and hours debating and arguing about. 2012. https://plus.google.com/+AngularJS/posts/aZNVhj355G2 ((дата звернення 20.03.2023)
11. Máriás Z., Molnár B. A Study on Formal Consistency Evaluation of Backend and Frontend Business Logic in a Modern Client-Server Application. 2010. URL:

https://www.researchgate.net/publication/343587318_A_Study_on_Formal_Consistency_Evaluation_of_Backend_and_Frontend_Business_Logic_in_a_Modern_Client-Server_Application (дата звернення 20.03.2023)

12. Garrett J. J. Ajax: A new approach to web applications. Adaptive Path, 2007

13. Progresseve Web Apps. URL: https://developers.google.com/web /progressive-web-apps. (дата звернення 20.03.2023)

14. Spiderwriting. Static vs Dynamic Website Design - SpiderWriting Web Design. URL: http://www.spiderwriting.co.uk/static-dynamic.php (дата звернення 20.03.2023)

15. Codeconquest.com. (2017). Static vs. Dynamic Websites. URL: http://www.codeconquest.com/website/static-vs-dynamic-websites/ (дата звернення 20.03.2023).

16. Best JavaScript Frameworks For 2023. URL: https://www.lambdatest.com/blog/best-javascript-frameworks/ (дата звернення 20.03.2023)

17. Sanctis V. ASP.NET Core 2 and Angular 5: full-stack web development with .NET Core and Angular. Birmingham, UK: Packt Publishing, 2017,

18. Mardan A. React quickly : painless web apps with React, JSX, Redux, and GraphQL. Shelter Island, NY: Manning Publications Co, 2017,

19. Hanchett E. Vue.js in action. Shelter Island, NY: Manning Publications Co, 2018,

20. Vue.js Guide. URL: https://vuejs.org/v2/guide/index.html (дата звернення 20.03.2023)

ДОДАТКИ

Додаток А. Структурна схема шаблону MVVM

Додаток Б. Лістинг програми

auth.controller.js

```
export const register = async newUser => {
  try {
    const { username, password, firstName, lastName } = newUser
    const candidate = await User.findOne({ username })
    if (candidate) {
      throw new Error(`User with username ${username} already exists`)
    }
    const hashedPassword = await bcrypt.hash(password, 12)
    const user = new User({ username, password: hashedPassword,
firstName, lastName })
    await user.save()

    const { accessToken, refreshToken } = generateTokens(user._id)

    await updateRefreshToken(user._id, refreshToken)
    return formatUserResponse(user, accessToken, refreshToken)
  } catch (e) {
    throw e
  }
}

export const login = async (username, password) => {
  try {
    const user = await User.findOne({ username }).select('+password')
    if (!user) {
      throw new Error(`User with username ${username} not found`)
    }

    const isPasswordCorrect = await bcrypt.compare(password,
user.password)
    if (!isPasswordCorrect) {
      throw new Error('Invalid credentials')
    }
    const { accessToken, refreshToken } = generateTokens(user._id)
    await updateRefreshToken(user._id, refreshToken)
    return formatUserResponse(user, accessToken, refreshToken)
  } catch (e) {
    throw e
  }
}

export const auth = async id => {
  try {
    const user = await User.findById(id)
    if (!user) {
      throw new Error(`User with id ${id} not found`)
    }
    const { accessToken, refreshToken } = generateTokens(user._id)

    await updateRefreshToken(user._id, refreshToken)
```

```
    return formatUserResponse(user, accessToken, refreshToken)
  } catch (e) {
    throw e
  }
}

export const refreshTokens = async (userId, refreshToken) => {
  try {
    const user = await User.findById(userId).select('+hashedRt')

    if (!user || !user.hashedRt) {
      throw new Error(`Access denied`)
    }

    const isRefreshTokenCorrect = await bcrypt.compare(refreshToken,
user.hashedRt)

    if (!isRefreshTokenCorrect) {
      throw new Error(`Access denied`)
    }

    await user.save()

    const { accessToken, refreshToken: newRefreshToken } =
generateTokens(user._id)
    await updateRefreshToken(user._id, newRefreshToken)

    return formatUserResponse(user, accessToken, newRefreshToken)
  } catch (e) {
    throw e
  }
}

export const logout = async userId => {
  try {
    const user = await User.findById(userId)
    if (!user) {
      throw new Error(`User with id ${userId} not found`)
    }
    user.hashedRt = null
    await user.save()
  } catch (e) {
    throw e
  }
}

const generateTokens = id => {
  const accessToken = jwt.sign({ id }, process.env.ACCESS_SECRET, {
    expiresIn: AT_AGE,
  })
  const refreshToken = jwt.sign({ id }, process.env.REFRESH_SECRET, {
    expiresIn: RT_AGE,
  })
  return { accessToken, refreshToken }
```

```
}

const updateRefreshToken = async (userId, refreshToken) => {
  try {
    const hashedRt = await bcrypt.hash(refreshToken, 12)
    const user = await User.findById(userId)

    if (!user) {
      throw new Error(`User with id ${userId} not found`)
    }

    user.hashedRt = hashedRt

    await user.save()
  } catch (e) {
    throw e
  }
}

const formatUserResponse = (user, accessToken, refreshToken) => {
  return {
    user: {
      _id: user._id,
      firstName: user.firstName,
      lastName: user.lastName,
      username: user.username,
      news: user.news,
    },
    accessToken,
    refreshToken,
  }
}
```

news.controller.js

```
export const createNews = (news, userId) => {
  try {
    const newNews = new News({ ...news, author: userId })
    return newNews.save()
  } catch (e) {
    throw e
  }
}

export const getNews = async (limit = 10, page) => {
  try {
    const count = await News.countDocuments()
    const lastPage = Math.ceil(count / limit)
    const skip = limit * (page - 1)

    const news = await News.find().select('title _id date
newsCoverPhoto').limit(limit).skip(skip).sort({ date: -1 })

    return {
```

```
      news,
      lastPage,
    }
  } catch (e) {
    throw e
  }
}

export const getListOfNews = async (limit, skip) => {
  try {
    const news = await News.find().select('title _id date
newsCoverPhoto').limit(limit).skip(skip).sort({ date: -1 })

    return news
  } catch (e) {
    throw e
  }
}

export const getNewsById = async id => {
  try {
    const news = await News.findById(id).populate('author')

    const formattedNews = news.toObject()

    const wordCount = formattedNews.text.split(' ').length
    const readingTime = Math.ceil(wordCount / 200)

    const titleWords = news.title.split(' ').filter(word => word.length
> 2) // Split title into words and filter out short words
    const titleRegex = titleWords.map(word => new
RegExp(`\\b${word}\\b`, 'i')) // Create regex patterns for individual
words

    const similarNews = await News.find({ _id: { $ne: id }, title: {
$in: titleRegex } })
      .select('title _id date newsCoverPhoto text')
      .limit(3)

    return { ...formattedNews, readingTime, similarNews }
  } catch (e) {
    throw e
  }
}

export const updateNews = async (id, news) => {
  try {
    const updatedNews = await News.findByIdAndUpdate(id, news, {
      new: true,
    }).populate('author')
    return updatedNews
  } catch (e) {
    throw e
  }
}
```

```
export const deleteNews = async id => {
  try {
    const deletedNews = await News.findByIdAndDelete(id)
    return deletedNews
  } catch (e) {
    throw e
  }
}
```

User.schema.js

```
const User = new Schema({
  firstName: { type: String, required: true },
  lastName: { type: String, required: true },
  username: { type: String, required: true },
  password: { type: String, required: true, select: false },
  hashedRt: { type: String, select: false },
  news: [{ type: Schema.Types.ObjectId, ref: 'File' }],
})
```

news.schema.js

```
const News = new Schema({
  title: { type: String, required: true },
  text: { type: String, required: true },
  newsCoverPhoto: { type: String, required: false },
  newsCoverPhotoName: { type: String, required: false },
  date: { type: Date, default: Date.now },
  author: { type: Schema.Types.ObjectId, ref: 'User', required: true },
})
```

news.router.js

```
router.get('/:limit/:page', [check('limit', 'limit is
required').exists(), check('page', 'page is required').exists()], async
(req, res) => {
  try {
    const errors = validationResult(req)
    if (!errors.isEmpty()) {
      return res.status(422).json({ errors: errors.array() })
    }
    const { limit, page } = req.params
    const news = await getNews(limit, page)
    res.json(news)
  } catch (e) {
    console.log(e)
    res.status(500).json({ message: e.message })
  }
})
```

```
router.get('/:id', [check('id', 'id is required').exists()], async (req,
res) => {
  try {
    const errors = validationResult(req)
    if (!errors.isEmpty()) {
      return res.status(422).json({ errors: errors.array() })
    }
    const { id } = req.params
    const news = await getNewsById(id)
    res.json(news)
  } catch (e) {
    console.log(e)
    res.status(500).json({ message: e.message })
  }
})

router.get('/list/:limit/:skip', [check('limit', 'limit is
required').exists(), check('skip', 'skip is required').exists()], async
(req, res) => {
  try {
    const errors = validationResult(req)
    if (!errors.isEmpty()) {
      return res.status(422).json({ errors: errors.array() })
    }

    const { limit, skip } = req.params
    const news = await getListOfNews(limit, skip)
    res.json(news)
  } catch (e) {
    console.log(e)
    res.status(500).json({ message: e.message })
  }
})

router.post('/', [check('title', 'Title is required').exists(),
check('text', 'Text is required').exists(), authMiddleware], async (req,
res) => {
  try {
    const errors = validationResult(req)
    if (!errors.isEmpty()) {
      return res.status(422).json({ errors: errors.array() })
    }
    const news = await createNews(req.body, req.user.id)
    res.json(news)
  } catch (e) {
    console.log(e)
    res.status(500).json({ message: e.message })
  }
})

router.put('/:id', [check('id', 'id is required').exists()], async (req,
res) => {
  try {
    const errors = validationResult(req)
    if (!errors.isEmpty()) {
```

```
      return res.status(422).json({ errors: errors.array() })
    }
    const { id } = req.params
    const news = await updateNews(id, req.body)
    res.json(news)
  } catch (e) {
    console.log(e)
    res.status(500).json({ message: e.message })
  }
})

router.delete('/:id', [check('id', 'id is required').exists()], async
(req, res) => {
  try {
    const errors = validationResult(req)
    if (!errors.isEmpty()) {
      return res.status(422).json({ errors: errors.array() })
    }
    const { id } = req.params
    const news = await deleteNews(id)
    res.json(news)
  } catch (e) {
    console.log(e)
    res.status(500).json({ message: e.message })
  }
})

router.post('/upload', upload.single('file'), async (req, res) => {
  try {
    const fileUrl = `http://localhost:4000/images/${req.file.filename}`
    res.json({ fileUrl })
  } catch (e) {
    console.log(e)
    res.status(500).json({ message: e.message })
  }
})
```

auth.router.js

```
router.post(
  '/register',
  [
    check('username', 'Username must be at least 3 characters long and
less than 20').isLength({
      min: 3,
      max: 20,
    }),
    check('password', 'Password must be at least 6 characters long and
less than 40').isLength({
      min: 6,
      max: 40,
    }),
    check('firstName', 'First name is required').exists(),
```

```
    check('lastName', 'Last name is required').exists(),
  ],
  async (req, res) => {
    try {
      const errors = validationResult(req)
      if (!errors.isEmpty()) {
        return res.status(422).json({ errors: errors.array() })
      }
      const { user, accessToken, refreshToken } = await
register(req.body)
      res.cookie('refreshToken', refreshToken, {
        maxAge: REFRESH_COOKIE_AGE,
        httpOnly: true,
      })
      res.json({
        user,
        accessToken,
      })
    } catch (e) {
      console.log(e)
      res.status(500).json({ message: e.message })
    }
  }
)

router.post(
  '/login',
  [
    check('username', 'Username must be at least 3 characters
long').isLength({
      min: 3,
    }),
    check('password', 'Password must be at least 6 characters
long').isLength({
      min: 6,
    }),
  ],
  async (req, res) => {
    try {
      const errors = validationResult(req)
      if (!errors.isEmpty()) {
        return res.status(422).json({ errors: errors.array() })
      }
      const { username, password } = req.body
      const { user, accessToken, refreshToken } = await login(username,
password)
      res.cookie('refreshToken', refreshToken, {
        maxAge: REFRESH_COOKIE_AGE,
        httpOnly: true,
      })
      res.json({
        user,
        accessToken,
      })
    } catch (e) {
```

```
      console.log(e)
      res.status(500).json({ message: e.message })
    }
  }
)

router.get('/me', authMiddleware, async (req, res) => {
  try {
    const { user, accessToken } = await auth(req.user.id)

    res.json({
      user,
      accessToken,
    })
  } catch (e) {
    console.log(e)
    res.status(500).json({ message: e.message })
  }
})

router.post('/refresh', authMiddleware, async (req, res) => {
  try {
    const { refreshToken: rt } = req.cookies
    const { user, accessToken, refreshToken } = await
refreshTokens(req.user.id, rt)
    res.cookie('refreshToken', refreshToken, {
      maxAge: REFRESH_COOKIE_AGE,
      httpOnly: true,
    })

    res.json({
      user,
      accessToken,
    })
  } catch (e) {
    console.log(e)
    res.status(500).json({ message: e.message })
  }
})

router.post('/logout', authMiddleware, async (req, res) => {
  try {
    await logout(req.user.id)
    res.clearCookie('refreshToken')
    res.json({ message: 'Logged out' })
  } catch (e) {
    console.log(e)
    res.status(500).json({ message: e.message })
  }
})
```