МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота бакалавра

ВЕБ-ДОДАТОК ДЛЯ ВЕДЕННЯ НОТАТОК

Здобувач освіти гр. ІН – 83 Владислав МАТВІЄНКО

Науковий керівник,
кандидат технічних наук,
асистент кафедри комп'ютерних наук Артем КОРОБОВ

Завідувач кафедри
доктор технічних наук, професор Анатолій ДОВБИШ

СУМИ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую_____
Зав. кафедрою Довбиш А.С.
"_____" _____2022 р.

ЗАВДАННЯ
до кваліфікаційної роботи

здобувача вищої освіти четвертого курсу, групи ІН-83 спеціальності «122 – Комп'ютерні науки» денної форми навчання Матвієнка Владислава Олександровича.

Тема: «ВЕБ-ДОДАТОК ДЛЯ ВЕДЕННЯ НОТАТОК »

Затверджена наказом по СумДУ
№_____ від _____2022 р.

Зміст пояснювальної записки: 1) літературний огляд за обраною тематикою роботи; 2) вибір оптимальних інструментів для розробки односторінкового веб-додатку; 3) розробка клієнтської та серверної частин веб додатку; 4) аналіз та висновки роботи веб-додатку.

Дата видачі завдання «_____»_____ 2022 р.
Керівник роботи _____Артем КОРОБОВ
Завдання прийняв до виконання _____ Владислав МАТВІЄНКО

ЗМІСТ

РЕФЕРАТ

Записка: 60 стор., 31 рис., 3 таблиці, 4 додатки, 20 літературних джерел.

Об'єкт дослідження — процес розробки односторінкового веб додатку.

Мета роботи — розробка SPA веб платформи для ефективної організації та візуалізації щоденних нотаток.

Методи дослідження — в основу методології дослідження покладено методи сучасного стеку технологій для створення односторінкових веб додатків та виконання HTTP запитів між клієнтом та сервером.

Результати — проведено аналіз літератури, інструментів та методів, які дозволяють розробити односторінковий веб-додаток, здійснювати обмін даними між клієнтом та сервером, проводити автентифікацію користувачів, створювати та зберігати замітки. Після ознайомлення з існуючими рішеннями було розроблено алгоритм програми та її реалізацію у вигляді сучасного односторінкового веб додатку, який дозволяє створювати, редагувати та зберігати замітки зареєстрованим користувачам з будь якого пристрою. Розроблений веб-додаток було реалізовано за допомогою мови програмування JavaScript, а саме для серверної частини використана платформа Node та фреймворк Express, для клієнтської – фреймворк React, для бази даних - MongoDB.

SPA, REST API, MONGODB, EXPRESS, REACT, NODE

ВСТУП

У сучасному світі просто неможливо уявити своє життя без інформаційних технологій, незважаючи на те, що в недалекому минулому людина і гадки не мала про них. У наше життя вони увійшли дуже міцно, застосовуються інформаційні технології у всіх сферах життя людиги, виконуючи особливо значущу роль. Інформаційні технології представляють весь накопичений досвід людства у форматизованому вигляді, придатному для практичного використання [1]. І в ньому сконцентровані наукові знання та матеріалістичний досвід людей для здійснення суспільних процесів, при цьому заощаджуються витрати праці, часу, енергії та речових засобів. І з кожним днем, все більше і більше ця роль нарощується [1].

У кожного з нас є проблеми з пам'яттю — можна піти за покупками в магазин і забути купити якусь річ або забути геть про те, що завтра потрібно здати дизайн головного меню. Звичайно, можна все записувати в блокноті але тягяти завжди його з собою та мати ще при собі ручку, це не дуже зручно, а можна просто знайти додаток для телефону або комп’ютера. Та й взагалі у таких додатків є дуже багато переваг перед простими паперовими блокнотами. Можна красиво оформлювати замітки для кращого розуміння написаного, для цього є багато мов розмітки тексту, замітка може мати теги, за якими вони можуть відноситися до однієї групи, за допомогою пошуку в додатках потрібна замітка шукається досить швидко, в звичайному блокноті це б зайняло більше часу.

Якщо ви школяр чи студент, то ваше завдання — якомога краще розібратися у матеріалі уроку або лекції, щоб у майбутньому використати отримані знання. Вони знадобляться вам як мінімум для того, щоб скласти іспит або виконати практичне завдання.

Метою роботи є вирішення вище зазначених проблем, тобто створення веб-додатку для введення та зберігання заміток.

1. ЛІТЕРАТУРНИЙ ОГЛЯД ЗА ОБРАНОЮ ТЕМАТИКОЮ РОБОТИ

1.1 Аналіз принципів побудови SPA додатків

SPA скорочення від "Single Page Application" перекладається як "односторінковий додаток", тобто, весь вивід і всі інші операції відбуваються в одному HTML документі, як правило для отримання даних від сервера використовуються AJAX запити [2].

Весь рендеринг відбувається за рахунок динамічного HTML, CSS і JavaScript, зазвичай залежно від URL у браузері і відбувається рендеринг потрібного користувачу контенту та компонентів, наприклад, кнопка авторизація користувача, якщо користувач авторизований, то її не буде, кнопку вихід те ж саме, але навпаки [2].

Ще дуже цікава особливість у тому, що такий тип WEB додатків не можна розрізнити від нативних, єдине, що все працює в браузері. На рисунку 1.1 зображено архітектуру SPA додатку.

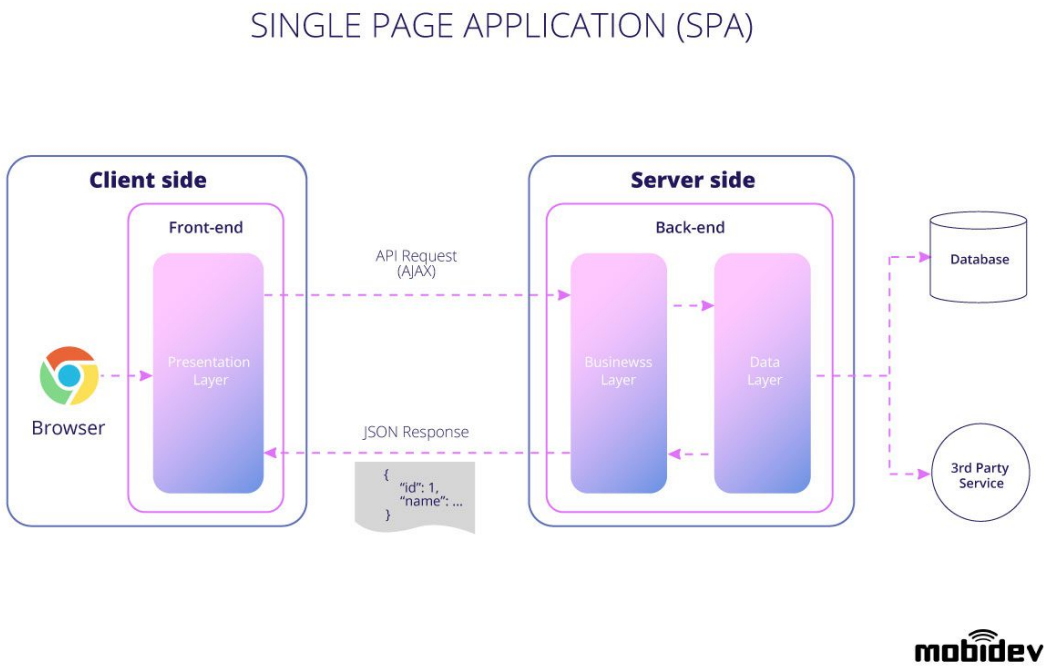

Рисунок 1.1 – Архітектура SPA додатку

Якщо програма досить складна і містить багатий функціонал, як, наприклад, система електронного документообігу, кількість файлів зі скриптами може досягати декількох сотень, а то й тисяч. А завантаження всіх скриптів аж ніяк не означає, що при завантаженні сайту будуть завантажені відразу всі сотні та тисячі файлів зі скриптами. Для вирішення проблеми завантаження великої кількості скриптів у SPA покликаний API під назвою AMD. AMD реалізує можливість завантаження скриптів на вимогу користувача. Тобто, якщо для "головної сторінки" односторінкового додатку знадобилося 4 скрипти, вони будуть завантажені одразу перед запуском додатку. А якщо користувач натиснув на іншу сторінку односторінкового ресурсу, наприклад "Допомога", то принцип AMD завантажить скрипт та розміткау тільки перед тим як перейти на цю сторінку [2].

Сторінка сайту, на якому розміщені всі посилання на всі CSS, та посилання на скрипти, необхідні для роботи SPA, можна назвати “Web-сторінка”. Файл з такою сторінкою зазвичай називається "index.html" (в ASP.NET MVC може бути index.cshtml або index.vbhtml або навіть index.aspx) А сторінки, які перемикає користувач усередині односторінкового порталу, називаються “модулі” або “компоненти”.

1.2 Переваги та недоліки SPA додатків

До переваг Single Page Applications можна віднести:

– Доступність. Можна отримати моментальний доступ до будь-якого типу пристрою без проблем із сумісністю, обсягом пам'яті, потужностями та часом на встановлення.

– Універсальність. Використовувати програму можна практично з будь-якого пристрою, якщо на ньому є хоча би доступ до інтернету. Якщо при розробці інтерфейсу враховувалися різні розміри екрану, то використовувати SPA однаково зручно як і з ПК та планшета, так і зі смартфона.

– Можливість використовувати доволі великі обсяги даних. Розмір програми та даних, що використовуються, не обмежений пам'яттю пристрою.

– Швидкість. Одна сторінка з усім необхідним не тільки заощаджує час на повторне завантаження потрібних даних, а й підвищує продуктивність роботи в декілька разів.

– Можливості розробки. Розробникам доступно дуже багатий вибір фреймворків, які спрощують створення архітектури проекту та надають чимало вже готових елементів для побудови додатка.

Приклади зручних та корисних односторінкових програм - Gmail і Google Translate. Багато людей використовує їх, і навряд чи в кого виникає бажання перейти на їх десктопні аналоги.

До недоліків:

– Необхідність інтернет-з'єднання. Без доступу до мережі використовувати цей додаток неможливо. Але, частіше за все, програмне забезпечення використовує в роботі зовнішні бази даних, то доступ до інтернету необхідний у будь-якому випадку.

– Проблеми з SEO. Особливості SPA ускладнюють або практично унеможливлюють процес індексації пошуковими системами всіх модулів програми. Це може спричинити труднощі з оптимізацією.

– Не працює у користувачів з вимкнутою підтримкою JS. Багато хто вимикає відображення JS-елементів у себе в браузерах, а Single Page Application використовує їх у своїй роботі, тому вона може просто не працювати.

1.3 Інструменти для розробки SPA додатків

Для розробки SPA додатків знадобиться односторінковий фреймворк веб-додатків — це веб-фреймворк, який забезпечує платформу для розробки веб-додатків SPA. Ці фреймворки SPA включають набір інструментів і бібліотек для уникнення повторення коду. Крім того, вони також підтримують

модульне тестування, автоматичне прив’язування даних, маршрутизацію по URL-адресам і маніпулювання тегами HTML [3].

Фреймворк односторінкового додатка інтегрується з великою кількістю утиліт та бібліотек, які спрощують процес розробки. Поряд з автоматизацією повторюваних завдань кодування та пропонуванням готових до використання компонентів, ці фреймворки також забезпечують підтримку HTML та AJAX, кешування даних, захист від вразливостей безпеки та підвищення продуктивності розробника.

Нижче наведено деякі популярні веб-фреймворки, які використовуються для розробки веб-додатків односторінкових програм.

1. EmberJS.
2. AngularJS.
3. ReactJS.
4. BackBoneJS.
5. Vue [3].

1.4 Огляд REST API

На найпростішому рівні API — це механізм, який дозволяє додатку чи службі отримати доступ до ресурсу в іншому додатку чи службі, через запити. Додаток або служба, яка здійснює доступ, називається клієнтом, а додаток або служба, що містить ресурс, називається сервером.

Деякі API, такі як SOAP або XML-RPC, накладають суворі вимоги для розробників. Але API REST можна розробляти практично з використанням будь-якої мови програмування та підтримують різноманітні формати даних.

API REST спілкуються через HTTP-запити для виконання стандартних функцій бази даних, таких як створення, читання, оновлення та видалення записів, відомих за загальною назвою CRUD, у ресурсі. Наприклад, REST API використовуватиме запит GET для отримання запису, запит POST для його

створення, запит PUT або PATCH для оновлення запису, а запит DELETE для його видалення. Усі методи HTTP можна використовувати у викликах API.

Стан ресурсу в будь-який конкретний момент або мітка часу відомий як представлення ресурсу. Цю інформацію можна надати клієнту практично в будь-якому виді, включаючи нотацію об’єктів JavaScript (JSON), HTML, XLT або звичайний текст. JSON наразі є найбільш популярний, доволі зрозумілий як для люди, так і машини, і він не залежить від конкретної мови програмування.

Заголовки та параметри запитів також дуже важливі у викликах REST API, оскільки вони містять важливу інформацію про ідентифікатори, таку як метадані, чи авторизований користувач, уніфіковані ідентифікатори ресурсів (URI), кешування, файли cookie тощо. Заголовки запитів і відповіді, а також звичайні коди статусу HTTP використовуються в добре розроблених API REST.

1.5 Постановка задачі

Метою роботи є розробка SPA додатку, який буде представляти веб додаток для ведення заміток.

В ході виконання роботи необхідно реалізувати наступні задачі:

1. виконати огляд інструментів для розробки та обрати оптимальні;
2. розробити інтуїтивно зрозумілий інтерфейс SPA додатка;
3. розробити базу даних для зберігання інформації;
4. розробити алгоритм HTTP-запитів;
5. виконати проектування архітектури SPA додатка реалізації створення та зберігання заміток;
6. розробити алгоритм роботи;
7. зробити відповідні висновки по роботі.

У разі успішної реалізації буде можливим створювати, редагувати та зберігати свої замітки в окремих каталогах, звертатися до них через теги.

2. МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

Для вирішення поставлених задач було взяті чотири основні інструменти розробки:

1. MongoDB - база даних документів.
2. Express(.js) – веб-фреймворк Node.js.
3. React(.js) - клієнтська платформа JavaScript.
4. Node(.js) – платформа, головний веб-сервер JavaScript

Всі ці інструменти складають стек MERN.

Архітектура MERN дозволяє легко побудувати 3-рівневу архітектуру (фронтенд, бекенд та база даних) повністю за допомогою JavaScript та JSON, зображено на рисунку 2.1. Тобто бекенд та фронтенд пишуться на одній мові програмуванні. І не потрібно для серверної частини використовувати інші мови, такі як, PHP та Java [4]. Це доволі полегшує розробку додатка.

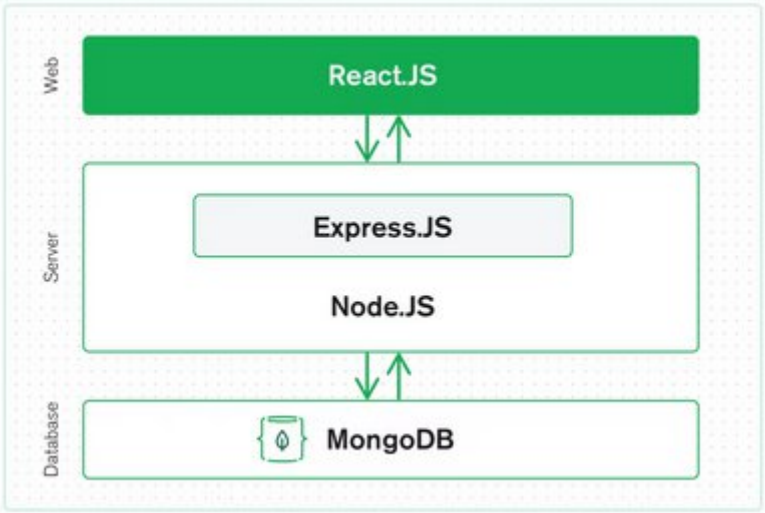

Рисунок 2.1 – 3-рівнева архітектура стеку MERN

Як і в будь-якому веб-стеку, можна створювати все, що забажає розробник, у MERN, хоча він ідеально підходить для випадків, коли є важкі JSON, є хмарними та мають динамічні веб-інтерфейси.

Прикладами можуть бути:

– керування робочим процесом;

– сайт з новинами;

– блоги;

– Todo програми;

– інтерактивні форуми / соціальні продукти [5];

Стек MERN чудово підходить для веб додатків, де користувач може вести свої замітки.

2.1 Фреймворк React JS

Верхній рівень стеку MERN — React.js, декларативна платформа JavaScript для створення динамічних програм на стороні клієнта в HTML. React дозволяє створювати складні клієнтські інтерфейси за допомогою доволі простих компонентів, підключати їх до даних на сервері та відображати їх у вигляді HTML [6].

Сильна сторона React — це обробка інтерфейсів, керованих даними, що керуються станом, з мінімальним кодом і мінімальними проблемами, і він має всі примхи, які очікуються від сучасного веб-фреймворку: чудова підтримка форм, обробки помилок, подій, списків тощо.

Сьогодні популярність React набагато перевершила популярність усіх інших фреймворків для розробки інтерфейсу [6].

React полегшує створення динамічних веб-додатків, оскільки вимагає менше кодування та пропонує більше функціональних можливостей, на відміну від ванільного JavaScript, де кодування часто стає складним доволі швидко.

React використовує Virtual DOM, тим самим створюючи веб-додатки швидше. Віртуальний DOM порівнює попередні стани компонентів і оновлює лише ті елементи в реальному DOM дереві, які були змінені, замість того, щоб знову оновлювати всі компоненти цілком, як це роблять звичайні веб-додатки.

Компоненти є будівельними блоками будь-якої програми React, і одна програма зазвичай складається з кількох компонентів. Ці компоненти мають

свою логіку та елементи керування, і їх можна повторно використовувати в додатку необмежену кількість разів, що, у свою чергу, значно скорочує затрати та час розробки програми [7].

React просувається за односпрямованим потоком даних. Це означає, що при розробці програми React розробники часто вкладають дочірні компоненти в батьківські. Оскільки дані надходять в одному напрямку, стає легше налагоджувати помилки та знати, де саме виникає проблема в програмі на даний момент [7].

React легко опанувати оскільки він переважно поєднує базові концепції HTML і JavaScript з деякими корисними доповненнями, як наприклад JSX. Проте, як і у випадку з іншими інструментами та фреймворками, доведеться витратити деякий час, щоб отримати відповідне розуміння бібліотеки [8].

Його можна використовувати як для розробки веб-додатків, так і для мобільних: відомо, що React використовується для розробки веб-додатків, але це ще не все, що він може зробити. Існує також фреймворк під назвою React Native, який дуже популярний і використовується для створення привабливих мобільних додатків. Таким чином, насправді React можна використовувати для створення як веб-, так і мобільних додатків [9].

Також Facebook випустив розширення, яке можна використовувати для налагодження програм React. Це робить процес налагодження веб-додатків React швидшим і простішим.

2.2 Фреймворк Express.js і Node.js

Наступний рівень нижче — це серверна платформа Express.js, яка працює всередині сервера Node.js. Express.js оголошує себе як «швидкий, носмислений, мінімалістський веб-фреймворк для Node.js», і це насправді саме те, чим він є. Express.js має потужні моделі для маршрутизації URL-адрес (узгодження вхідної URL-адреси з функцією сервера) та обробки HTTP запитів і відповідей [10].

Здійснюючи запити XML HTTP (XHR) або GET або POST із інтерфейсу React.js, можна підключитися до функцій Express.js, які забезпечують роботу певної програми. Ці функції, у свою чергу, використовують драйвери MongoDB Node.js, або через зворотні виклики з використанням Promises, для доступу та оновлення даних у базі даних MongoDB [11].

Node (або як більш формально Node.js) — це кросплатформне середовище виконання з відкритим вихідним кодом, яке дозволяє розробникам створювати всі види серверних інструментів і програм на JavaScript. Середовище виконання призначене для використання поза контекстом браузера, на движку JavaScript V8 (тобто для запуску безпосередньо на комп'ютері чи на віддаленому сервері) [12]. Таким чином, середовище опускає специфічні для браузера API JavaScript і додає підтримку більш традиційних API ОС, включаючи HTTP і бібліотеки файлової системи [13].

2.3 База даних MongoDB

Якщо певна програма зберігає будь-які дані (профілі користувачів, вміст, коментарі, завантаження, події, замітки тощо), знадобиться база даних, з якою так само легко працювати, як з React, Express і Node.

Ось тут і найбільш підходить MongoDB: документи JSON, створені у інтерфейсі React.js, можуть бути відправлені на сервер Express.js, де вони можуть бути оброблені та (за умови, що вони дійсні) збережені безпосередньо в MongoDB для подальшого використання.

Кожна база даних містить колекції, які, у свою чергу, містять документи. Кожен документ може відрізнятися різною кількістю полів. Розмір і зміст кожного документа можуть відрізнятися один від одного [14].

Структура документа більше відповідає тому, як розробники будують свої класи та об'єкти на відповідних мовах програмування. Розробники часто говорять, що їхні класи не є рядками та стовпцями, а мають чітку структуру з парами ключ-значення.

Рядки (або документи, які називаються в MongoDB) не повинні мати заздалегідь визначену схему зі всіма полями. Натомість поля можна створювати вже в подальшому під час розробки додатку.

Модель даних, доступна в MongoDB, дозволяє легше представляти ієрархічні зв'язки, зберігати масиви та інші доволі складні структури.

Середовища MongoDB дуже масштабовані. Компанії по всьому світу використовують кластери, деякі з яких працюють із понад сотні вузлами з приблизно мільйонами документів у базі даних [15].

2.4 Проектування бази даних

Даний веб-додаток буде містити три колекції (таблиці) в базі даних MongoDB:

1. користувачі (див. табл. 2.1);
2. каталоги (див. табл. 2.2);
3. замітки (див. табл. 2.3).

Таблиця 2.1 – Колекція для користувачів

Назва документа(рядка)	Опис документа	Тип документа	Unique	Required
_id	Ідентифікайний номер користувача	ObjectId	True	True
email	Електронна почта користувача	String	True	True
password	Пароль користувача	String		True
name	Ім'я користувача	String		True

Продовження таблиці 2.1

1	2	3	4	5
status	Статус користувача (підтвердив користувач почту чи ні)	String		
confirmationCode	Код для підтвердження почти та скидання паролю	String	True	
plan	План користувача (free/basic/pro)	String		True
image	Назва аватару користувача для пошуку на сервері.	String		

Таблиця 2.2 – Колекція для каталогів

Назва документа(рядка)	Опис документа	Тип документа	Unique	Required
_id	Ідентифікайний номер каталога	ObjectId	True	True
name	Ім’я катлога	String		True
user	Id користувача якому належить даний каталог	Ref ‘users’		

Таблиця 2.3 – Колекція для заміток

Назва документа(рядка)	Опис документа	Тип документа	Unique	Required
_id	Ідентифікайний номер замітки	ObjectId	True	True
title	Заголовок замітки	String		True
text	Текст замітки	String		
createDate	Дата створення замітки	Date		True
updateDate	Дата оновлення замітки	Date		True
tag	Теги замітки	Array		
favorite	Замітка у важливих	Boolean		
color	Колір замітки	String		
user	Id користувача якому належить дана замітка	Ref ‘users’		True
folder	Id каталога якому належить дана замітка	Ref ‘folders’		True

2.4 Спосіб аутентифікації користувача

Аутентифікація — це процес ідентифікації користувачів, які хочуть отримати доступ до системи, мережі, сервера, додатка, веб-сайту тощо [16]. Основна мета аутентифікації – переконатися, що користувач є тим, за кого себе видає. Неавторизованим користувачам заборонено доступ до конфіденційних даних сервісу, тому це вирішується за допомогою аутентифікації користувача. Аутентифікація покращує безпеку, дозволяючи будь-якому адміністратору даної організації керувати ідентифікацією та доступом окремого користувача. Основна аутентифікація, яка використовується для перевірки ідентифікації та контролю доступу користувача, — це логін або електронна пошта та пароль.

Найбільш поширеним є такі методи аутентифікації:

1. Вхід на основі пароля. Потрібно лише ввести комбінацію свого імені користувача, номера мобільного телефону або електронну пошту та пароля під час використання методу аутентифікації. Якщо ці дані збігаються з тими які існують в базі даних, то користувача автентифікують. Однак, даний метод аутентифікації є доволі слабкий для безпеки користувача.

2. Багатофакторна автентифікація. Метод аутентифікації, при якому користувач повинен передати ще декілька інших параметрів, щоб отримати доступ до служби або мережі. Наприклад, користувач повинен надати другий параметр у вигляді одноразового коду, який повинен прийти на телефон або електронною поштою на додаток до свого електронного адресу та пароля.

3. Аутентифікація на основі токенів. Аутентифікація на основі токенів дозволяє користувачам вводити свої дані лише один раз і натомість отримати єдиний у своєму роді зашифрований рядок (токен). Після цього користувачу не доведеться вводити свої облікові дані кожен раз, коли він хочете ввійти або отримати доступ до додатку. Токен гарантує, що доступ до додатку вже наданий. Також розробник може встановити термін дії токену, після закінчення якого користувач повинен знову надати свої дані. Більшість

випадків використання, наприклад Restful API, вимагають аутентифікації за допомогою токенів.

В проекті аутентифікація користувачів буде здійснюватися за допомогою токенів, а саме використання JWT (JSON Web Tokens).

JWT відрізняються від інших веб-токенів тим, що містять набір певних вимог. Вимоги використовуються для передачі інформації між двома сторонами. Вимоги, залежать від конкретного випадку використання. Наприклад, вимога може стверджувати, хто видав токен, кому, час видачі, який в нього термін дії або які дозволи надано власнику даного токена.

Поширеним способом використання JWT є маркери носія OAuth. У цьому прикладі сервер авторизації створює JWT на запит клієнта, точніше під час реєстрації або входу до веб ресурсу, зберігає в базі даних, або в куках, чи сесії, і підписує його, щоб він не міг бути змінений іншою стороною. Потім користувач надішле цей JWT зі своїм запитом до REST API. REST API кожен раз буде перевіряти, що підпис JWT відповідає збереженому, його корисному навантаженню та заголовку, а також що термін дії ще триває, щоб визначити, що JWT є наразі дійсний. Коли REST API перевірив JWT, він може використовувати певні вимоги, щоб задовольнити або відхилити запит клієнта.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Інформаційна модель

Структура веб-додатку складається з наступних елементів: React, Node.js, Express.js, MongoDB. React виступає в ролі фронтенд частини на боці клієнта, Node.js – на бекенді, для запуску JavaScript коду, Express.js потрібен для створення API серверу, MongoDB виступає в ролі бази даних, зберігає потрібну інформацію користувачів.

На рисунку 3.1 більш детально показано зв'язок даних елементів.

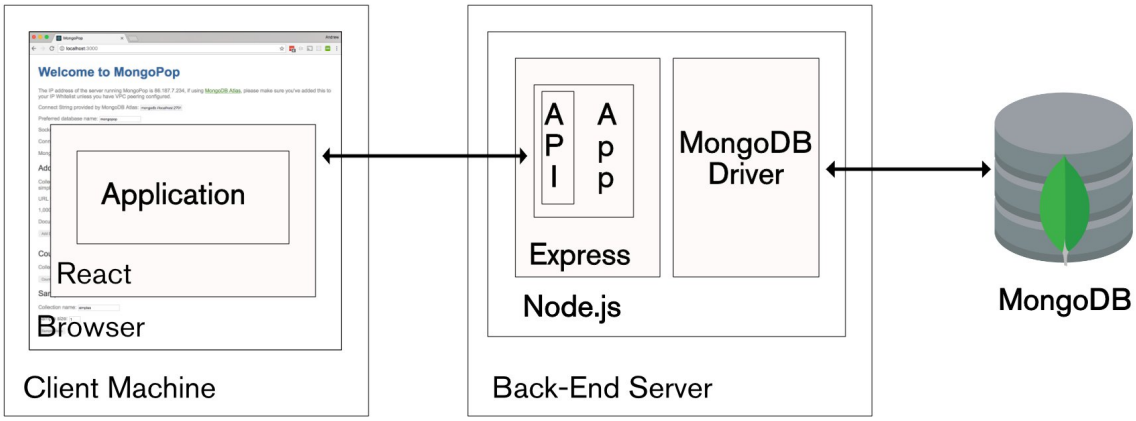

Рисунок 3.1 – Схема взаємодії компонентів системи

Для зв’язування бази даних MongoDB та фреємворка Express.js використовується MongoDB Driver, а точніше бібліотека Mongoose, тобто реалізує підключення до певної бази даних та виконує різні дії з моделями, на кшталт створення, редагування колекцій в моделі та видалення.

База даних складається з таких моделів:

1) Users.

2) Folders.

3) Notes.

3.2 Реалізація серверу та моделів

Для створення та функціонування сервера потрібно лише ввести потрібні команди та методи (див. дод. А).

Далі, перш за все, потрібно встановити з'єднання з базою данних та створити відповідні моделі(сутності). На рисунку 3.2 зображено лістинг коду для підключення до хмарної бази даних MongoDB, за допомогою бібліотеки Mongoose (див. дод. Б).

```
if (process.env.NODE_ENV === 'production' || process.env.NODE_ENV === 'development') {
mongoose.connect(keys.mongoURI)
  .then(() => console.log("MongoDb connect"))
  .catch(error => console.log(error))
}
```

Рисунок 3.2 – Підключення до бази даних MongoDB

Також за допомогою бібліотеки Mongoose створено схеми для моделів бази даних.

Для користувачів:

```
var mongoose = require('mongoose');

var Schema = mongoose.Schema;

const userSchema = new Schema({
    email: {
        type: String,
        required: true,
        unique: true,
    },
    password: {
        type: String,
        required: true,
```

```
    },
    name: {
        type: String,
        required: true,
    },
    status: {
        type: String,
        enum: ['Pending', 'Active'],
        default: 'Pending'
    },
    confirmationCode: {
        type: String,
        unique: true
    },
    plan: {
        type: String,
        required: true,
        default: 'FREE'
    },
    image:{
        type: String,
    }
})

module.exports = mongoose.model('users', userSchema)
```

Для каталогів:

```
var mongoose = require('mongoose');

var Schema = mongoose.Schema;
```

```
const foldersSchema = new Schema({
    name: {
        type: String,
        required: true,
    },
    user: {
        ref: 'users',
        type: Schema.Types.ObjectId
    }
})

module.exports = mongoose.model('folders', foldersSchema)
```

Для заміток:

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

const notesSchema = new Schema({
    title: {
        type: String,
        required: true,
    },
    text: {
        type: String
    },
    createDate: {
        type: Date,
        default: new Date(),
    },
```

```
    updateDate: {
        type: Date,
        default: new Date(),
    },
    tag: {
        type: Array,
    },
    favorite:{
        type: Boolean,
        default: false,
    },
    color:{
        type: String,
        default: '#a8becc',
    },
    user: {
        ref: 'users',
        type: Schema.Types.ObjectId,
        required: true
    },
    folder: {
        ref: 'folders',
        type: Schema.Types.ObjectId,
        required: true
    }
})
module.exports = mongoose.model('notes', notesSchema)
```

В даних схемах описані типи колекцій моделів, обов’язкові вони до заповнення чи ні, значення по умовчуванню, чи повинні значення бути унікальними та їх зв’язки з іншими схемами.

Модель замітки з’єднана з моделями користувачі та каталоги. Вона містить посилання на їх ObjectId.

3.3 Реалізація REST API

Далі потрібно реалізувати REST API, для HTTP-запитів клієнта на сервер [17]. На рисунку 3.3 зображено основні URI – універсальні ідентифікатори ресурсу.

```
app.use('/api/pages', pagesRouter);
app.use('/api/folders', foldersRouter);
app.use('/api/auth', authRouter);
app.use('/api/profile', profileRouter);
```

Рисунок 3.3 – Основні URI

Запити реалізовані на аутентифікацію користувача, роботу з каталогами та замітками, та на роботу з профілем користувача.

На рисунку 3.4 зображено реалізація методів HTTP-запитів до каталогів.

```
router.get('/', passport.authenticate('jwt', { session: false }), ctrlFolders.getAll);
router.get('/:id', passport.authenticate('jwt', { session: false }), ctrlFolders.getById);
router.delete('/:id', passport.authenticate('jwt', { session: false }), ctrlFolders.remove);
router.post('/', passport.authenticate('jwt', { session: false }), ctrlFolders.create);
router.patch('/:id', passport.authenticate('jwt', { session: false }), ctrlFolders.update);
```

Рисунок 3.4– Методи HTTP-запитів до каталогів

Методи GET потрібні для отримання інформації про ресурс, в даному випадку про каталог. Метод POST для створення нового ресурсу, в даному випадку каталогу. Метод DELETE для видалення ресурсу (каталогу). Метод PATCH для редагування ресурсу (каталогу) [18].

Дані функції містять посилання на запит, middleware, в даному випадку він потрібен для того щоб переконатися що користувач ввійшов в систему, інакше запит не буде виконаний, більш детально в наступному розділі, та контроллери, які реалізують весь функціонал запитів.

Для каталогів це контролери для надання користувачу даних про всі існуючи його каталоги, для одного конкретного, для створення та видалення каталогів (див. дод. В).

API запити, в разі успіху, повертають користувачеві код статусу та потрібну інформацію, в даному випадку про каталоги, в JSON форматі, якщо щось пішло не так, то повертають код статусу помилки та коротку інформацію.

На рисунку 3.5 зображено лістинг коду відповіді запиту під час виникнення помилки на сервері.

```
module.exports = (res, error) => {
    res.status(500).json({
        success: false,
        message: error.message ? error.message : error
    })
}      vlad, 4 months ago • First commit …
```

Рисунок 3.5 – Помилка з кодом статусу 500

Інші помилки з різними статусами викликаються безпосередньо в контролерах, вони можуть буди зв'язані, наприклад, з валідацією даних.

3.4 Реалізація аутентифікації користувача

Для аутентифікації користувачів використовується Passport.js. Це доволі популярне проміжне програмне забезпечення для аутентифікації користувачів для додатків Node.js. З його допомогою аутентифікацію можна інтегрувати в будь-яку програму написаних на базі Node і Express. Бібліотека Passport має дуже багато механізмів автентифікації, включаючи JWT, на основі Google,

Facebook, Github та просту автентифікацію за допомогою імені користувача та пароля [19].

Для даної програми буде використано метод аутентифікації за допомогою JWT токену.

На рисунку 3.6 зображено middleware, який зчитує JWT із заголовка авторизації HTTP зі схемою «Bearer».

```
const keys = require('../config/keys')
const mongoose = require('mongoose')
const User = mongoose.model('users')

var JwtStrategy = require('passport-jwt').Strategy,
    ExtractJwt = require('passport-jwt').ExtractJwt;
var opts = {}
opts.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken();
opts.secretOrKey = keys.jwt;

module.exports = passport => {
    passport.use(
        new JwtStrategy(opts, async (jwt_payload, done) => {
            try {
                const user = await User.findById(jwt_payload.userId).select('email id')

                if (user) {
                    done(null, user)
                } else {
                    done(null, false)
                }
            } catch (error) {
                console.log(error);
            }

        })
    )
};
```

Рисунок 3.6 – Лістинг коду для зчитування JWT із заголовка авторизації HTTP

Під час авторизації користувача буде створений JWT, зображено на рисунку 3.7, який кожен раз буде передаватися у відповідні HTTP запити. Даний middleware буде зчитувати токен, та надавати доступ до успішного виконання запиту. Щоб даний токен перевірявся потрібно добавити middleware в шлях запиту, як це зображено на рисунку 3.8.

```
const generateToken = (email, userId) => {
    return jwt.sign({
        email: email,
        userId: userId
    }, keys.jwt, { expiresIn: "24h" })
}
```

Рисунок 3.7 – Функція генерації токену

```
router.get('/', passport.authenticate('jwt', { session: false }), ctrlNotes.getNotesLimit);
router.get('/favorites', passport.authenticate('jwt', { session: false }), ctrlNotes.getFavoritesNotes);
router.get('/all', passport.authenticate('jwt', { session: false }), ctrlNotes.getAll);
router.get('/:id', passport.authenticate('jwt', { session: false }), ctrlNotes.getById);
```

Рисунок 3.8– Лістинг коду запиту з додаванням middleware аутентифікації

Запити, які не містять middleware аутентифікації, можуть виконуватися без необхідністю аутентифікації користувача.

3.5 Реалізація HTTP-запитів на сервер

Перед тим як почати розробляти інтерфейс SPA додатку, потрібно розповісти про Fetch API, який є сучасним інтерфейсом, що дозволяє робити HTTP-запити до серверів із веб-браузерів по мірі необхідності [20]. Метод fetch() саме дозволяє користувачу робити HTTP-запити до бекенда. За допомогою цього методу можна виконувати різні типи операцій за допомогою методів HTTP, як то метод GET для запиту даних від кінцевої точки, POST для надсилання даних до кінцевої точки тощо. Саме цим методом буде виконуватися запити на сервер. На рисунку 3.9 зображена структура запиту.

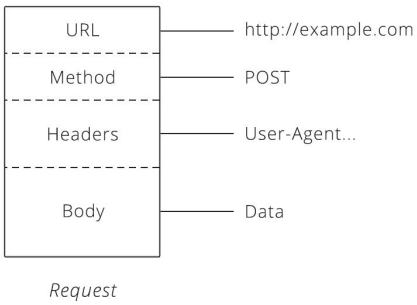

Рисунок 3.9 – Структура HTTP-запиту

В URL потрібно передати адресу, куди буде відправлено запит, в METHOD – тип запиту (GET, POST тощо), в BODY – тіло запиту, якщо це GET запит, зазвичай якусь інформацію в JSON форматі, в HEADES - заголовки, якщо вони потрібні (див. дод. Г). На рисунку 3.10 зображено приклад GET запиту.

```
const foldersHandler = async () => {
  try {
    const folders = await request("/api/folders", "GET", null, {
      Authorization: `${auth.token}`,
    });
    setData(folders);
  } catch (err) {}
};
```

Рисунок 3.10 – GET запит на отримання інформації про каталоги

Тобто в URL було передано адресу “/api/folders”,в тип методу - GET, тіло методу пусте, в заголовки - токен авторизації.

3.6 Інтерфейс користувача

Коли користувач зайшов на якийсь сайт, перш за все він звертаю увагу на його оформлення, адже це формує перші враження від веб-додатку, після цього користувач вже вирішує чи використовувати додаток далі, чи краще знайти інший, більш привабливий. Але й перевантаження різними інструментами, кнопками та іншими елементами інтерфейсу є не дуже доцільно, користувач просто злякається та буд шукати щось простіше.

Тому потрібно знайти золоту середину, зробити дизайн простим, привабливим, лаконічним та сучасним, який би зміг хоча би трішки позмагатися з іншими веб-додатками в галузі ведення електронних заміток.

Користувацький інтерфейс розроблено за допомогою JavaScript-бібліотеки React.js, який має в собі достатньо потрібних інструментів для

розробки повноцінного інтерфейсу, а за допомогою додаткових модулів та бібліотек можна розширити перелік компонентів та спростити розробку.

На рисунку 3.11 зображено сторінку входу на сайті, користувач побачить його перш за все інше.

Рисунок 3.11 – Сторінка входу

На рисунку 3.12 зображено сторінку реєстрації на сайті, користувачу потрібно ввести свої дані, та підтвердити свій обліковий запис в електронній пошті на яке прийде відповідне повідомлення.

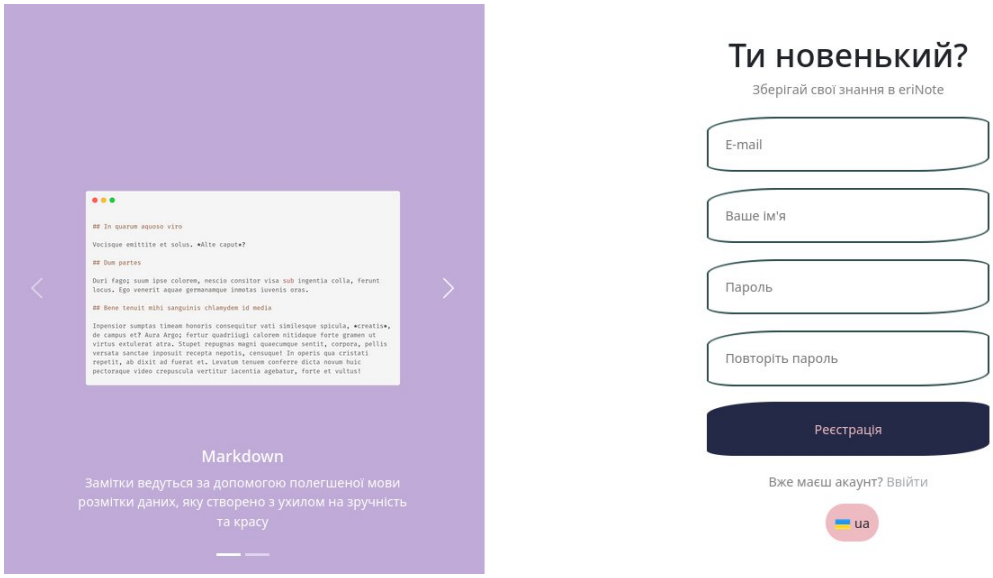

Рисунок 3.12 – Сторінка реєстрації

Якщо користувач зареєструвався але забув свій пароль, то він може відновити його на сторінці відновлення паролю, зображено на рисунку 3.13.

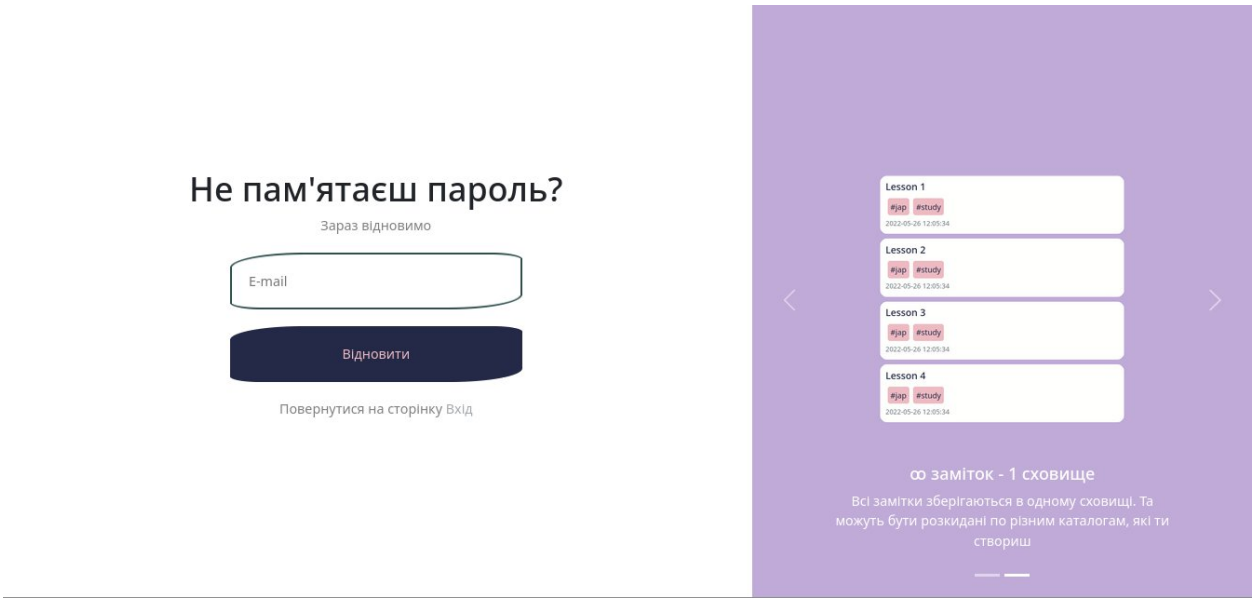

Рисунок 3.13 – Сторінка відновлення паролю

Після того як користувач зареєструється та ввійде, він побачить головну сторінку веб-додатка, яка містить меню навігації, список всіх заміток користувача та вікно, де буде показуватися відкрита замітка, зображено на рисунку 3.14.

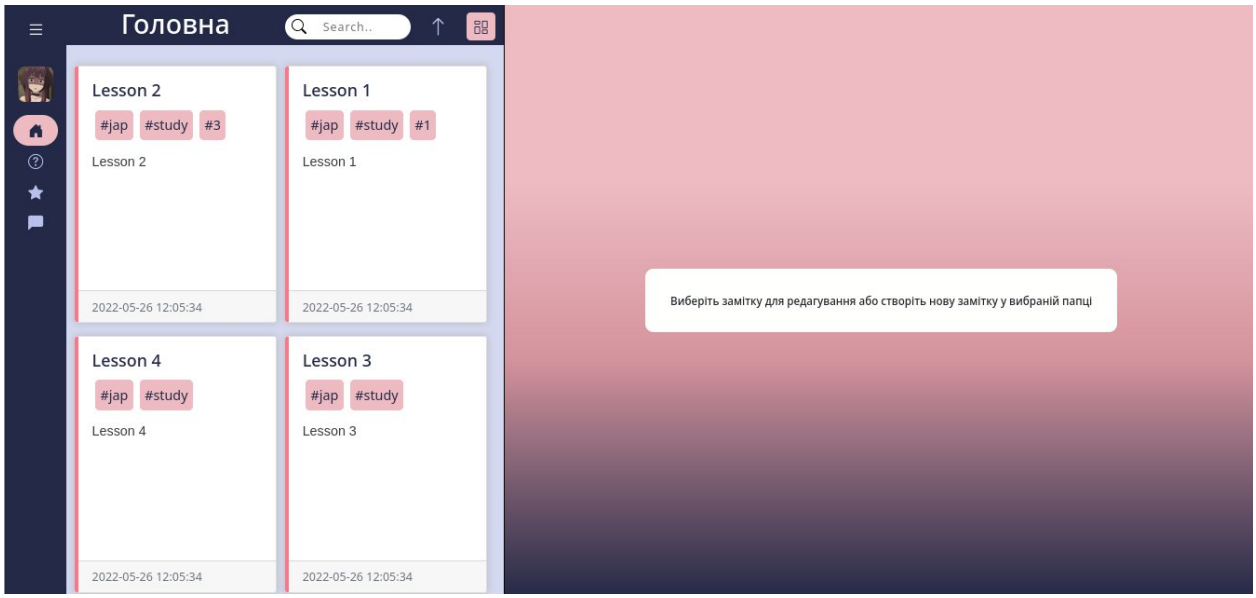

Рисунок 3.14 – Головна сторінка

Одним із компонентів не тільки головної сторінки, а й всього веб-додатка є меню навігації. Воно містить фото користувача, його ім’я, посилання на сторінку редагування профілю, також містить посилання на основні сторінки додатку, а саме на головну, сторінку допомоги, на важливі замітки та сторінку підтримки.

Меню навігації має в собі у свою чергу список посилань на каталоги, де зберігаються замітки та список посилань використаних тегів у записах.

Крім того меню ще має в собі кнопку для створення нових каталогів.

На рисунку 3.15 зображено навігаційне меню.

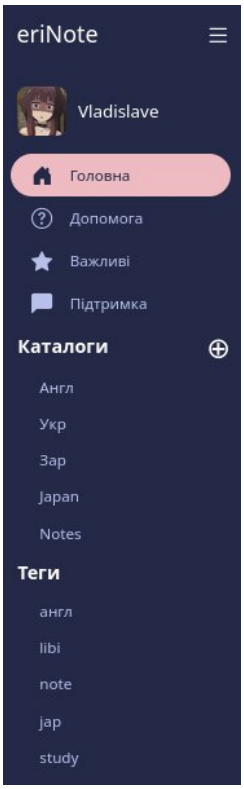

Рисунок 3.15 – Навігаційне меню

Інший компонент, який відіграю значну роль у веб-додатку, це список заміток. В списку відображаються всі існуючи замітки створені користувачем, а на сторінках каталогів та тегів, лише замітки, які безпосередньо належать їм. Також список містить верхнє меню, де розташовано поле для пошуку заміток та кнопки для сортування та виду показу списку.

Список заміток зображено на рисунку 3.16.

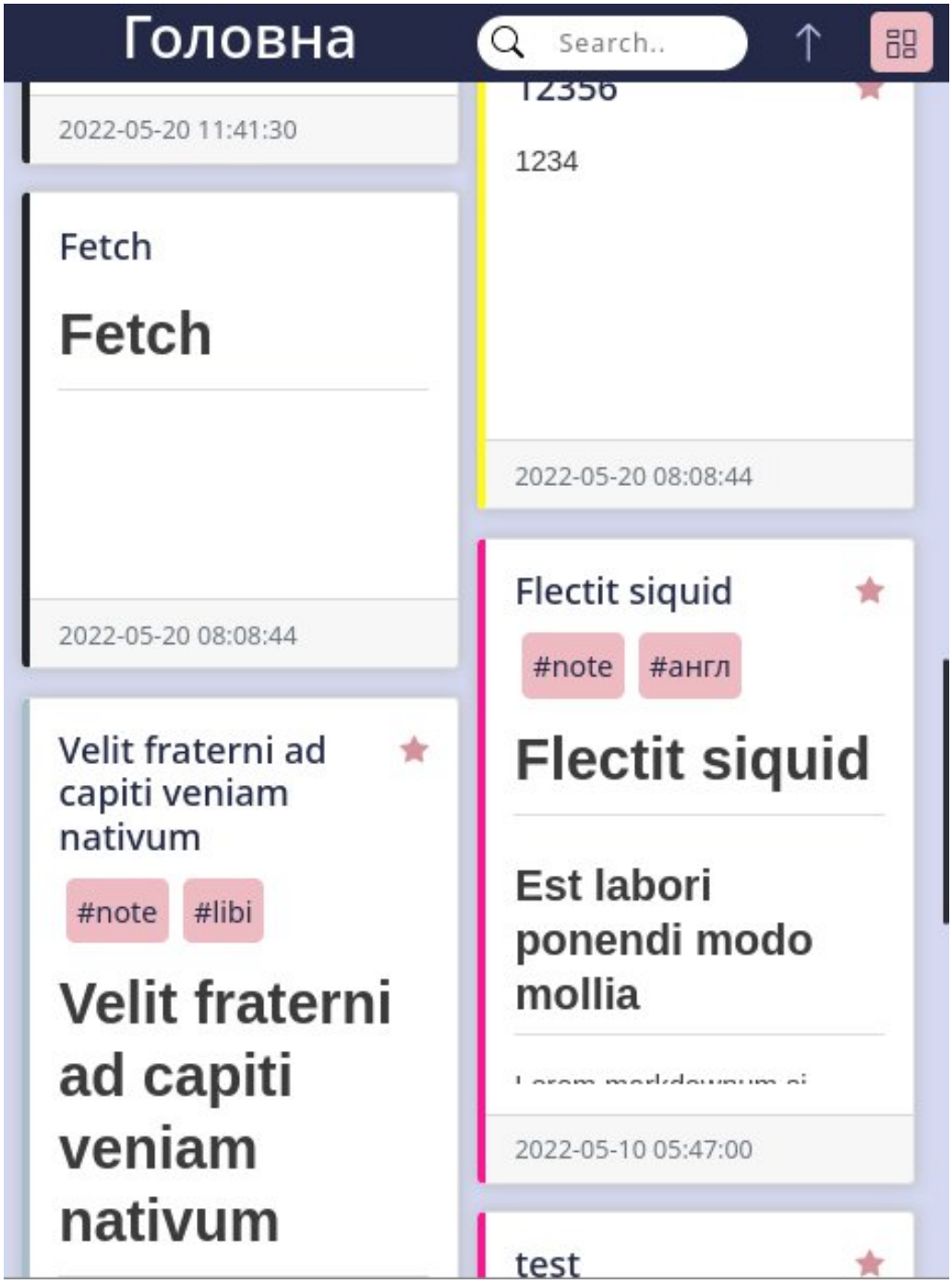

Рисунок 3.16 – Картковий список заміток

На рисунку 3.17 зображено список заміток з іншим видом відображення списку.

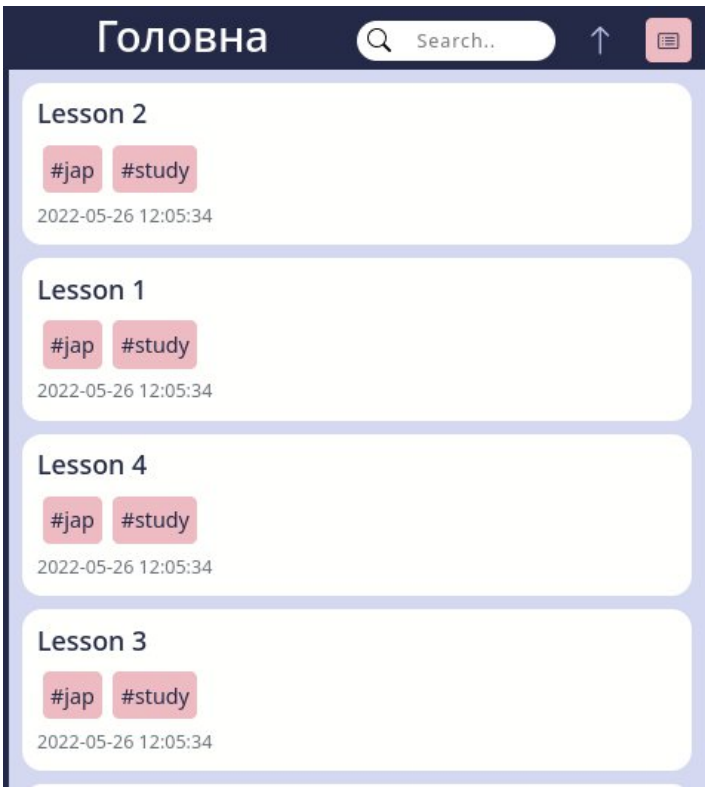

Рисунок 3.17 – Строковий список заміток

Окремо можна описати картку-посилання на замітку. Вона містить в собі заголовок замітки, мітку, яка вказує чи є замітка важливою чи ні, список тегів, пару рядків самого тексту замітки та дату створення, зображено на рисунку 3.18.

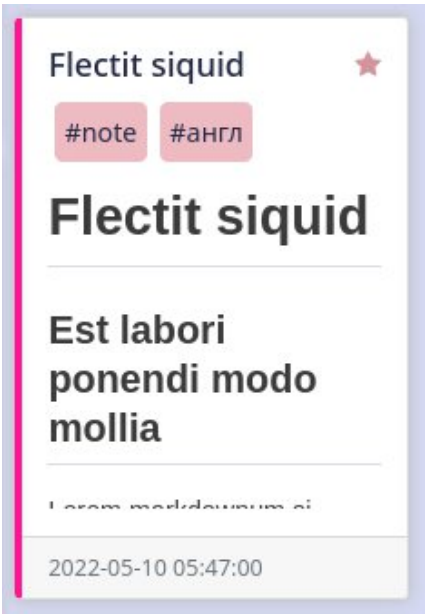

Рисунок 3.18 – Картка-посилання на замітку

Після головної сторінки йде сторінка допомоги користувачу. На цій сторінці розміщена коротка інформація про мову розмітки Markdown, саме на ній ведеться введення заміток на цьому веб-додатку. Зображено на рисунку 3.19.

Рисунок 3.19 – Сторінка допомоги

Далі по списку йде сторінка з важливими замітками. На даній сторінці розміщуються лише ті замітки, які користувач позначив як важливі, вони помічені зіркою в правому куті картки. Сторінка з важливими замітками зображена на рисунку 3.20.

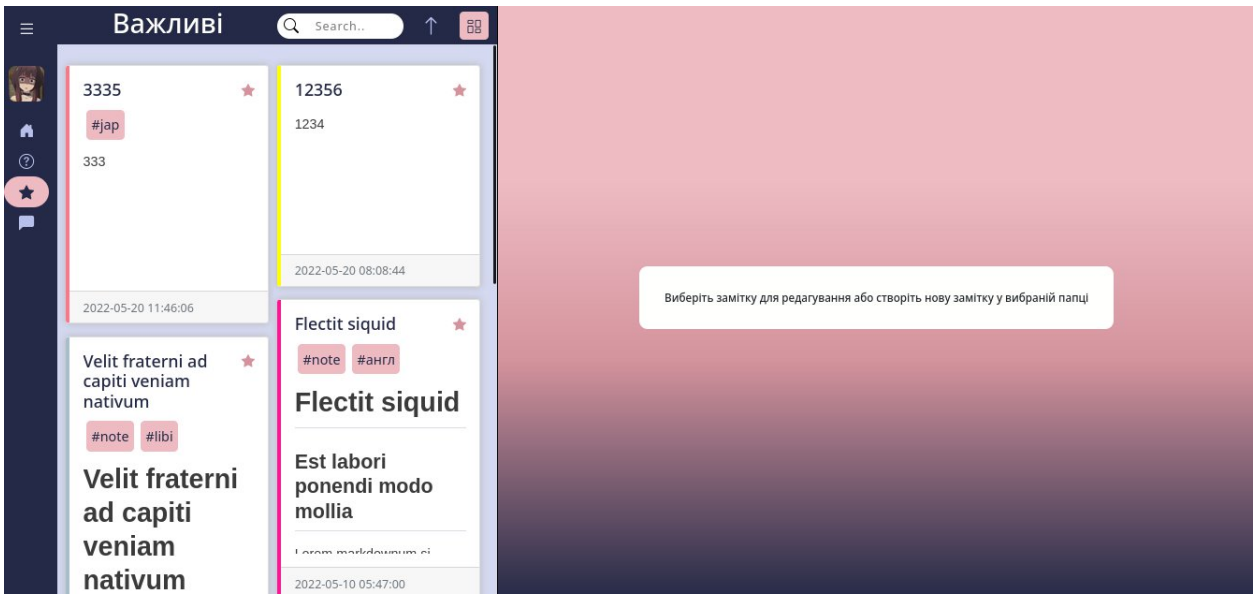

Рисунок 3.20 – Сторінка важливих заміток

Остання основна сторінка, це сторінка підтримки, вона призначена для зворотнього зв'язку користувача до розробника. Користувач може відправити форму з побажанням щось змінити для покращення використання веб-додатку, або заявити про знайдені баги або недоліки. Сторінка підтримки зображена на рисунку 3.21.

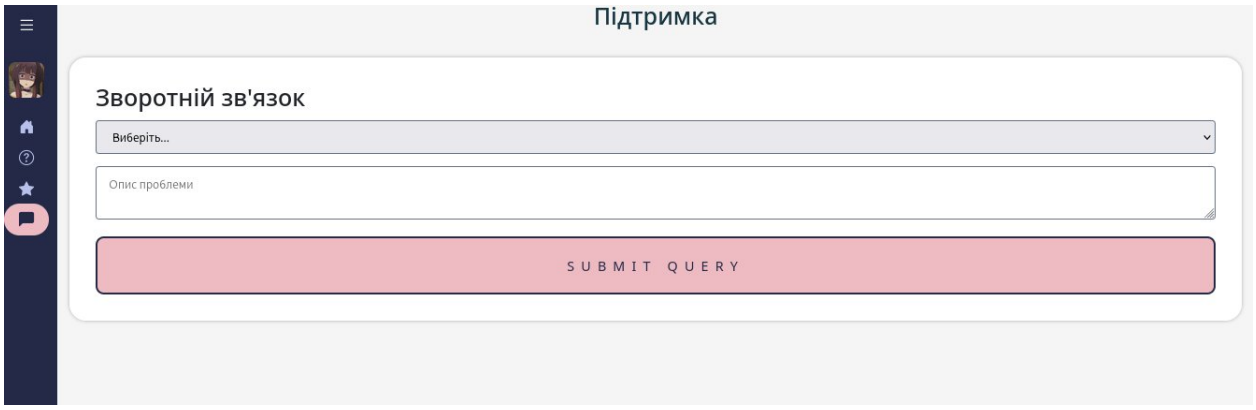

Рисунок 3.21 – Сторінка підтримки

Також слід розповісти про сторінку редагування профілю. Щоб до неї потрапити потрібно натиснути на свою аватарку або на своє ім’я в навігаційному меню, в результаті якого на екрані з’явиться невеличке меню,

яке зображене на рисунку 3.22, де потрібно вибрати відповідне посилання, також в даному меню можна вийти з облікового запису.

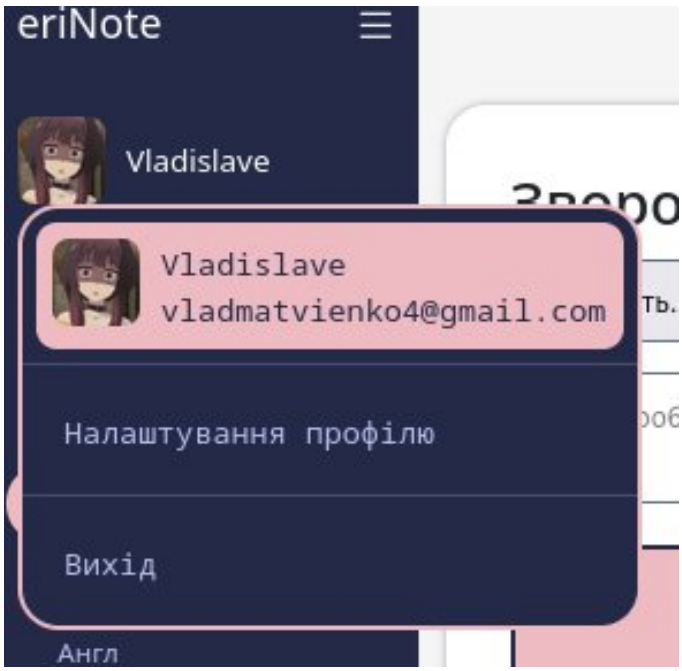

Рисунок 3.22 – Меню для вибору редагування профілю

На сторінці для редагування профілю можна змінити мову відображення веб-додатку, змінити своє ім’я та аватар, а також змінити пароль свого облікового запису. На рисунку 3.23 зображена сторінка для редагування профілю.

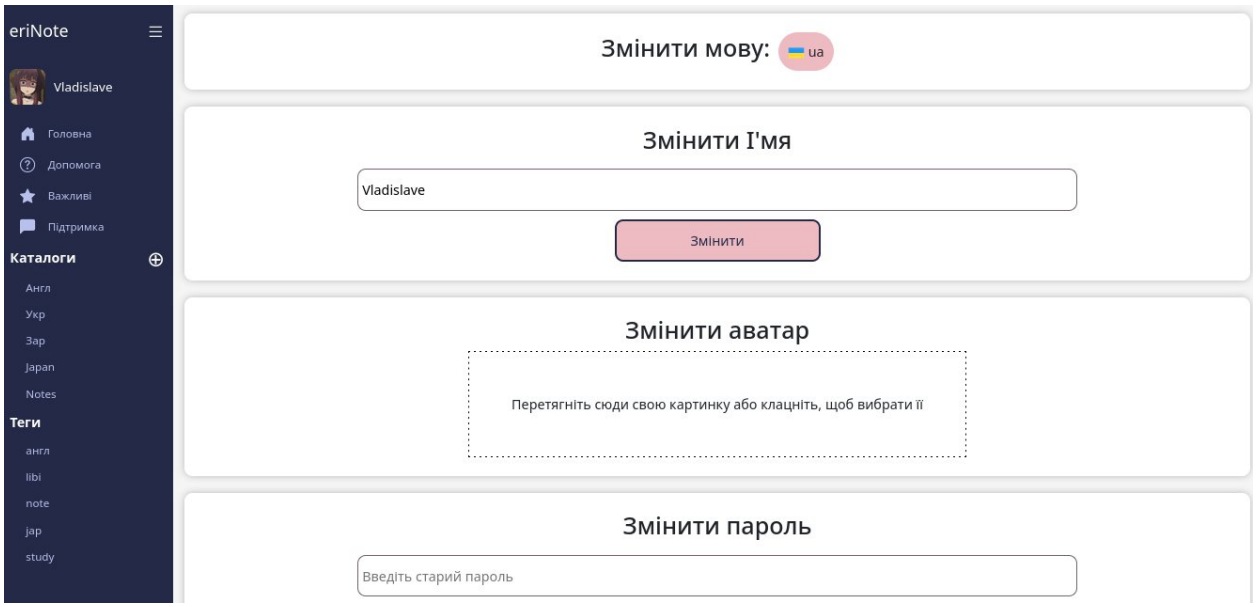

Рисунок 3.23 – Сторінка для редагування профілю

Головна частина додатку це каталоги. Вони структурують замітки по певній темі, предмету. Так замітки можна більш швидко знайти, і не витрачати свій час. Сторінка одного із каталогів зображено на рисунку 3.24.

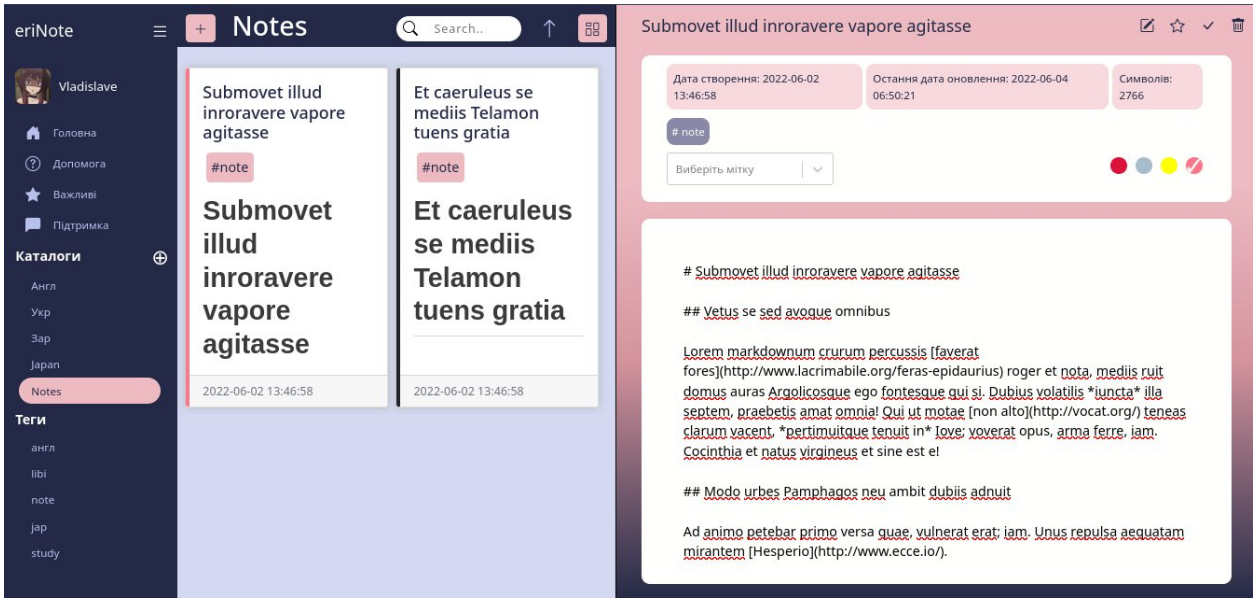

Рисунок 3.24 – Сторінка каталогу

Як і головна сторінка, сторінка каталогу містить навігаційне меню, список заміток, які належать до каталогу та вікна, де буде розташовуватися відкрита замітка. Але ще дана сторінку містить кнопку для створення нової замітки в каталозі. При натисканні на кнопку відкривається вікно для введення інформації про нову замітку. Також це вікно відкривається безпосередньо разом із відкриттям каталогу, на рисунку 3.25 показане дане вікно.

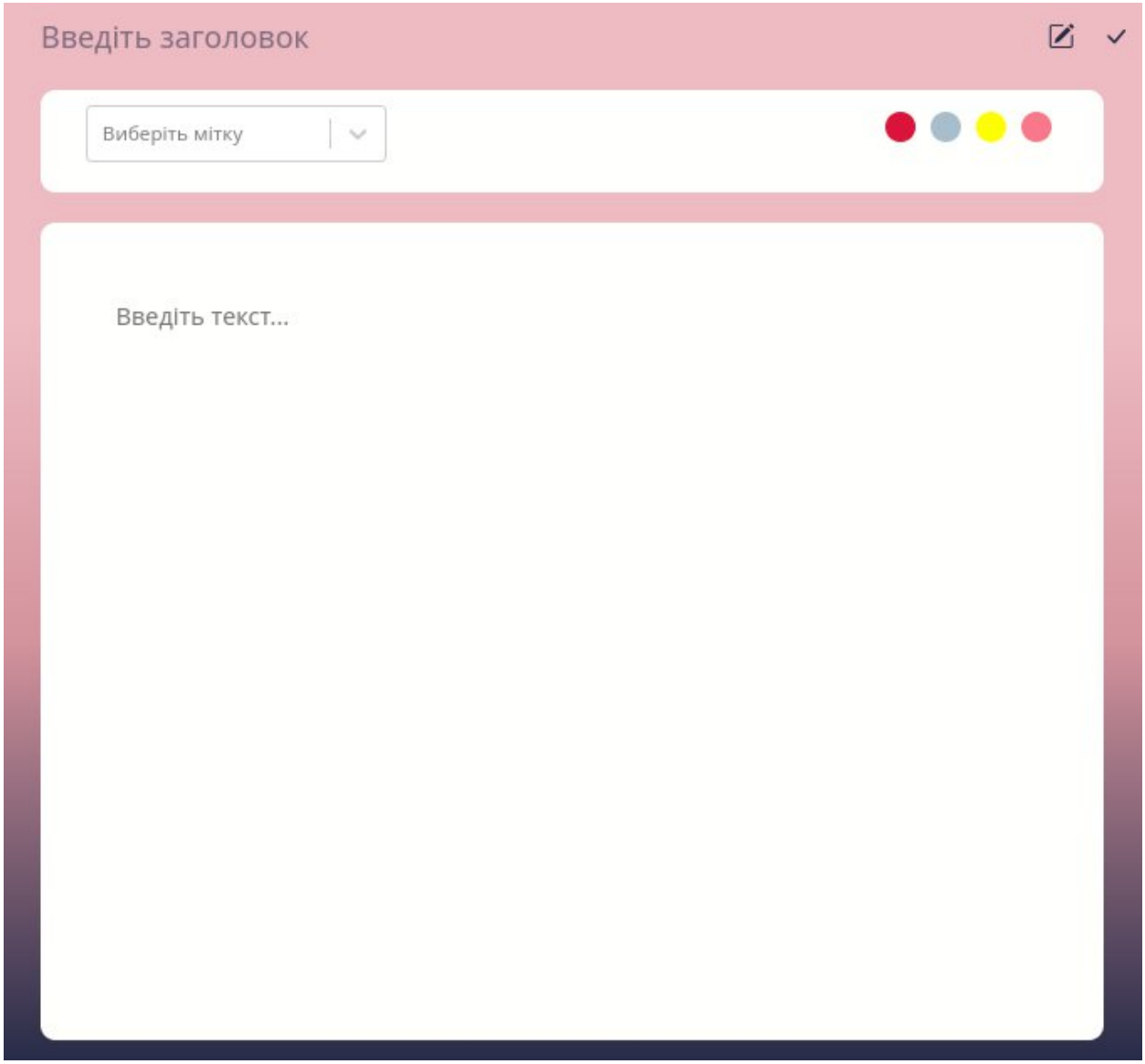

Рисунок 3.25 – Вікно для створення нової замітки

Щоб створити нову замітку потрібно лише ввести її заголовок, текст, додати існуючи теги або створити нові, та, за бажанням, вибрати колір лівого краю картки-посилання замітки для виділення серед інших.

Поряд з полем для введення заголовку знаходиться панель кнопок. Одна кнопка показує текст вже в застосованій розмітці Markdown, друга кнопка зберігає замітку в каталозі.

Також слід розглянути вікно відкритої замітки, яке зображено на рисунку 3.26.

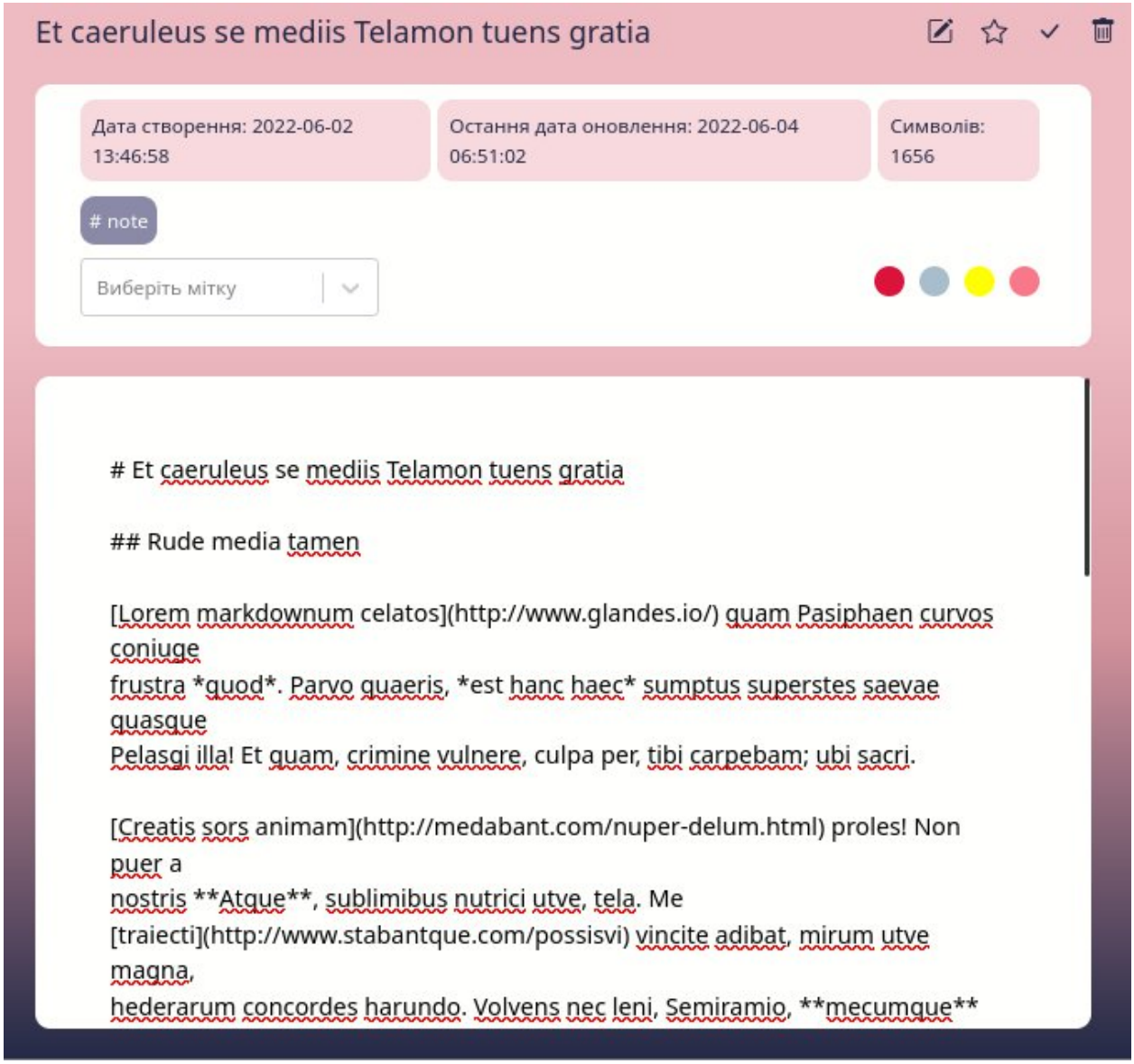

Рисунок 3.26 – Вікно відкритої замітки

Дане вікно як і вікно для створення замітки містить поле де знаходиться заголовок, поле з текстом замітки, селектор для додавання тегів, вибір кольору замітки, панель з кнопками, до якого додаються ще кнопки з видаленням замітки та додаванням до важливих. Також вікно має інформаційну панель, де можна дізнатися коли замітка була створена, коли останній раз оновлювалась та кількість символів загалом.

Дане вікно має два стани: редактор та показ. В стані редактора можна змінювати інформацію замітки, а саме заголовок, теги, текст, колір. В стані показу до тексту застосовується Markdown стилістика, як показано на рисунку 3.27.

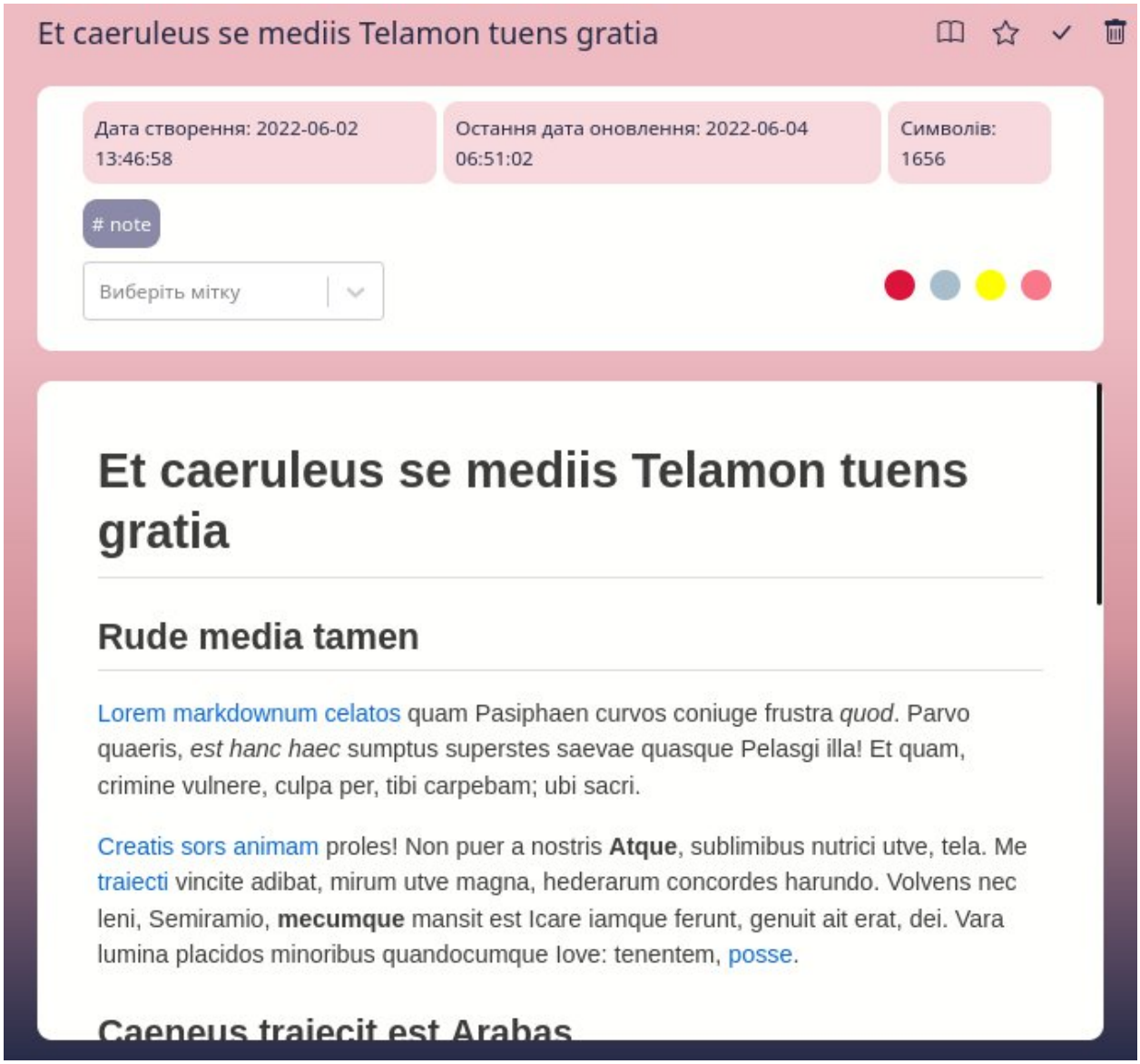

Рисунок 3.27 – Вікно відкритої замітки в стані показу

Було розглянуто основні елементи SPA-додатку. Окремо можна показати, як додаток враховує характеристики різних пристроїв у користувача, забезпечуючи правильне відображення веб-додатку на екранах різного розміру, тобто на смартфонах, планшетах та на інших розмірах екрана моніторів. На рисунку 3.28 зображено головну сторінку на планшеті iPad, а на рисунку 3.29 на смартфоні Galaxy S20.

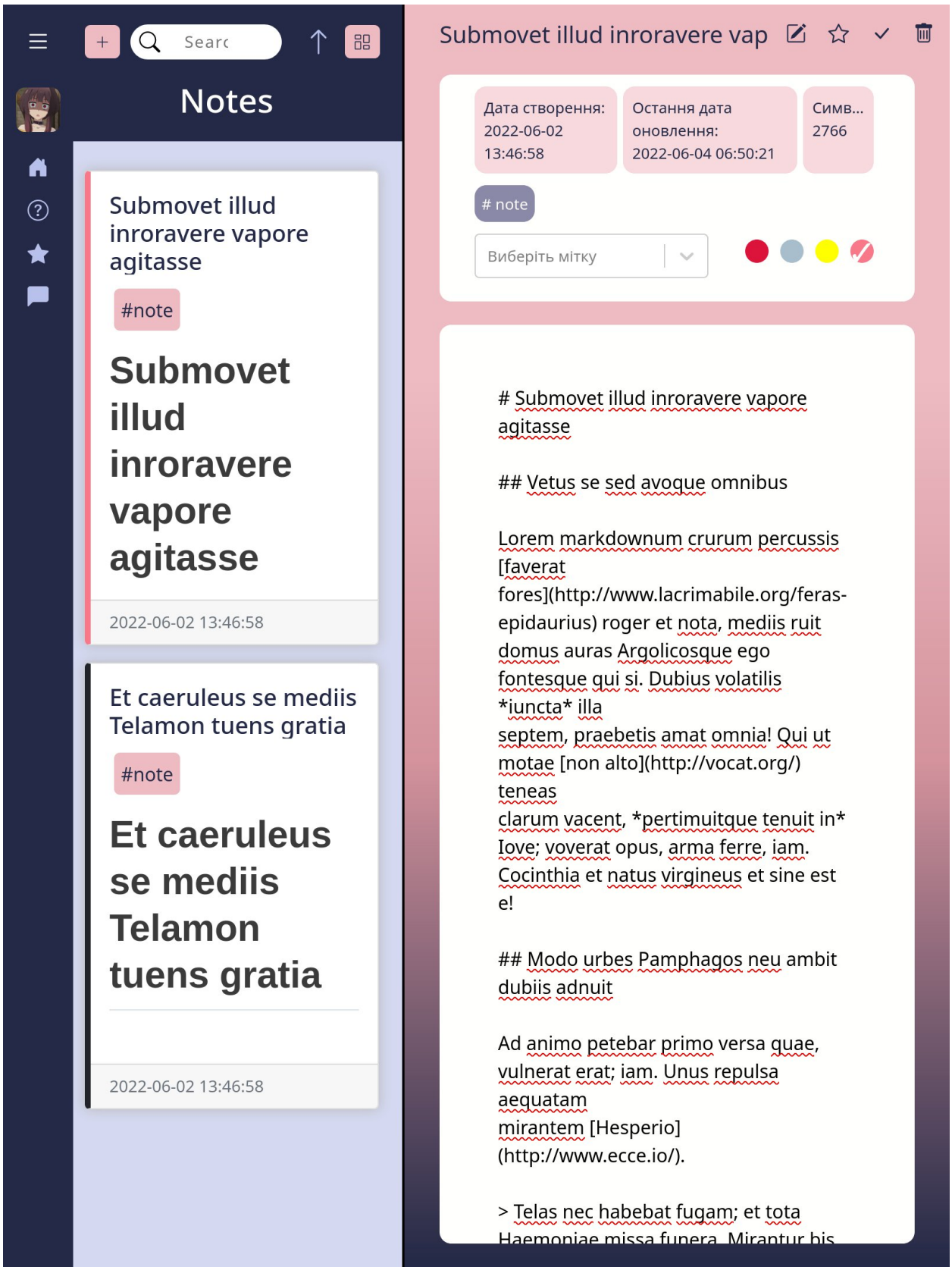

Рисунок 3.28 – Відображення сторінки каталогу на планшеті

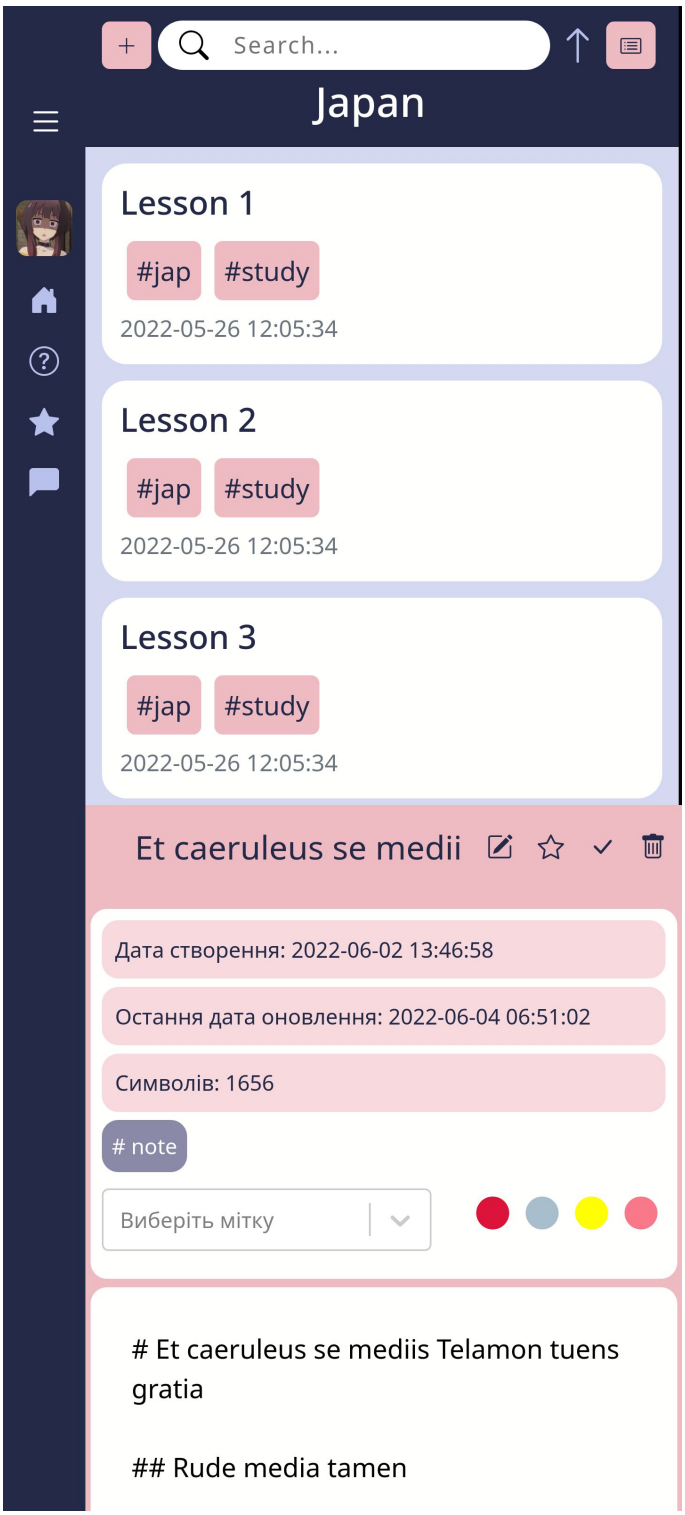

Рисунок 3.29 – Відображення сторінки каталогу на смартфоні

На рисунку 3.30 та 3.31 зображено сторінку редактування профілю на різних пристроях.

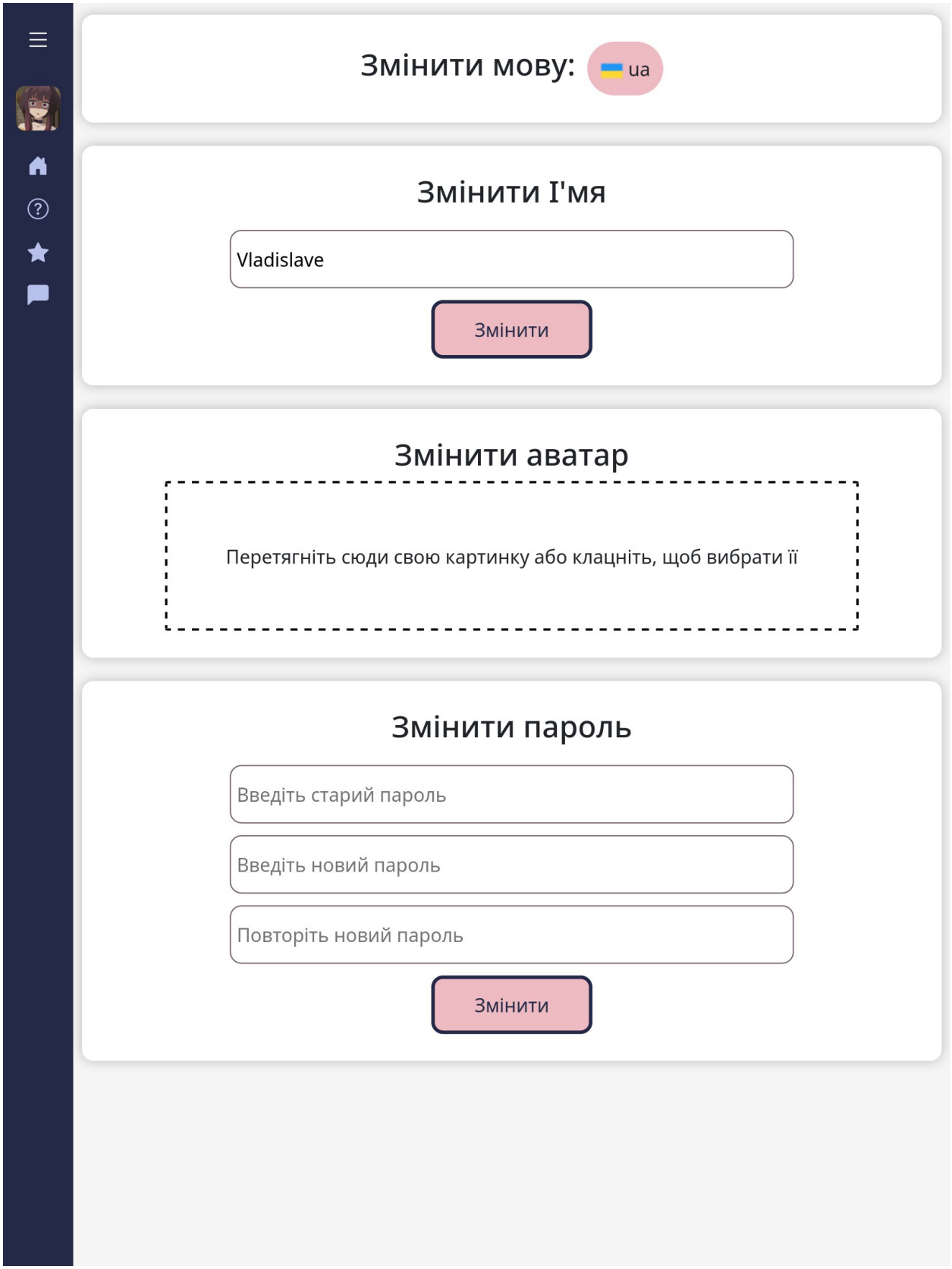

Рисунок 3.30 – Відображення сторінки редактування профілю на планшеті

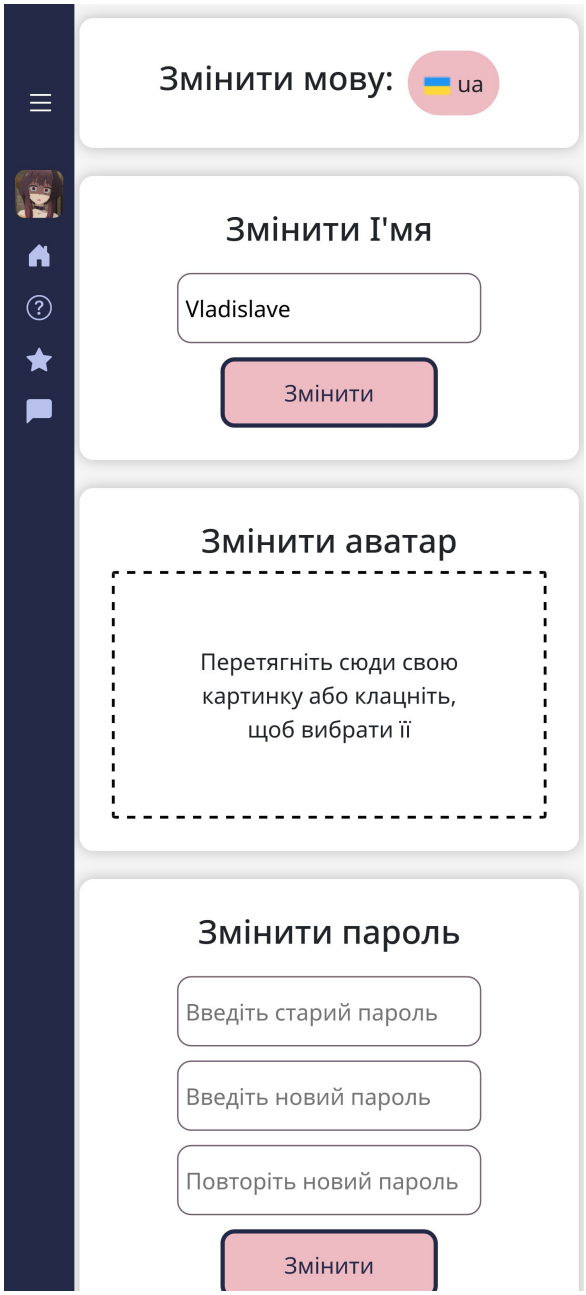

Рисунок 3.31 – Відображення сторінки редактування профілю на смартфоні

В результаті було створено швидкий та привабливий інтерфейс, та доволі простий у використанні, ненавантажений не потрібними інструментами, за допомогою мови розмітки Markdown, замітки мають зрозумілий вигляд, в замітки можна додати таблиці, посилання на інші ресурси, картинки, анімацію, лістинг коду та багато іншого. Важливі замітки можна відокремити.

ВИСНОВКИ

При використанні різних додатків для ведення заміток, все частіше виявляються деякі проблеми у їх функціональності, комфортній роботі з ними та й не дуже простому використанні.

Тому було вирішено розробити веб-додаток який би зміг якомога краще задовольнити потреби потреби користувачів, які хочуть з легкістю вести свої замітки.

У результаті було обрано та проаналізовано найоптимальніші інструменти для розробки SPA додатку, а саме стек MERN, який складається з таких технологій, як MongoDB, Express.js, React.js та Node.js. Для авторизації та аутентифікації користувачів було обрано метод JWT токенів.

Також продумано інтуїтивно зрозумілий інтерфейс SPA додатку для простоти використання користувачем.

Проаналізовані потрібні вимоги до веб-додатка який повинен мати:

– ефективне введення заміток;
– зберігання і накопичування заміток користувачем;
– можливість пошуку потрібних заміток по заголовку та тексту;
– каталоги для сортування заміток за деякими критеріями;
– налаштування профілю під потреби користувача.

Розроблений веб-додаток задовольняє всім вимогам, поставленим на етапі постановки завдання.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Людина у сучасному інформаційному просторі [Електронний ресурс] // https://www.bsmu.edu.ua – Режим доступу до ресурсу: https://www.bsmu.edu.ua/blog/3869-lyudina-u-suchasnomu-informatsiynomu-prostori/

2. Веб-додатки: види, архітектура та принципи роботи [Електронний ресурс] // https://highload.today – Режим доступу до ресурсу: https://highload.today/veb-prilozheniya/

3. SPA (Single-page application) [Електронний ресурс] // https://developer.mozilla.org – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Glossary/SPA

4. Web Application Architecture in 2022: Moving in the Right Direction [Електронний ресурс] // https://mobidev.biz/ – Режим доступу до ресурсу: https://mobidev.biz/blog/web-application-architecture-types.

5. MERN Stack Explained [Електронний ресурс] // https://www.simplilearn.com – Режим доступу до ресурсу: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs.

6. The Best Guide to Know What Is React [Електронний ресурс] // https://www.mongodb.com – Режим доступу до ресурсу: https://www.mongodb.com/mern-stack..

7. React. A JavaScript library for building user interfaces [Електронний ресурс] // – Режим доступу до ресурсу: https://reactjs.org/

8. Посібник: знайомство з React [Електронний ресурс] // https://uk.reactjs.org – Режим доступу до ресурсу: https://uk.reactjs.org/tutorial/tutorial.html.

9. Learning React: Modern Patterns for Developing React Apps Eve Porcello, Alex Banks, 2020.

10. Express/Node introduction [Електронний ресурс] // https://developer.mozilla.org – Режим доступу до ресурсу: https://developer.mozilla.org/docs/Learn/Server-side/Express_Nodejs/Introduction.

11. Guides [Електронний ресурс] // https://nodejs.org – Режим доступу до ресурсу: https://nodejs.org/en/docs/guides/.

12. Web Development with Node and Express: Leveraging the JavaScript Stack Ethan Brown, 2014

13. Node.js in Action Mike Cantelon, Marc Harter, Nathan Rajlich, T.J. Holowaychuk, 2011

14. What is MongoDB? Introduction, Architecture, Features & Example [Електронний ресурс] // https://www.guru99.com – Режим доступу до ресурсу: https://www.guru99.com/what-is-mongodb.html.

15. MongoDB Documentation [Електронний ресурс] // https://www.mongodb.com – Режим доступу до ресурсу: https://www.mongodb.com/docs/

16. What is Authentication? Different Types of Authentication [Електронний ресурс] // https://blog.miniorange.com – Режим доступу до ресурсу: https://blog.miniorange.com/different-types-of-authentication-methods-for-security/

17. Введення в REST API [Електронний ресурс] // https://habr.com – Режим доступу до ресурсу: https://habr.com/ru/post/483202/

18. RESTful Web APIs: Services for a Changing World Leonard Richardson, Sam Ruby, 2013

19. Using Passport for authentication in Node.js [Електронний ресурс] // https://blog.logrocket.com – Режим доступу до ресурсу: https://blog.logrocket.com/using-passport-authentication-node-js/

20. Using the Fetch API [Електронний ресурс] // https://developer.mozilla.org – Режим доступу до ресурсу: https://developer.mozilla.org/enUS/docs/Web/API/Fetch_API/Using_Fetch

ДОДАТКИ

Додаток А. Лістинг програмного коду запуску сервера

```
#!/usr/bin/env node

/**
 * Module dependencies.
 */

var app = require('../app');
var debug = require('debug')('mean:server');
var http = require('http');

/**
 * Get port from environment and store in Express.
 */

var port = normalizePort(process.env.PORT || '5000');
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);

/**
 * Listen on provided port, on all network interfaces.
 */
```

```
server.listen(port);
server.on('error', onError);
server.on('listening', onListening);

/**
 * Normalize a port into a number, string, or false.
 */

function normalizePort(val) {
  var port = parseInt(val, 10);

  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
    // port number
    return port;
  }

  return false;
}

/**
 * Event listener for HTTP server "error" event.
 */

function onError(error) {
```

```
  if (error.syscall !== 'listen') {
    throw error;
  }

  var bind = typeof port === 'string'
    ? 'Pipe ' + port
    : 'Port ' + port;

  // handle specific listen errors with friendly messages
  switch (error.code) {
    case 'EACCES':
      console.error(bind + ' requires elevated privileges');
      process.exit(1);
      break;
    case 'EADDRINUSE':
      console.error(bind + ' is already in use');
      process.exit(1);
      break;
    default:
      throw error;
  }
}

/**
 * Event listener for HTTP server "listening" event.
 */

function onListening() {
  var addr = server.address();
  var bind = typeof addr === 'string'
```

```
    ? 'pipe ' + addr
    : 'port ' + addr.port;
  debug('Listening on ' + bind);
}
```

Додаток Б. Лістинг програмного коду головного файлу

```
var createError = require('http-errors');
var express = require('express');
var mongoose = require('mongoose')

var passport = require('passport');

var bodyParser = require('body-parser');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

const i18next = require('i18next')
const Backend = require('i18next-fs-backend')
const middleware = require('i18next-http-middleware')

var authRouter = require('./app_server/routes/auth');
var pagesRouter = require('./app_server/routes/pages');
var foldersRouter = require('./app_server/routes/folders');
var profileRouter = require('./app_server/routes/profile');

var keys = require('./app_server/config/keys')

var app = express();
```

```
app.use(bodyParser.urlencoded({ extended: true }))
app.use(bodyParser.json())


if (process.env.NODE_ENV === 'production' || process.env.NODE_ENV ===
'development') {
mongoose.connect(keys.mongoURI)
  .then(() => console.log("MongoDb connect"))
  .catch(error => console.log(error))
}

if (process.env.NODE_ENV === 'test') {
  mongoose.connect(keys.mongoURIforTests)
  .then(() => console.log("MongoDb for tests connect"))
  .catch(error => console.log(error))
}

app.use(passport.initialize())
require('./app_server/middleware/passport')(passport)

i18next.use(Backend).use(middleware.LanguageDetector).init({
  fallbackLng: 'ua',
  preload: ['ua'],
  backend: {
    loadPath: './app_server/public/locales/{{lng}}/translation.json'
  }
})
```

```
// view engine setup
app.set('views', path.join(__dirname, 'app_server', 'views'));
app.set('view engine', 'jade');

app.use(middleware.handle(i18next));

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));
app.use(express.json({ extended: true }))

app.use('/api/pages', pagesRouter);
app.use('/api/folders', foldersRouter);
app.use('/api/auth', authRouter);
app.use('/api/profile', profileRouter);

app.use('/images', express.static(path.join(__dirname, 'app_server', 'images')))

if (process.env.NODE_ENV === 'production') {
  app.use('/', express.static(path.join(__dirname, 'app_client', 'build')))

  app.get('*', (req, res) => {
    res.sendFile(path.resolve(__dirname, 'app_client', 'build', 'index.html'))
  })
}

// catch 404 and forward to error handler
app.use(function (req, res, next) {
```

```
  next(createError(404));
});

// error handler
app.use(function (err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
```

Додаток В. Лістинг програмного коду контролерів

```
const Folders = require('../models/Folders');
const Notes = require('../models/Notes');
const User = require('../models/Users');
const errorHandler = require('../utils/errorHandler');

var consts = require('../constants/consts')

module.exports.getAll = async (req, res) => {
    try {
        const folders = await Folders.find({
            user: req.user.id
        });
```

```
        res.status(200).json(folders)
    } catch (error) {
        errorHandler(res, error)
    }
};

module.exports.getById = async (req, res) => {
    try {
        const folder = await Folders.findById(req.params.id)
        res.status(200).json(folder)
    } catch (error) {
        errorHandler(res, error)
    }
};

module.exports.create = async (req, res) => {

    const folder = new Folders({
        name: req.body.name,
        user: req.user.id
    })
    try {
        const candidate = await User.findOne({ _id: req.user.id })
        const countFolder = await Folders.find({ user: req.user.id }).count()
        const planUser = candidate.plan.toLowerCase();
        if (consts[planUser].folder <= countFolder) {
            res.status(401).json({
                message: req.t('limit_folders')
            })
        } else {
```

```
            await folder.save()
            res.status(201).json({
                id: folder._id,
                name: folder.name,
                message: "Папка створена"
            })
        }
    } catch (error) {
        errorHandler(res, error)
    }
};

module.exports.update = async (req, res) => {
    const updated = {
        name: req.body.name
    }
    try {
        await Folders.findOneAndUpdate(
            {
                _id: req.params.id,
                user: req.user.id
            },
            { $set: updated },
            { new: true }
        )
        res.status(200).json({
            message: "Ім'я папки змінено"
        })
    } catch (error) {
        errorHandler(res, error)
```

```
    }
};

module.exports.remove = async (req, res) => {
    try {
        await Folders.deleteOne({ _id: req.params.id, user: req.user.id })
        await Notes.deleteMany({ folder: req.params.id })
        res.status(200).json({
            message: req.t('remove_folder')
        })
    } catch (error) {
        errorHandler(res, error)
    }
};
```

Додаток Г. Лістинг програмного коду http-запитів

```
import jwtDecode from "jwt-decode";
import { useCallback, useContext, useState } from "react";
import { AuthContext } from "../context/AuthContext";
import { useMessage } from "../hooks/message.hook";

export function useHttp() {
  const message = useMessage();
  const auth = useContext(AuthContext);

  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  const request = useCallback(
```

```
async (url, method = "GET", body = null, headers = {}) => {
  setLoading(true);
  try {
    const token = localStorage.getItem("userData");
    if (token) {
      const decodedToken = jwtDecode(token);
      const currentDate = new Date();

      if (decodedToken.exp * 1000 < currentDate.getTime()) {
        message("Session end");
        auth.logout();
      }
    }

    if (body) {
      body = JSON.stringify(body);
      headers["Content-Type"] = "application/json";
    }

    const req = await fetch(url, {
      method,
      body,
      headers,
    });
    const data = await req.json();
    setLoading(false);
    if (!req.ok) {
      throw new Error(data.message);
    }
```

```
        return data;
      } catch (e) {
        setLoading(false);
        //auth.logout()
        setError(e.message);
        throw e;
      }
    },
    []
  );
  const clearError = useCallback(() => setError(null), []);

  return { error, loading, request, clearError };
}
```