

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Інформаційна екосистема в складі веб-додатку і
Telegram-бота на космічну тематику»**

Виконав студент групи ПЗ-19-1
ФЕДЧЕНКОВ Іван Юрійович

Керівник роботи:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
МАРЧУК Галина Вікторівна

Житомир – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

В.о. зав. кафедри інженерії
програмного забезпечення
Андрій МОРОЗОВ
«14» лютого 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ФЕДЧЕНКОВ Іван Юрійович**

Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**

Тема роботи: «Система для управління та організації проектів»,
затверджена Наказом закладу вищої освіти від «14» лютого 2023 р., №70с

Вихідні дані для роботи: використання веб-орієнтованих технологій для розробки комплексної інформаційної системи у складі веб-додатку і Telegram-бота на космічну тематику

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єфремов Ю.М.	16.02.2023р.	16.02.2023р.
2	Єфремов Ю.М.	16.02.2023р.	16.02.2023р.
3	Єфремов Ю.М.	16.02.2023р.	16.02.2023р.

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі	5 лютого 2023 – 20 лютого 2023	Виконано
2	Аналіз моделей аналогів	21 лютого 2023 – 1 березня 2023	Виконано
3	Постановка технічного завдання	2 березня 2023 – 12 березня 2023	Виконано
4	Вивчення літературних джерел	13 березня 2023 – 17 березня 2023	Виконано
5	Проектування структури	18 березня 2023 – 24 березня 2023	Виконано
6	Написання програмного коду	25 березня 2023 – 18 квітня 2023	Виконано
7	Тестування роботи програми	19 квітня 2023 – 21 квітня 2023	Виконано
8	Аналіз одержаних результатів	22 квітня 2023 – 25 квітня 2023	Виконано
9	Оформлення пояснювальної записки	15 червня 2023 – 21 червня 2023	Виконано
10	Попередній захист	26 квітня 2023	Виконано

РЕФЕРАТ

Кваліфікаційна робота складається з комплексної програмної системи для взаємодії з інформаційними матеріалами, а також пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи містить 76 сторінок, 28 ілюстрацій, 9 таблиць, 2 додатки та 20 літературних джерел.

Основною метою даної роботи є проектування і створення комплексного програмного продукту, що надає користувачам доступ до інформаційних матеріалів на тему космосу і сучасної науки, як в контексті читання і обговорення в коментарях, так і в контексті написання власних матеріалів, а також дає можливість робити запити у відкриту базу даних астрономічних об'єктів SIMBAD прямо з месенджера Telegram

В роботі визначено ключові завдання, пов'язані з розробкою системи, та проведено аналіз подібних систем, а також наведено сценарії роботи програмного комплексу. Представлено огляд загальної структури системи, описано алгоритми взаємодії між окремими модулями системи, а також наведено опис об'єктної структури (діаграму класів) системи. Крім того, в статті представлено результати тестування системи.

КЛЮЧОВІ СЛОВА: CMS, ВЕБ-СИСТЕМА, КОСМОС, СТАТТІ, ОБГОВОРЕННЯ, SIMBAD

					ІПЗ.КР.Б – 121 – 23 – ПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	Інформаційна екосистема у складі веб-сайту і Telegram-бота на космічну тематику Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		І.Ю.Федченко						
Керівник		Ю.М. Єфремов					4	76
Рецензент		Г.В.Марчук				Житомирська політехніка, група ІПЗ-19-1		
Зав. каф.		А.В.Морозов						

ABSTRACT

The bachelor's thesis consists of a comprehensive software system for interaction with informational content, as well as an explanatory note. The explanatory note to the thesis contains 76 pages, 28 illustrations, 9 tables, 2 appendices, and 20 references.

The main purpose of this thesis is to design and develop a comprehensive software system which gives users access to informational content about space exploration and modern science. It includes reading, writing and discussing articles, and also simplifies querying public database of astronomical objects SIMBAD.

The paper identifies key tasks related to the development of the system, analyses existing systems similar to the one under development, and presents scenarios of the software complex. An overview of the general structure of the system is presented, the algorithms of interaction between individual modules of the system are described, and the object structure (class diagram) of the system is described. In addition, the article presents the results of testing the system.

KEYWORDS: CMS, WEB-SYSTEM, SPACE, ARTICLES, DISCUSSION, SIMBAD.

					ПЗ.КР.Б – 121 – 23 – ПЗ	
						5

Зміст

РЕФЕРАТ	6
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	8
ВСТУП	9
РОЗДІЛ 1. АНАЛІЗ ЗАДАЧІ ТА ОСОБЛИВОСТЕЙ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ	11
1.1.Постановка задачі	11
1.2.АНАЛІЗ АНАЛОГІВ ПРОГРАМНОГО ПРОДУКТУ	13
1.3 Розробка архітектури інформаційної екосистеми на космічну ТЕМАТИКУ	17
1.4 ОБҐРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ТА ВИМОГИ ДО АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ.....	20
Висновки до розділу 1.....	22
РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ.....	23
2.1 Визначення варіантів використання та структури системи.....	23
2.2. ОПИС СТРУКТУРИ СИСТЕМИ	26
2.3. ПРОЄКТУВАННЯ БАЗ ДАНИХ СИСТЕМИ.....	29
2.4. РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ЕКОСИСТЕМИ ГАММА, СТАТЕЙ, КОМЕНТАРІВ, ЗАУВАЖЕНЬ, СКАРГ ТА КОРИСТУВАЧІВ.....	32
Висновки до розділу 2.....	51
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ	52
3.1. Порядок встановлення та налаштування системи.	52
3.2.СТРУКТУРА ІНТЕРФЕЙСУ. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ.	53
3.3.ТЕСТУВАННЯ РОБОТИ СИСТЕМИ.....	60
Висновки до розділу 3.....	61
ВИСНОВКИ.....	62

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	63
ДОДАТКИ.....	65
ДОДАТОК А.....	66
ДОДАТОК Б	70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

MVC – Model-View-Controller (Модель–представлення–контролер)

ORM – Object-Relational-Mapping (Об’єктно-реляційне відображення)

ERD – Entity Relationship Diagram (Модель «сутність-зв'язок»)

RBAC – Role-Based Access Control (Керування доступом на основі ролей)

REST – Representational State Transfer (підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів)

API - Application Programming Interface (Прикладний програмний інтерфейс)

UML – Unified Modeling Language (Уніфікована мова моделювання)

CMS – Content Management System (Система управління вмістом)

CRUD – create, read, update and delete (створення, читання, редагування, видалення)

БД – База даних

SPA – Single Page Application (патерн для створення веб-додатків)

S3 — Хмарне файлове сховище від Amazon, частина AWS

ВСТУП

Актуальність теми. Тематика космосу, і науки в цілому, стає все більш і більш популярною. Саме тому виникає потреба в інформаційних порталах, які популяризують і систематизують новини з цієї сфери, роблять доступ до потрібної інформації простішим, а також дають можливість зробити свій внесок у розвиток науково-популярного дискурсу.

Веб-сайт з актуальними статтями про космос надає людям можливість дізнаватись останні новини, відкриття, космічні місії та події. Крім того, можливість обговорення статей у коментарях дозволяє людям висловлювати свої думки, обмінюватись інформацією та спілкуватись з однодумцями. А можливість писати власні статті надає платформу для людей, які мають цікаву інформацію, дослідження або власний погляд на певні питання.

Така система буде мати наступні переваги:

Доступність: Обговорення улюбленої теми стає доступним з будь-якого місця, де є підключення до інтернету.

Відкритість: Користувач може приєднатись до обговорення не тільки шляхом коментування, а й написавши власну статтю, яка після перегляду і схвалення верифікованим автором буде опублікована.

Обмін думками: Система надає можливість обговорювати статті з іншими людьми, дізнаватись нове і ділитись власними знаннями

Стимулювання цікавості до теми: Система заохочує користувачів цікавитись чимось новим з даної сфери.

Знайомства з новими людьми: Система дозволяє познайомитись з новими людьми, з якими користувач має спільні інтереси.

Доступ до не надто популярної серед аматорів інформації: Система надає можливість робити запити у відкриту базу даних SIMBAD, яка містить дані про безліч астрономічних об'єктів і виводить цю інформацію у форматі, зручному для сприйняття.

Пошук статей на потрібну тему: Система дозволяє шукати статті за різними критеріями.

					ІПЗ.КР.Б – 121 – 23 – ПЗ	
						9

Метою даної кваліфікаційної роботи є проектування і розробка інформаційної системи на космічну тематику. Створення такої системи має на меті спростити доступ до інформації про досягнення сучасної науки, а також дозволити користувачам ділитись власними знаннями на тему космосу.

Цей проект спрямований на вирішення проблем, з якими стикаються користувачі, що мають бажання дізнатись більше про космос, або поділитись власними знаннями на цю тему, а також знайти нових людей зі спільними інтересами.

Встановлена мета обумовлює наступні завдання:

- аналіз завдань, пов'язаних з науково-популярним дискурсом на тему космосу
- розробка загальної структури системи та визначення її ключових компонентів
- вибір відповідних інструментів і технологій для реалізації системи
- впровадження системи

Об'єктом дослідження є спрощення доступу до обговорення космосу, а також популяризація даної тематики.

Предметом дослідження є використання веб-орієнтованих технологій для розробки комплексної інформаційної системи що враховує специфіку теми і відповідає їй.

Впровадження інформаційної системи на космічну тематику має потенціал для значного підвищення популярності цієї теми в світі, а також спростить процес обговорення останніх і не тільки новин, досліджень. А також дасть можливість кожному користувачу мати зручний і швидкий доступ до інформації з відкритого джерела-бази даних SIMBAD.

РОЗДІЛ 1. АНАЛІЗ ЗАДАЧІ ТА ОСОБЛИВОСТЕЙ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1. Постановка задачі

Головною метою створення інформаційної системи на космічну тематику є спрощення доступу до інформації, її обговорення а також поширення власних думок і знань шляхом написання статей. Така система стане в нагоді не тільки людям, що вже знайомі з темою, так і новим людям, яким дана тема може бути цікавою.

Кваліфікаційна робота вимагає створення веб-додатку і Telegram-бота, за допомогою веб-додатку можна буде переглядати, коментувати і писати статті, а бот стане в нагоді у разі інтересу до певного астрономічного об'єкту і його точних характеристик, а також надасть можливість ознайомитись зі статтями, що стосуються певного об'єкта, у разі наявності таких статей в системі.

Створення такої системи включає наступні етапи:

- Створення серверної частини системи:
 - Створення ядра системи в окремому модулі
 - Підключення до бази даних
 - Підключення до Redis для комунікації мікросервісів між собою
 - Ініціалізація можливих дій мікросервісів
 - Створення модуля з типами даних, що актуальні одразу для кількох мікросервісів
 - Створення мікросервісу users, що відповідає за функціонал користувачів
 - Створення бази даних users і моделей в мікросервісі users:
 - Модель User і таблиця users
 - Модель Token і таблиця tokens
 - Реалізація CRUD для перелічених вище моделей
 - Створення мікросервісу article, що відповідає за взаємодію зі статтями

- Створення бази даних article і моделей в мікросервісі article:
 - Модель Article і таблиця articles
 - Модель Comment і таблиця comments
 - Модель Remark і таблиця remarks
 - Модель Report і таблиця reports
- Реалізація CRUD для перелічених вище моделей
- Створення мікросервісу gateway, що має http сервер і відповідає за комунікацію з клієнтською частиною
 - Проектування і створення маршрутизації
 - Реалізація аутентифікації
- Створення мікросервісу files, що відповідає за взаємодію з файлами і хмарним сховищем
 - Реалізація додавання файлів до хмарного сховища AWS S3
- Створення мікросервісу simbad, що відповідає за взаємодію з базою даних SIMBAD
 - Реалізація запиту в SIMBAD і оптимізації відповіді
- Створення мікросервісу telegram-bot, що відповідає за роботу Telegram-бота
 - Реалізація отримання даних з мікросервісу simbad і їх виведення
- Тестування серверної частини системи
- Створення клієнтської частини веб-додатку
 - Створення ядра клієнтської частини додатку
 - Створення клієнтської маршрутизації
 - Створення допоміжного функціоналу для взаємодії з API серверної частини
 - Створення всіх необхідних сторінок додатку
 - Тестування клієнтської частини веб-додатку
- Тестування всієї системи

Результатом виконання поставленої задачі є мікросервісна система, що складається з веб-додатку і Telegram-бота, основною характерною рисою якої є

					ІПЗ.КР.Б – 121 – 23 – ПЗ	
						12

масштабованість і простота додання нового функціоналу, адже система є модульною, кожен мікросервіс працює незалежно від інших, і додання нових нового функціоналу не порушує роботу вже існуючого. Система дає можливість перегляду, написання, обговорення статей для звичайного користувача, а також огляд, публікація, або додання зауважень до ще не опублікованих статей для привілейованого користувача ролі “Автор”, перегляд скарг і видалення коментарів або користувачів у разі ролі “Модератор” і зміна ролей користувачів для ролі “Адміністратор”.

1.2. Аналіз аналогів програмного продукту

Тема космосу набирає популярність, а отже вже існують деякі веб-додатки, що є аналогом нашого додатку. Варто зауважити, що подібного до створеного Telegram-бота не існує в широкому доступі, тому нижче буде наведене порівняння лише веб-додатків

Alpha Centauri — найбільший україномовний сайт про космос. Має значний функціонал, з переваг-зручний пошук, поділ новин на розділи, є можливість додавати коментарі, вхід через Telegram, Google, Facebook. Також має допоміжний функціонал, наприклад календар запусків космічних місій, список найактивніших користувачів, і так далі. З недоліків, деякий вміст сайту є платним. Деякі статті, і інший функціонал доступний лише за умови підтримки проекту фінансово. Інтерфейс є трохи перевантаженим і може досить важко сприйматись, особливо для новоприбулих користувачів. Неможливо переглядати коментарі, не увійшовши в акаунт. На рис. 1.1 показано сайт Alpha Centauri

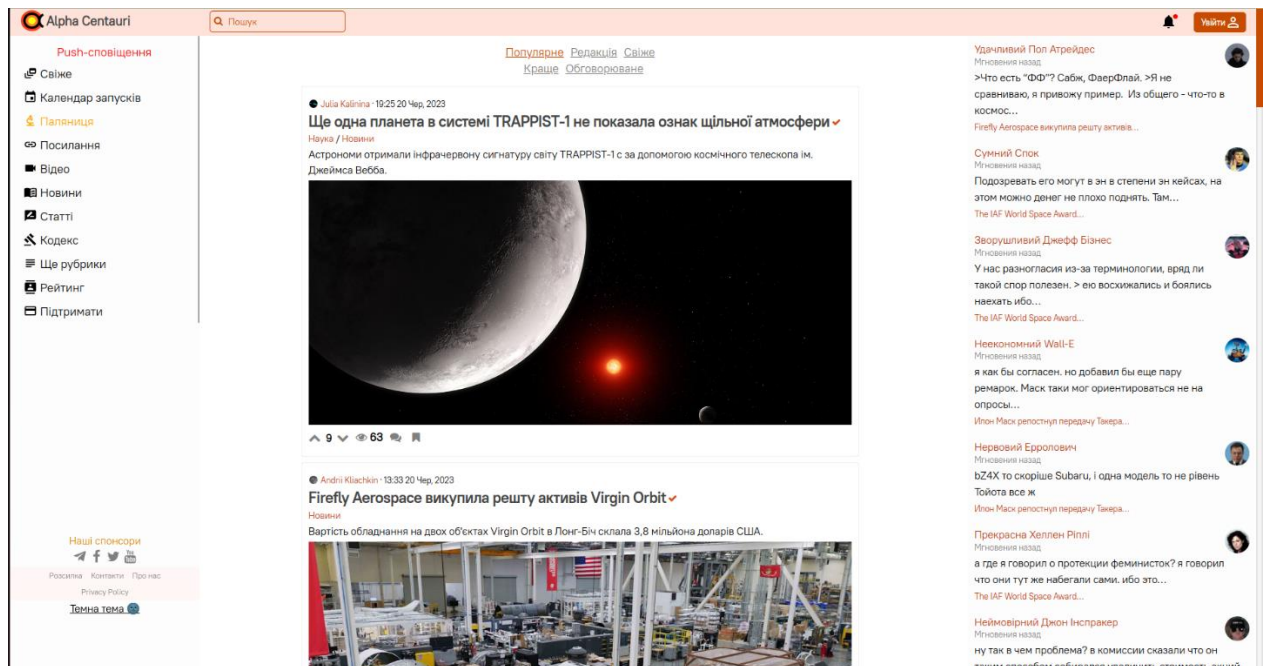


Рис. 1.1 — Сайт Alpha Centauri

Переваги:

- Зручний і ефективний пошук і фільтрація статей
- Широкий допоміжний функціонал (календар пусків, список найактивніших користувачів)
- Вхід за допомогою Google, Telegram, Facebook

Недоліки:

- Деякий вміст вимагає фінансово підтримати проект для використання
- Недоступність коментарів для неавторизованого користувача
- Перевантаженість інтерфейсу, що може викликати труднощі у новоприбулих користувачів

space.vn.ua — веб-сайт з статтями на космічну тематику. Серед переваг — простий і мінімалістичний дизайн, додатковий функціонал, наприклад, місячний календар. З недоліків — неможливість коментувати статті, абсолютно незручний пошук, в цілому відсутність ролі користувача в житті і розвитку сайту. Тобто додаток є достатньо ізольованим, і втручання в його наповнення є абсолютно неможливим для звичайного користувача. На рис. 1.2 показано сайт space.vn.ua.

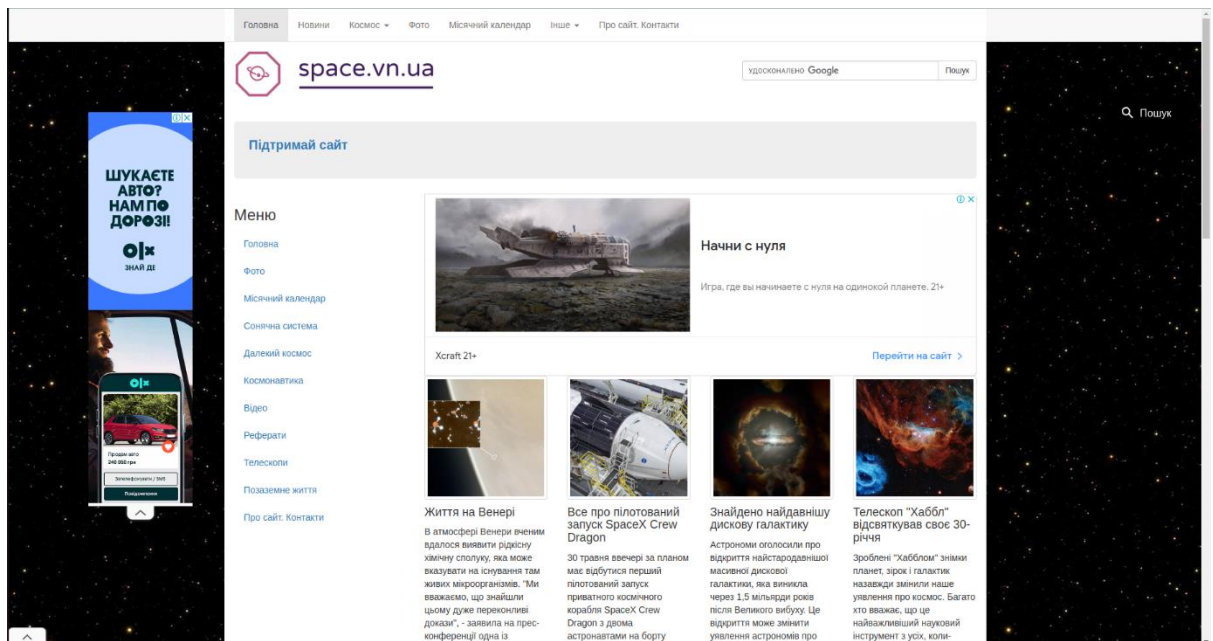


Рис. 1.2 – Сайт space.vn.ua

Переваги:

- Простий і зрозумілий інтерфейс
- Додаткові функції (місячний календар і т.д.)

Недоліки:

- Відсутність коментарів
- Незручний пошук
- Відсутність ролі користувача, як такої, а відповідно неможливість зробити власний внесок у розвиток сайту

Space.com.ua — досить відомий інформаційний сайт про космос. Його перевагами є зручний інтерфейс, наявність на головній сторінці декількох розділів, а також групування новин за тематикою. А недоліки — відсутність розширеного пошуку, відсутність коментарів і в цілому участі користувача в житті сайту, не відповідний тематиці космосу дизайн сайту, крім того, ніщо на сайті не говорить про його “космічне” спрямування.

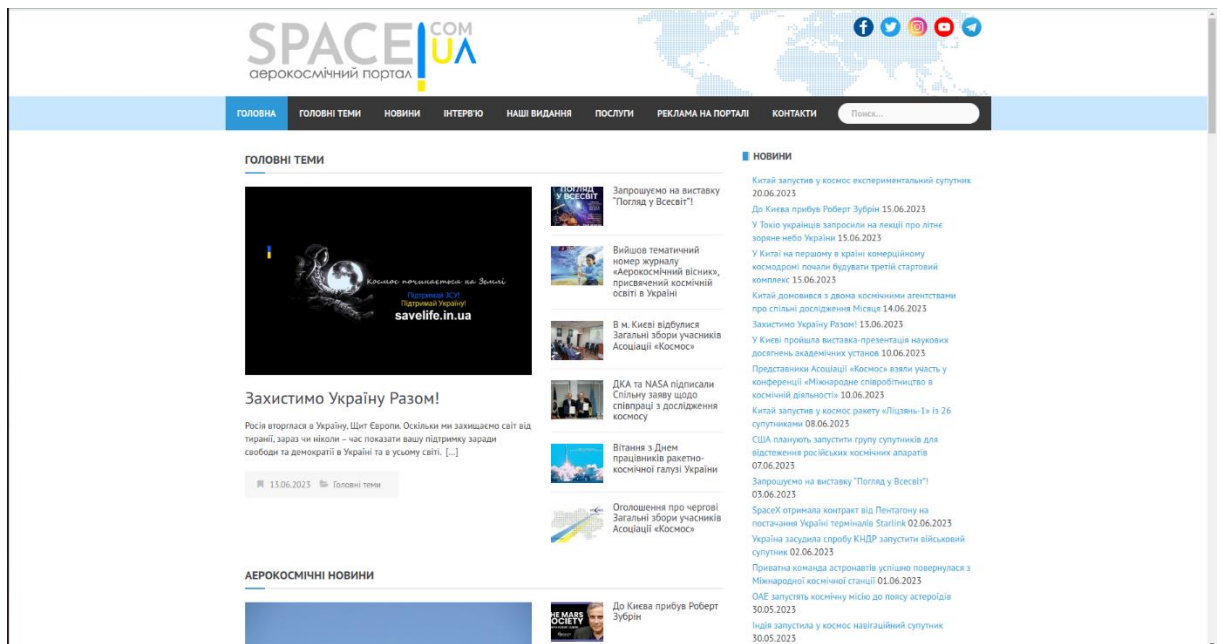


Рис. 1.3 – Сайт Space.com.ua

Переваги:

- Зручний інтерфейс
- Наявність розділів
- Групування новин за тематикою

Недоліки:

- Відсутність розширеного пошуку
- Відсутність коментарів
- Нульова участь користувача в житті сайту
- Невідповідність дизайну і тематики
- Відсутність в системі чогось специфічного, що вказувало б на її космічне спрямування

Переваги і недоліки варто розписати у вигляді порівняльної таблиці. Це буде дуже корисно при створенні вимог до системи, що розроблюється. Критерії будуть підібрані відповідно до раніше виділених переваг та недоліків кожного з веб-додатків

Три вказані вище веб-додатки порівняні в таблиці 1.1

Порівняння існуючих аналогів менеджерів проектів

	Alpha Centauri	space.vn.ua	Space.com.ua
Зручність інтерфейсу	±	+	+
Відповідність сайту вимогам тематики	+	±	-
Зручність пошуку	+	-	±
Наповнення функціоналом	+	±	-
Зручність відображення інформації	+	±	±
Участь користувача в житті сайту	+	-	-

Проаналізувавши аналоги веб-додатку що розроблюється, можна однозначно виділити лідера-сайт Alpha Centauri. Переваги і недоліки кожного з цих сайтів має сенс врахувати, щоб розробити максимально ефективний продукт. Також, дана сфера має властивість досить швидко рости і розвиватись, відповідно система, яка створюється, має бути гнучкою і не викликати проблем при розширенні функціоналу в майбутньому. Завдяки мікросервісній архітектурі можна забезпечити вищезгадані характеристики. Окрім того, веб-додатки, вказані вище не є частиною певної екосистеми, на відміну від системи, що розроблюється, що є однозначним мінусом кожного з них.

Тому, головними рисами нашої системи будуть: простий і зручний дизайн, зручний пошук, сортування, можливість коментувати і писати статті, а також додатковий функціонал, а саме можливість робити запити в базу SIMBAD

1.3 Розробка архітектури інформаційної екосистеми на космічну тематику

Так як система має бути гнучкою і легко розширюватись, було вирішено використати мікросервісну архітектуру. На відміну від монолітної, вона є стійкішою, простіша в розширенні, але складніша в проектуванні і на початку

розробки. Для розробки було обрано мову TypeScript [2] і фреймворк Moleculer.js [1].

Мікросервісна архітектура передбачає розбиття всіх можливих дій системи на декілька мікросервісів, які виконують всю необхідну роботу і відправляють результат виконання на інтерфейс користувача. На рис. 1.4 проілюстровано схему мікросервісної архітектури

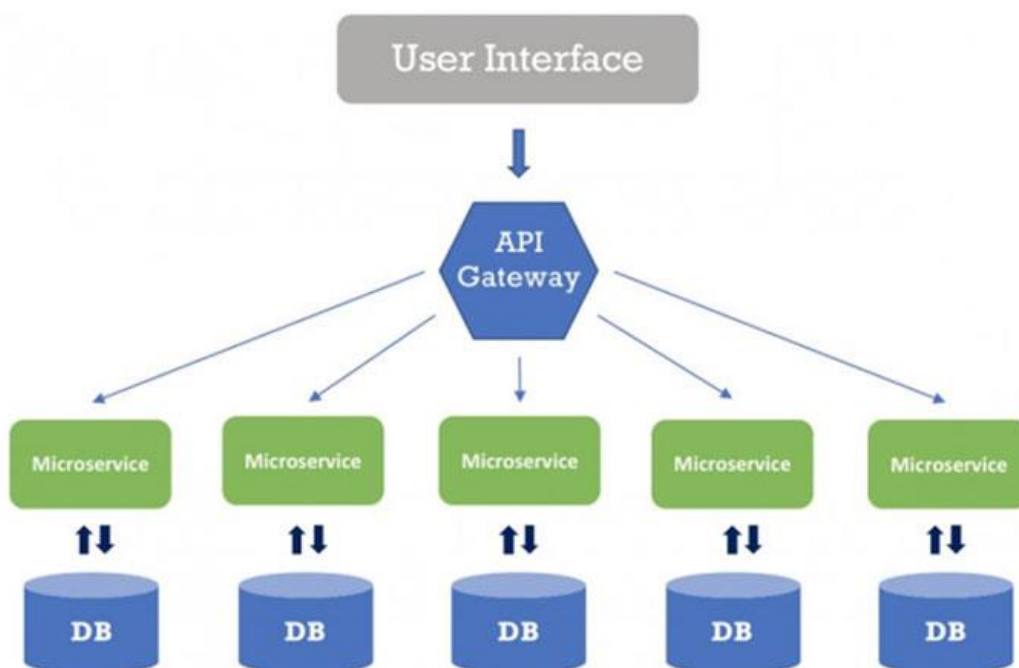


Рис. 1.4 – Схема мікросервісної архітектури

Мікросервісна архітектура набуває все більшої популярності останнім часом, у зв'язку з зростаючим попитом на системи що є більш відмовостійкими і простішими в розширенні, а також простішими в плані адаптації під різні платформи.

Веб-орієнтована мікросервісна архітектура має наступну будову:

API Gateway — мікросервіс, що відповідає за безпосередню взаємодію з клієнтською частиною, саме через нього йдуть всі запити, які він направляє на інші сервіси

Мікросервіси, що не взаємодіють з користувачем напряму — мікросервіси, що виконують основну роботу, запити в бази даних, операції з даними, можуть комунікувати між собою.

Бази даних — зберігають дані. Важливо, що кожен мікросервіс, за потребою, має власну базу даних, адже спільна база для всіх мікросервісів буде гальмувати роботу кожного з них

Це типовий приклад. В нашому випадку маємо ще один мікросервіс, про який варто згадати

Мікросервіс-Telegram-бот — відповідає за комунікацію з користувачем через месенджер Telegram, аналогічно до API Gateway але іншим шляхом

Мікросервісна архітектура підходить для розробки інформаційної екосистеми з таких причин:

- Гнучкість і модульність. Змінення одного компонента не несе фатальних наслідків для системи в цілому
- Розподілення обов'язків. Кожен компонент відповідає за свою частину роботи, що спрощує розробку, підтримку і масштабування
- Розширюваність. Додання нового функціоналу не перешкоджає роботі вже існуючого, що дуже важливо для системи, яка має адаптуватись під поточну ситуацію
- Високий відсоток перевикористання програмного кода

Також, більша частина системи — це веб додаток, то варто згадати про клієнтську частину. Для її реалізації було використано React.js

Таким чином, система також реалізує шаблон MVC. Даний шаблон передбачає наявність моделей — вони відповідають за взаємодію з базою даних, контролерів-вони обробляють дані, взаємодіють з моделями, і віддають користувачу представлення-дані для веб-сторінки, або повідомлення в месенджері Telegram. На рисунку 1.5 зображено схему роботи шаблону MVC

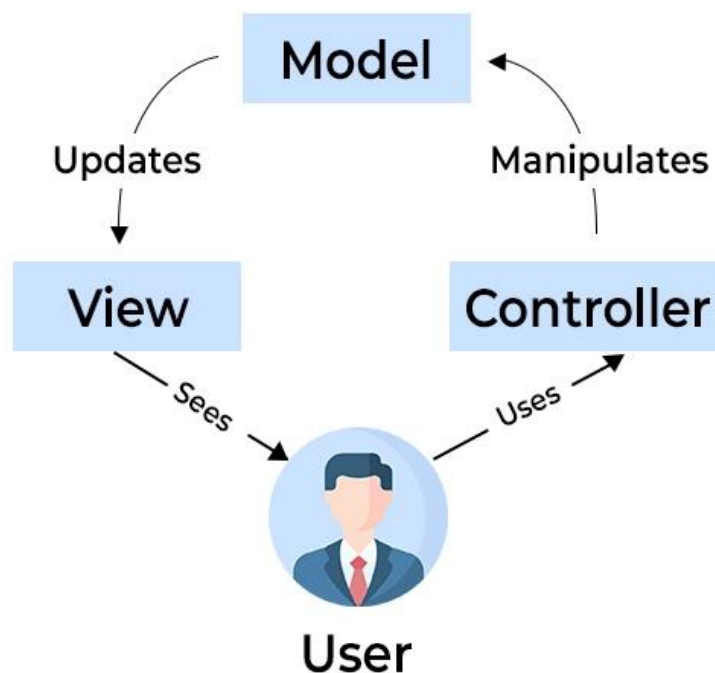


Рис. 1.5. - Ілюстрація шаблону MVC

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

При створенні інформаційної екосистеми потрібно обрати інструменти, які зроблять систему максимально ефективною, а також пришвидшать і спростять процес розробки, і подальшої підтримки

В першу чергу, треба обрати мову програмування і мікросервісний фреймворк. Так як було обрано мікросервіси і веб-розробка, то потрібно обрати сучасну мову, яка має спеціалізовані мікросервісні фреймворки. Такою мовою є TypeScript, на платформі Node.js [4]

Переваги:

- Швидкодія. Node.js є досить ефективною платформою, і здатна швидко виконувати поставлені задачі
- Оптимізація навантаження. Завдяки тому що Node.js використовує асинхронні API для виконання синхронних задач і єдиний синхронний потік для виконання синхронних задач, що потребує менших потужностей для роботи
- Строга типізація. TypeScript є надбудовою над звичайним JavaScript, додає строгу типізацію, що робить код безпечнішим і більш передбачуваним

- Можливість використовувати користувацькі модулі, що суттєво розширяє базовий функціонал
- Простота використання. Node.js не ускладнює використання JavaScript чи TypeScript, а розширює їх можливості

Також, було обрано фреймворк Moleculer.js для реалізації мікросервісної архітектури. Його перевагами є:

- Відповідність вимогам сучасності. Фреймворк є доволі новим, підтримує всі необхідні технології і активно підтримується розробниками
- Модульність. Має багато функціоналу, що може бути задіяний за потребою, а без потреби жодних проблем з налаштуванням незадіяного функціоналу не створює.
- Має вбудований HTTP сервер, що є одним з модулів і за потребою може бути застосований
- Підтримка інших сучасних модулів

Крім того, була обрана база даних. Розглядались такі варіанти, як PostgreSQL, MySQL, SQLite, MongoDB та FireBaseDB

Так як буде використано мову зі строгою типізацією, база має добре справлятися з складними запитами, а також база даних має відносно легко розширюватись, то було обрано реляційну базу даних. Їх плюсами є стабільність, продуктивність і швидкодія [19]. Такі бази даних важчі в контексті розширюваності, але однозначним лідером в плані розширюваності [5] є PostgreSQL. Також, її однозначною перевагою є те, що вона безкоштовна

Також варто обрати бібліотеку або фреймворк для створення клієнтської частини веб-додатку. З доступних варіантів є React.js, Vue.js і Angular

Перевагами React.js є функціональний і компонентний підхід, гнучкість, значні обсяги документації, значні можливості щодо перевикористання коду [6]

Щодо Angular, його перевагами є вбудованість багатьох важливих функцій, таких як маршрутизація, управління станами, тестування. Крім того, цей фреймворк існує лише для мови TypeScript і з самого початку створювався саме для неї.

Vue.js пропонує швидкодію, легкість і гнучкість у використанні, але не є настільки популярним як дві опції, зазначені вище

Враховуючи вказані вище переваги, а також власний попередній досвід, було обрано саме React.js в силу гнучкості, компонентного підходу і хорошої роботи з TypeScript

Також була потреба у розробці Telegram-бота. Варіантів тут не настільки багато, серед них node-telegram-api, Grammy і Telegraph. Нормальну підтримку мови TypeScript серед них може надати лише Grammy, тому було обрано саме цей модуль [8].

Висновки до розділу 1

Був проведений аналіз існуючих систем, виявлено їх плюси і недоліки, а відповідно і вимоги до системи що створюється. Це гнучкість, розширюваність, можливість обговорювати і писати статті, і в цілому відкритість системи до користувача. А також можливість робити запити в базу даних SIMBAD не має аналогів, окрім власне сайту SIMBAD

Також було обрано технології для використання для створення системи. Це мова TypeScript мікросервісний фреймворк Moleculer.js, база даних PostgreSQL, бібліотека для клієнтської частини React.js, і модуль для Telegram-бота Grammy

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Визначення варіантів використання та структури системи.

В цьому розділі представлено аналіз вимог до програмного продукту, виявлено функціональні та нефункціональні вимоги. Також, було визначено вимоги до часу відгуку на дії користувача і окреслено варіанти використання системи

Почати варто з вимог користувачів

Не авторизовані користувачі мають доступ до наступних функцій:

- Переглядати статті
- Шукати статті
- Читати статті
- Переглядати коментарі
- Логін
- Реєстрація

Авторизовані користувачі мають доступ до наступних функцій:

- Додавати коментарі
- Скаржитись на коментарі
- Писати статті
- Редагувати статті (до публікації)

Користувачі адміністративної частини, в залежності від ролі, мають доступ до наступних функцій:

Роль “Автор”

- Переглядати ще не опубліковані статті
- Лишати зауваження до не опублікованих статей
- Видаляти статті
- Публікувати статті

Роль “Модератор”

- Переглядати скарги на коментарі
- Видаляти скарги

					ІПЗ.КР.Б – 121 – 23 – ПЗ	
						23

- Видаляти коментарі
- Видаляти користувачів

Роль “Адміністратор”

- Переглядати і шукати користувачів
- Змінювати ролі користувачів

Також, має сенс розписати вимоги до Telegram-бота. Користувач повинен мати можливість:

- Робити запити в базу даних SIMBAD
- Отримувати статті, які пов’язані з ідентифікатором об’єкта

Функціональні вимоги:

• Клієнтська та адміністративна частина повинна мати можливість сортування та пошуку, за потребою-фільтрації

- Можливість додання зображень до тексту статті і їх зміни
- Відображення як автора, так і рецензента статті
- Можливість пошуку статті за декількома полями

Нефункціональні вимоги:

- Веб-додаток має працювати в сучасних браузерях
- Telegram-бот має підтримуватись на сучасних пристроях
- Зручний інтерфейс
- Зручний і зрозумілий інтерфейс
- Розширюваність і масштабованість

Щоб задовольнити потреби користувачів, система пропонує різні варіанти використання. Це дозволяє користувачам, залежно від їх ролей, адаптувати систему до власних потреб, відповідно, роблячи її більш ефективною. Тому має сенс створити діаграму варіантів використання (рис. 2.1). На ній будуть присутні всі можливі варіанти користувачів – гість, авторизований користувач, автор, модератор, адміністратор

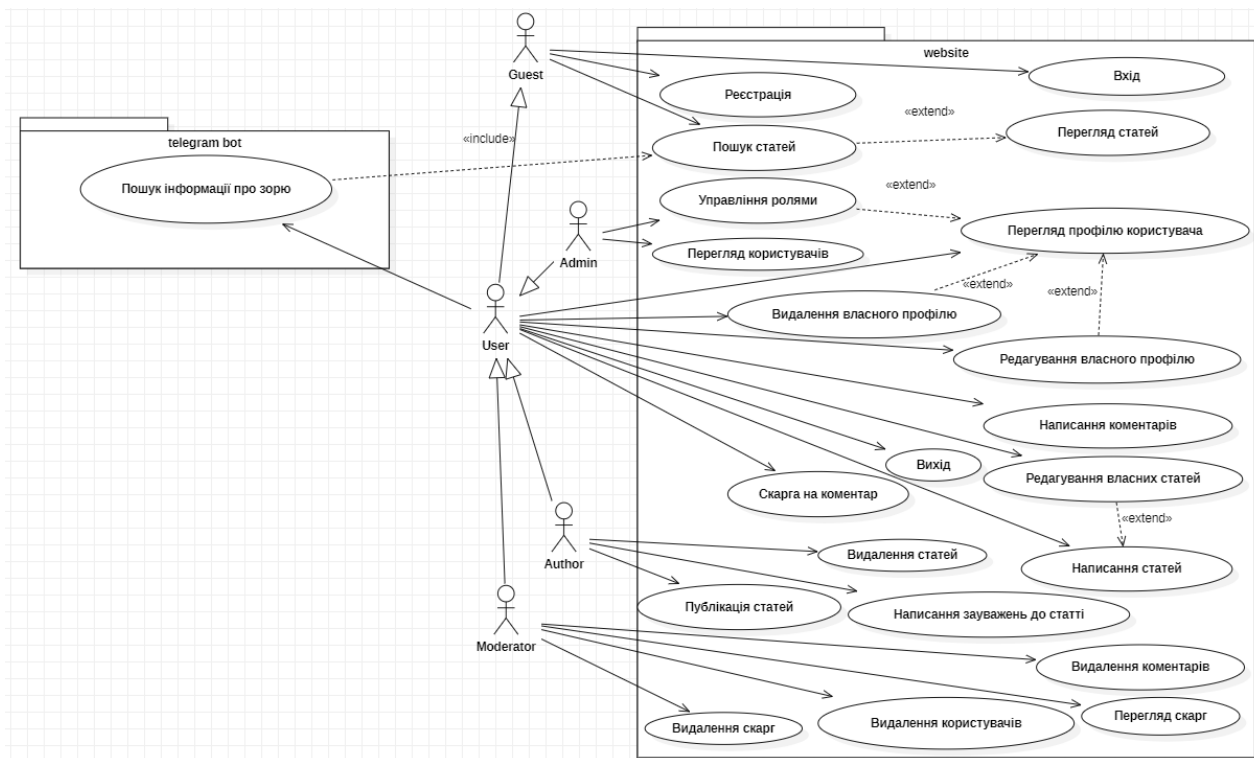


Рис. 2.1. Діаграма варіантів використання

Guest-гість, або неавторизований користувач може увійти, зареєструватись, шукати і переглядати статті.

User-користувач, або авторизований користувач може робити все те, що і неавторизований, плюс писати коментарі, статті, редагувати статті згідно з зауваженнями, а також редагувати власний профіль, або взагалі його видалити.

Author-автор, або користувач адміністративної частини сайту з роллю автор. Може переглядати неопубліковані статті, писати зауваження до них, публікувати статті, а також видаляти їх.

Moderator-модератор, або користувач адміністративної частини сайту з роллю модератор. Може переглядати скарги на коментарі, видаляти коментарі, видаляти скарги, а також видаляти користувачів.

Admin-адміністратор, або користувач адміністративної частини сайту з роллю адміністратор. Може змінювати ролі користувачів, і в цілому переглядати список користувачів.

Доступ до адміністративної частини влаштований наступним чином:

Якщо користувач має адміністративну роль, то для нього будуть одразу

доступні всі необхідні функції. Якщо ж ні, і користувач буде намагатись якось ними скористатись-він побачить повідомлення про помилку і пропозицію авторизуватись як користувач, що має доступ до адміністративної частини.

2.2. Опис структури системи

Серверна частина системи була побудована з використанням мікросервісної архітектури. Це означає що функціонал системи було розподілено між декількома проектами, замість використання лише одного. На діаграмі компонентів системи (рис.2.2) зображено всі мікросервіси, що використовується і їх можливі взаємодії, а також бази даних і сторонні API

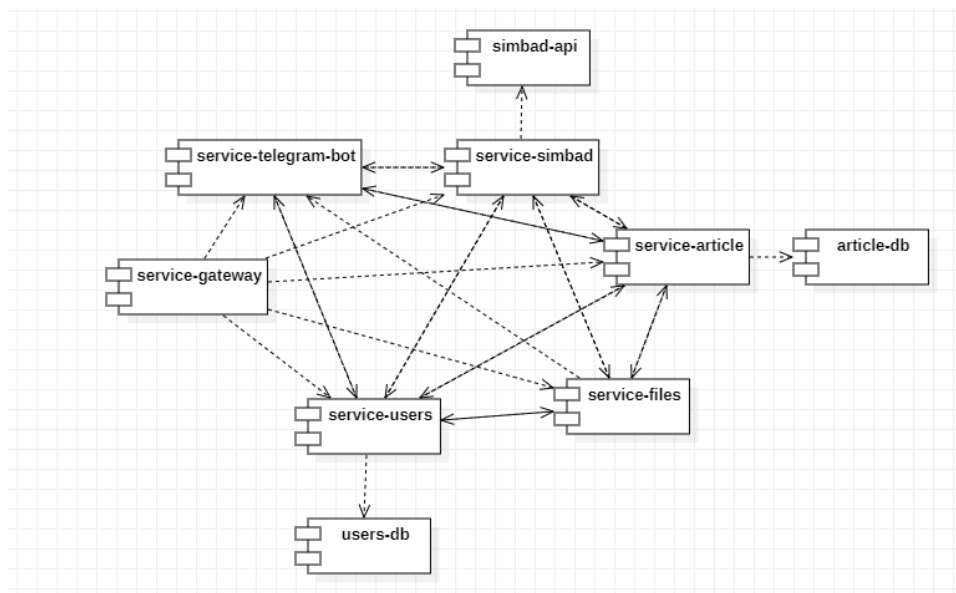


Рис. 2.2 – Діаграма компонентів серверної частини системи

Наявні мікросервіси:

- service-gateway. Сервіс, що відповідає за обробку HTTP запитів і перенаправлення їх до відповідних мікросервісів.
- service-telegram-bot. Сервіс, що відповідає за роботу Telegram-бота.
- service-simbad. Сервіс, що відповідає за запити до бази даних SIMBAD.
- service-article. Сервіс, що відповідає за взаємодію зі статтями, коментарями, скаргами та зауваженнями.
- service-files. Сервіс, що відповідає за завантаження файлів до хмарного сховища AWS S3.

- service-users. Сервіс, що відповідає за функціонал користувачів.

Також при розробці серверної частини було використано об'єктно орієнтовану структуру, тобто кожен елемент системи представлено як об'єкт з конкретними властивостями та методами. Нижче на рис. 2.2 наведено діаграму класів одного з мікросервісів, а саме article, тобто сервіс, що використовується для взаємодії зі статтями. Діаграма є демонстрацією структури цього сервісу, адже всі інші побудовані аналогічно, з рідкими винятками для специфічного функціоналу. На цій діаграмі відсутній клас App, адже він реалізований в модулі package-app, і використовується напрямую звідки, із застосуванням специфічних для даного мікросервісу налаштувань.

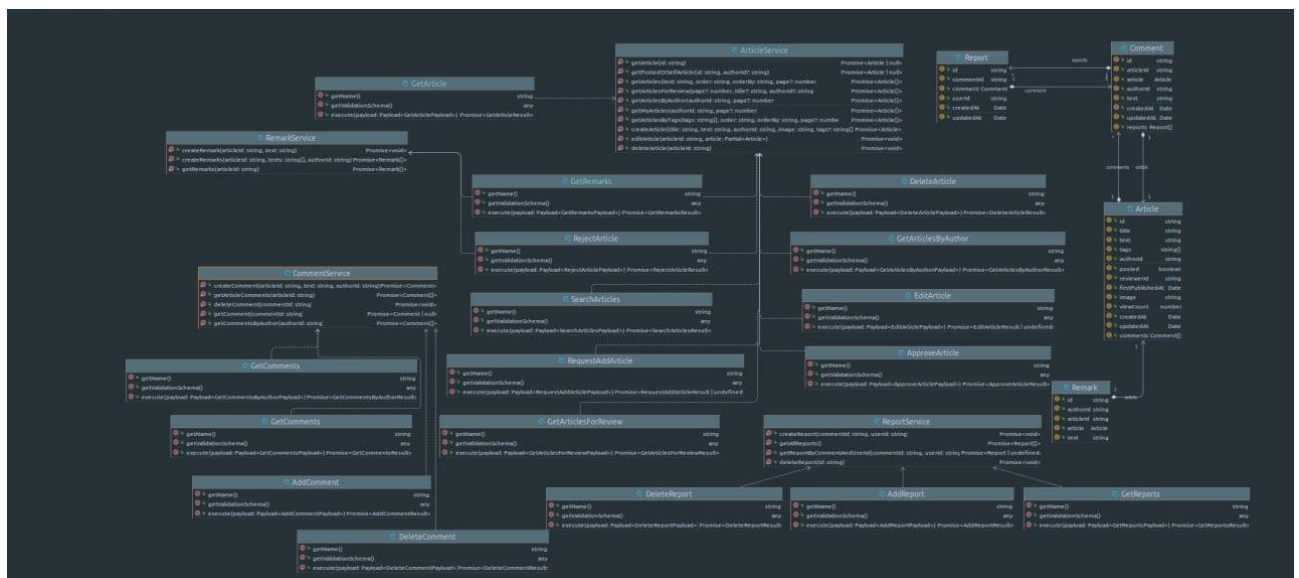


Рис. 2.3 – Діаграма класів мікросервісу article

Клас App, відсутній на діаграмі з вказаних вище причин, реалізує основний функціонал мікросервісу, тобто підключення до бази даних, до інших мікросервісів, комунікація з ними, за потребою-використання HTTP API [18].

Класи, що мають методи getName(), getValidationSchema() і execute() це дії даного мікросервісу, які можуть викликатись з інших мікросервісів

Класи, назви яких закінчуються на ...Service – це класи, які реалізують взаємодію з базою даних для кожної з наявних в мікросервісі сутностей

Класи Article, Report, Remark, Comment – це безпосередньо моделі, які описують відповідні об'єкти і таблиці в базі даних

На клієнтській частині системи було застосовано функціональний підхід, тому діаграму класів сформувати не вдасться. Структура проекту зображена на рис. 2.3.

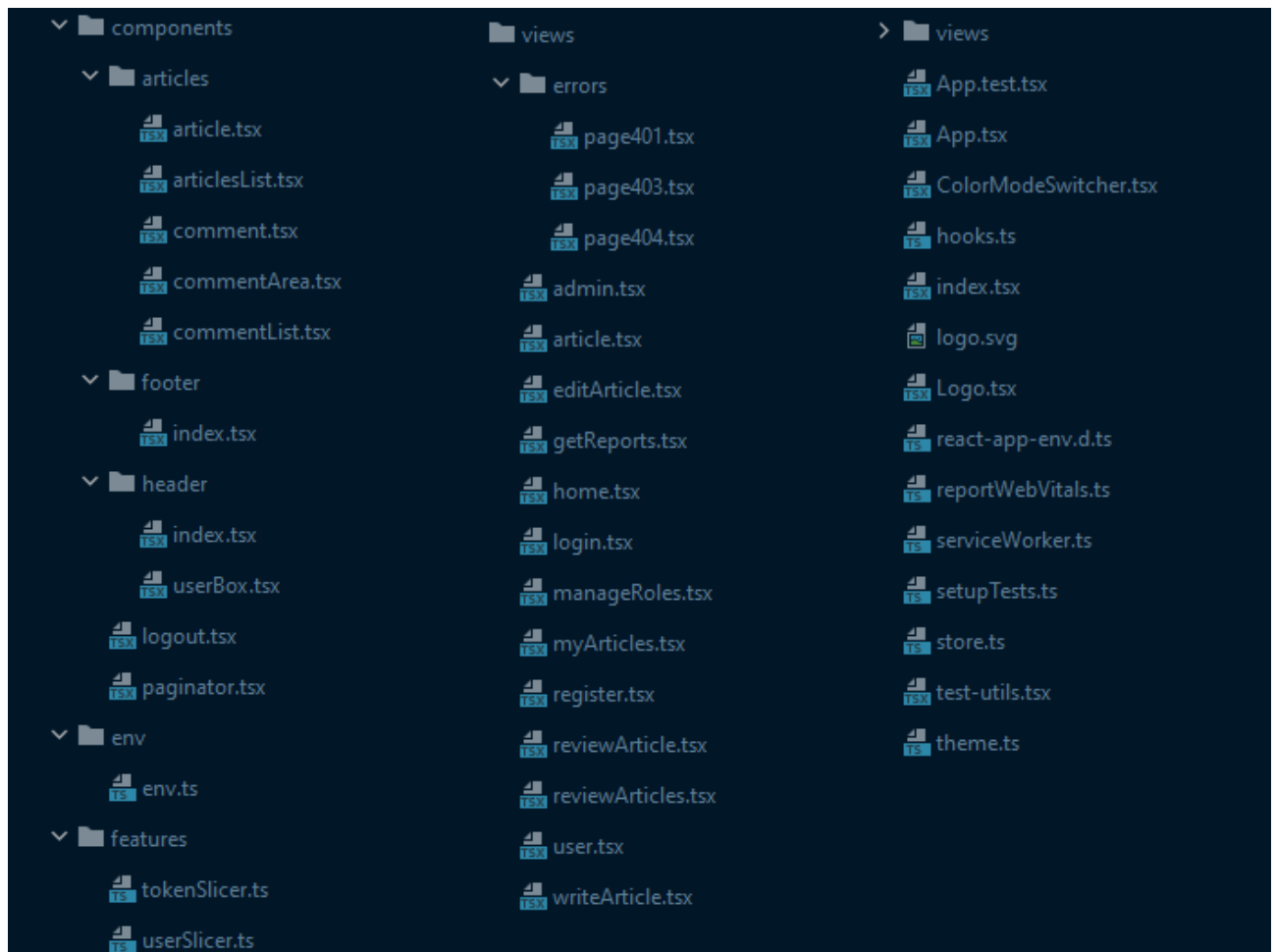


Рис. 2.4 – Структура клієнтської частини системи

В папці components знаходяться компоненти, а саме частини проекту, які використовуються одразу в декількох місцях

В папці env знаходять налаштування проекту

В папці features знаходяться необхідні для роботи бібліотеки Redux методи

В папці views знаходять веб-сторінки проекту

Файл App.tsx є основним файлом проекту, з якого починається його робота

Файли ColorModeSwitcher.tsx і theme.tsx необхідні для роботи графічного фреймворку Chakra UI

Файли hooks.ts і store.ts необхідні для роботи Redux [11]

2.3. Проектування баз даних системи

Система використовує дві бази даних, users і articles.

База даних users має дві таблиці: users і tokens.

На рисунку (2.5) зображено діаграму таблиць бази даних users

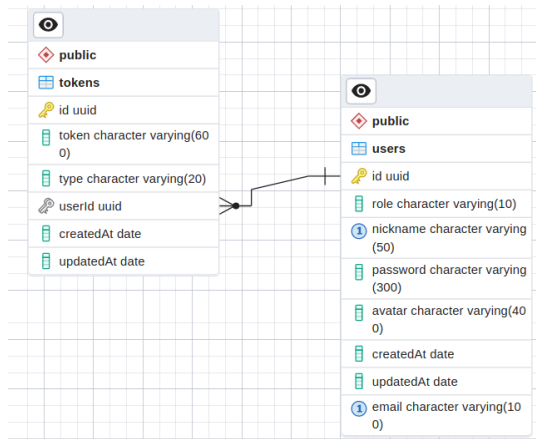


Рис. 2.5 – Діаграма таблиць бази даних users

Таблиця users містить наступні поля:

Таблиця 2.1

Назва	Тип	PK	FK	Опис
id	uuid	+		Ідентифікатор користувача
role	varchar(10)			Роль користувача
nickname	varchar(50)			Нікнейм користувача
password	varchar(300)			Хешований пароль
avatar	varchar(400)			Аватар користувача
createdAt	date			Дата створення профіля
updatedAt	date			Дата останнього оновлення профіля
email	varchar(100)			Токен скидання

Таблиця tokens містить наступні поля:

Таблиця 2.2

Назва	Тип	PK	FK	Опис
id	uuid	+		Ідентифікатор токена
token	varchar(600)			Безпосередньо токен
type	varchar(20)			Тип токена, access або refresh

userId	uuid		+	Ідентифікатор користувача, якому належить токен
createdAt	date			Дата створення токена
updatedAt	date			Дата останньої зміни токена

База даних articles містить чотири таблиці: articles, comments, reports і remarks

На рисунку (2.6) зображено діаграму таблиць бази даних articles

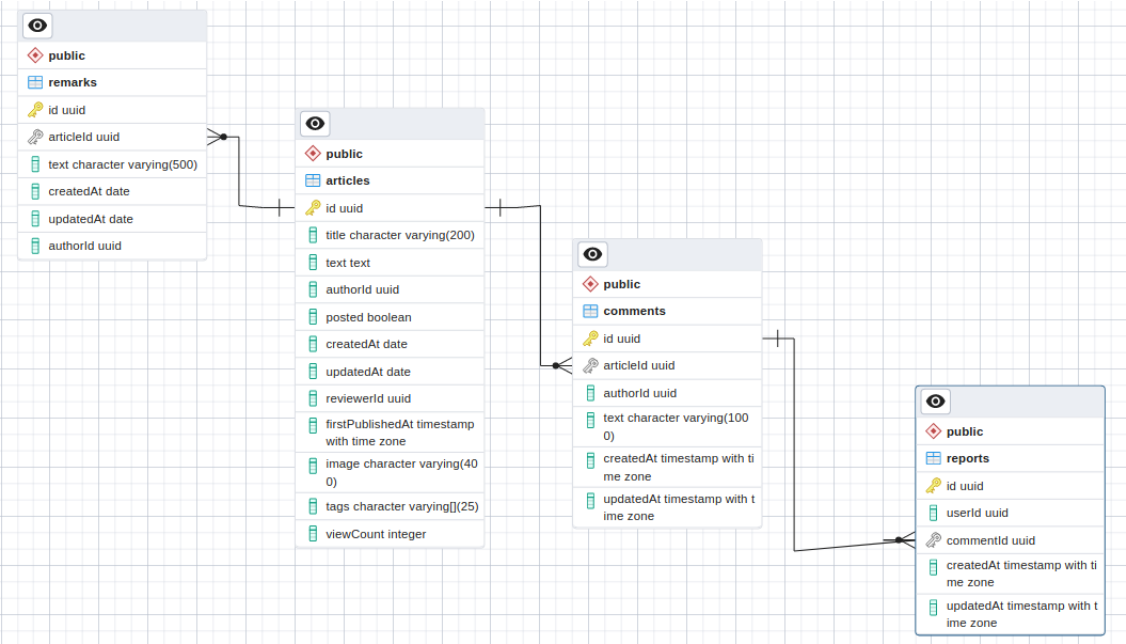


Рис. 2.6. Діаграма таблиць бази даних articles

Таблиця articles містить наступні поля:

- id-ідентифікатор статті
- title-заголовок статті
- text-текст статті
- authorId-ідентифікатор користувача-автора статті
- posted-чи опублікована стаття
- createdAt-дата створення статті
- updatedAt-дата останнього оновлення статті
- reviewerId-ідентифікатор користувача-рецензента статті
- firstPublishedAt-дата першої публікації статті
- image-головне зображення статті

- tags-ключові слова статті
- viewCount-лічильник переглядів статті

Таблиця articles містить наступні поля:

Таблиця 2.3

Назва	Тип	PK	FK	Опис
id	uuid	+		Ідентифікатор статті
title	varchar(200)			Заголовок
text	text			Текст
authorId	uuid			Ідентифікатор автора
posted	boolean			Чи опублікована стаття
createdAt	date			Дата створення
updatedAt	date			Дата останньої зміни
reviewerId	uuid			Ідентифікатор рецензента
firstPublishedAt	date			Дата першої публікації
image	varchar(400)			Шлях до головного зображення
tags	varchar[](125)			Ключові слова
viewCount	integer			Лічильник переглядів

Таблиця comments містить наступні поля

Таблиця 2.4

Назва	Тип	PK	FK	Опис
id	uuid	+		Ідентифікатор коментаря
articleId	uuid		+	Ідентифікатор статті, до якої написано коментар
authorId	uuid			Ідентифікатор автора коментаря
text	varchar(1000)			Текст коментаря
createdAt	date			Дата створення
updatedAt	date			Дата останньої зміни

Таблиця remarks містить наступні поля:

Таблиця 2.5

Назва	Тип	PK	FK	Опис
id	uuid	+		Ідентифікатор зауваження
articleId	uuid		+	Ідентифікатор статті, до якої написано зауваження
authorId	uuid			Ідентифікатор автора зауваження
text	varchar(500)			Текст зауваження
createdAt	date			Дата створення
updatedAt	date			Дата останньої зміни

Таблиця reports містить наступні поля

Таблиця 2.6

Назва	Тип	PK	FK	Опис
id	uuid	+		Ідентифікатор скарги
userId	uuid			Ідентифікатор користувача, що поскаржився
commentId	uuid		+	Ідентифікатор коментаря
createdAt	date			Дата створення
updatedAt	date			Дата останньої зміни

2.4. Реалізація інформаційної екосистеми Gamma, статей, коментарів, зауважень, скарг та користувачів

Серверна частина системи була розроблена з дотриманням принципів REST [9] на мові програмування TypeScript, з застосуванням фреймворку Moleculer.js. Було написано ядро, а також додатковий модуль з типами даних, які є необхідними для роботи системи. Дані між мікросервісами і між клієнтом і сервером передаються у форматі JSON. Для взаємодії з базою даних було обрано ORM Sequelize [7].

Для початку варто ознайомитись з будовою ядра системи. Воно знаходиться в модулі package-app

```
import { ServiceActionsSchema, ServiceBroker, ServiceSchema } from 'moleculer';
import { Payload, ServiceConfig } from './interfaces';
import * as path from 'path';
import { ActionPayload, ActionResult, ServiceName } from 'package-types';
import * as fs from 'fs';
import recursiveRead from 'recursive-readdir';
import { Sequelize } from "sequelize-typescript";

export * from './interfaces';
export * from './types';

export class App {
  private broker: ServiceBroker;
  private static instance: App;
  private dbConnection?: Sequelize;
  private readonly config: ServiceConfig;

  constructor() {
    this.config = this.getConfig();
    this.broker = new ServiceBroker({ ...this.config.brokerConfig });
    if (this.config.dbConnection) {
      const { password, username, database, host, port, dialect } =
this.config.dbConnection;
      this.dbConnection = new Sequelize(database, username, password, { host, port,
dialect });
    }
  }

  public static getInstance(): App {
    if (!App.instance) {
      App.instance = new App();
      process.on('SIGINT', App.instance.onShutDown);
      process.on('SIGQUIT', App.instance.onShutDown);
      process.on('SIGTERM', App.instance.onShutDown);

      process.on('uncaughtException', (err: any): void => {
        App.logError('Uncaught exception', err, { msgCode: 'UNCAUGHT_EXCEPTION'
});
      });

      process.on('unhandledRejection', (err: any): void => {
        App.logError('Unhandled rejection', err, { msgCode: 'UNHANDLED_REJECTION'
});
      });
    }
    return this.instance;
  }

  public async run(serviceSchema: ServiceSchema): Promise<void> {
    try {
      serviceSchema.actions = await this.initActions();
      this.broker.createService(serviceSchema);
      await this.broker.start();
    } catch (err) {
      App.logError(err);
      throw err;
    }
  }

  private getConfig(): ServiceConfig {
    const configPath = path.resolve(process.cwd(), 'dist/env/env.js');
    return require(configPath).config;
  }

  private async onShutDown(err: any): Promise<void> {
    if (err) {

```

```

        App.logError('Unexpected shutdown', err, { msgCode: 'UNEXPECTED_SHUTDOWN' });
    }
    App.logDebug('Stop service broker');
    await App.getInstance().broker.stop();
    await this.dbConnection?.close();

    setImmediate(() => process.exit(1));
}

public static async call<P, R>(serviceName: ServiceName, actionName: string, params:
P): Promise<R> {
    const { broker } = App.getInstance();
    try {
        return broker.call(`${serviceName}.${actionName}`, params);
    } catch (err) {
        App.logError(`Error while calling ${serviceName}.${actionName}. Error:
${err}`);
        throw err;
    }
}

public static logDebug(...args: any[]) {
    App.getInstance().broker.logger.debug(...args);
}

public static logInfo(...args: any[]) {
    App.getInstance().broker.logger.info(...args);
}

public static logWarn(...args: any[]) {
    App.getInstance().broker.logger.warn(args);
}

public static logError(...args: any[]) {
    App.getInstance().broker.logger.error(...args);
}

public static logFatal(...args: any[]) {
    App.getInstance().broker.logger.fatal(...args);
}

public static logTrace(...args: any[]) {
    App.getInstance().broker.logger.trace(...args);
}

private async initActions(): Promise<ServiceActionsSchema> {
    try {
        const actionsPath = path.resolve(process.cwd(), 'dist/actions');
        if(!fs.existsSync(actionsPath)) {
            return {};
        }
        const files: string[] = path ? await recursiveRead(actionsPath, ['*.map']) :
[];

        const actions: ServiceActionsSchema = {};
        files.forEach((fileName: string): void => {
            const action = require(path.resolve(fileName)).default;
            const schema = action.getValidationSchema();
            async function handler(ctx: Payload<ActionPayload>):
Promise<ActionResult> {
                App.logInfo(`Received call ${action.getName()}. Executing...`,
ctx.params);
                try {
                    const { currentUser } = ctx.params;
                    const result = await action.execute(ctx);
                    App.logInfo(`Received call ${action.getName()}. Done!`, result);
                    if(currentUser) {
                        result.currentUser = currentUser;
                    }
                    return result;
                } catch (err) {
                    App.logError(`Error while executing action ${action.getName()}`,

```

```

err);
        throw err;
    }

    }
    const name = action.getName();
    actions[name] = {
        params: schema,
        //@ts-ignore
        handler
    }
    App.logInfo(`Action ${action.getName()} initialized`);
});
return actions;
} catch (err) {
    App.logError(err);
    throw err;
}
}

public getDBConnection(): Sequelize {
    if(this.dbConnection) {
        return this.dbConnection;
    } else throw new Error('No db here');
}
}

```

Поля цього класу:

- **broker** – інструмент, завдяки якому відбувається комунікація між мікросервісами

- **instance** – екземпляр класу App, адже тут реалізовано шаблон Singleton
- **dbConnection** – об'єкт підключення до бази даних, якщо воно існує
- **config** – об'єкт налаштувань сервіса

Методи класу:

- **getInstance()** – створює або повертає екземпляр класу App. Він необхідний для роботи шаблону Singleton

- **run()** – запускає мікросервіс
- **getConfig()** – зчитує конфігурацію з файлу
- **onShutDown()** – обробляє зупинку роботи сервіса
- **call()** – відповідає за виклик дії на іншому мікросервісі, і виконує всі необхідні підготовчі дії, і дії після повернення результату

- **logDebug(), logInfo(), logWarn(), logError(), logFatal(), logTrace()** – методи, які відповідають за запис інформації про роботу мікросервіса

- **initActions()** – ініціалізує дії, що знаходяться в папці actions
- **getDBConnection()** – повертає екземпляр підключення до бази даних

Для того, щоб запустити мікросервіс з використанням цього ядра,
достатньо виконати наступний код:

```
import {App} from 'package-app';
import { config } from './env/env.js';
import Article from './models/article';
import Remark from './models/remark';
import Comment from './models/comment';
import Report from './models/report';

const app = App.getInstance();

async function main(): Promise<void> {
  await app.run({
    name: config.name,
  });
}

app.getDBConnection().addModels([Article, Comment, Remark, Report]);
main().then();
```

Код вище запускає сервіс article.

Для зручної роботи з базою даних необхідними є моделі, в яких описано
всі ті ж поля, які є в базі даних, плюс додаткові поля для реалізації зв'язків

Модель статті, створена за допомогою Sequelize виглядає наступним

чином

```
import Comment from './comment'
import {Column, PrimaryKey, Table, Model, DataType, Default, AllowNull, Index, HasMany}
from "sequelize-typescript";

@Table({tableName: 'articles'})
class Article extends Model<Article> {
  @PrimaryKey
  @Column({
    type: DataType.UUIDV4,
    defaultValue: DataType.UUIDV4
  })
  id: string;

  @Index('article_search_index')
  @Column(DataType.CHAR(200))
  title: string;

  @Index('article_search_index')
  @Column(DataType.TEXT)
  text: string;

  @Column(DataType.ARRAY(DataType.STRING(25)))
  tags: string[];

  @Column(DataType.UUIDV4)
  authorId: string;

  @Default(false)
  @Column(DataType.BOOLEAN)
  posted: boolean;

  @AllowNull
  @Column(DataType.UUIDV4)
  reviewerId?: string;

  @AllowNull
```

```

@Column(DataType.DATE)
firstPublishedAt?: Date;

@Column(DataType.CHAR(400))
image: string;

@Column(DataType.NUMBER)
viewCount: number;

@Column(DataType.DATE)
createdAt: Date;

@AllowNull
@Column(DataType.DATE)
updatedAt?: Date;

@HasMany(() => Comment)
comments?: Comment[];
}
export default Article

```

Було створено і інші моделі, наприклад модель коментаря:

```

import Article from "../article";
import {
  Column,
  PrimaryKey,
  Model,
  DataType,
  Table,
  ForeignKey,
  AllowNull,
  BelongsTo, HasMany
} from "sequelize-typescript";
import Report from "../report";

@Table({tableName: 'comments'})
class Comment extends Model<Partial<Comment>> {
  @PrimaryKey
  @Column({
    type: DataType.UUIDV4,
    defaultValue: DataType.UUIDV4
  })
  id: string;

  @Column(DataType.UUIDV4)
  @ForeignKey(() => Article)
  articleId: string;

  @BelongsTo(() => Article)
  article: Article;

  @Column(DataType.UUIDV4)
  authorId: string;

  @Column(DataType.CHAR(1000))
  text: string;

  @Column(DataType.DATE)
  createdAt: Date;

  @AllowNull
  @Column(DataType.DATE)
  updatedAt?: Date;

  @HasMany(() => Report)
  reports: Report[];
}
export default Comment;

```

Ці і інші моделі наявні в додатку А

Для демонстрації принципу роботи HTTP API має сенс показати повний процес, який проходить система для виконання певної дії. Було обрано процес додання статті

Починається все з маршруту.

```
{
  path: '/article/proposeArticle',
  method: HttpMethod.POST,
  role: Role.USER,
  actionName: ArticleActionName.RequestAddArticle,
  serviceName: ServiceName.Article,
  middlewares: [fileUpload]
},
```

- path – відносна URL, сам маршрут
- method- HTTP метод
- role – необхідна роль користувача для доступу до цієї дії
- actionName – назва дії, що обробляє даний маршрут
- serviceName – назва мікросервісу, який містить потрібну дію
- middlewares – проміжні дії

Так як масив з проміжними діями не пустий, буде виконана дія з завантаження файлів, а також дія, що аутентифікує користувача, яка за замовчуванням присутня в усіх маршрутах

Аутентифікація:

```
import {config} from "../env/env";
import {App} from "package-app";
import {AuthPayload, AuthResult, ServiceName, UsersActionName} from "package-types";

export async function authenticate (req: any, res: any, next: any) {
  const access: string = req.headers['authorization'];
  req.$params.currentUser = undefined;
  if(access) {
    if(!config.accessTokenSecret) {
      throw new Error('No secret word for authentication provided!');
    }
    try {
      const result = await App.call<AuthPayload, AuthResult>(ServiceName.Users,
        UsersActionName.Auth, { access });
      if(!result.currentUser) {
        throw new Error('Unauthorized');
      }
      req.$params.currentUser = result.currentUser;
    } catch (err) {
      App.logError(err);
      res.statusCode = 401;
    }
  }
  next();
}
```

Дана проміжна дія перевіряє чи авторизований користувач, і додає до об'єкта запита далі об'єкт поточного користувача. Дія не потребує адміністративної ролі, тому перевірка на доступ непотрібна.

Далі можна перейти до дії безпосередньо на мікросервісі article

```
import {Action, App, Payload} from 'package-app';
import {
  ArticleActionName, FilesActionName, RequestAddArticlePayload,
  RequestAddArticleResult, ServiceName, UploadFilesPayload, UploadFilesResult
} from "package-types";
import {ArticleService} from "../services/article";
import {CurrentUserSchema} from "package-types/dist/validationSchemas/currentUser";
import {setImagesToArticleText} from "../helpers/setImagesToArticleText";

export default new class RequestAddArticle implements Action{
  getName(): string{
    return ArticleActionName.RequestAddArticle;
  }

  getValidationSchema(): any {
    return {
      title: {
        type: 'string',
        max: 200
      },
      text: {
        type: 'string'
      },
      tags: {
        type: 'array',
        items: {
          type: 'string',
          max: 50
        }
      },
      currentUser: {
        type: 'object',
        props: CurrentUserSchema
      },
      image: {
        type: 'object'
      }
    };
  }

  async execute(payload: Payload<RequestAddArticlePayload>):
  Promise<RequestAddArticleResult | undefined> {
    const { title, text, tags, currentUser, image} = payload.params;
    try {
      const { fileKeys } = await App.call<UploadFilesPayload,
      UploadFilesResult>(ServiceName.Files, FilesActionName.UploadFiles, {files: {articleImage:
      image}});
      const {articleImage} = fileKeys;
      if(!articleImage) {
        throw new Error(`Unable to upload file ${image}`);
      }
      const textWithImages = await setImagesToArticleText(text);
      const article = await ArticleService.createArticle(title, textWithImages,
      currentUser.id, articleImage, tags);
      return { article };
    } catch (err) {
      App.logError(err);
      return ;
    }
  }
}
```

Типова дія мікросервіса виглядає саме так. Є три метода

- `getName()` – повертає назву дії, що необхідно для ініціалізації дій
- `getValidationSchema()` – повертає валідаційну схему вхідних даних
- `execute()` – виконує роботу. Розглянемо цей метод ближче

Як бачимо, всі вхідні дані знаходяться в `payload.params`, незалежно від того в якій частині HTTP запита вони були передані. Далі робиться запит на мікросервіс `files`, для завантаження основного зображення статті на хмарне сховище. Після цього – робота з текстом, оптимізація зображень в ньому, за потребою – їх завантаження на хмарне сховище. А вже після цього стаття додається в базу даних, і вже створений об'єкт повертається назад, на інтерфейс користувача.

Також має сенс продемонструвати метод, який завантажує файл до хмарного сховища і є частиною мікросервіса `files`

```
public static async uploadFile(file: UploadedFile): Promise<string> {
  if(!allowedFileExtensions.includes(file.name.split('.').pop())) {
    throw new Error(`image extension is not supported ${file}`);
  }
  const {name, data, mimetype} = file;
  const uploadPath = fileNameBuilder(name);
  const dataBuffer = Buffer.from(data.data);
  const uploadedFile = await bucket.upload({
    Bucket: config.s3Config.s3Name,
    Body: dataBuffer,
    Key: uploadPath,
    ContentType: mimetype
  }).promise();
  return uploadedFile.Location;
}
```

Приймається файл, створюється його ім'я, у якості якого використовується звичайний `uuid`, створюємо `Buffer`, тобто код файла, і завантажуюмо його за допомогою модуля `s3`. Після чого повертаємо посилання на файл, яке в майбутньому буде використано для відображення зображення на клієнтській частині

Більш детальна маршрутизація продемонстрована в таблиці

У табл. 2.7 представлено основні кінцеві точки для статей та дії, які вони виконують.

					ІПЗ.КР.Б – 121 – 23 – ІЗ	
						40

Кінцеві точки для статей

Кінцева точка	HTTP-метод	Роль	Опис
/article/proposeArticle	POST	Користувач	Додає статтю до не опублікованих
/article/getArticlesForReview	GET	Автор	Повертає список не опублікованих статей
/article/approoveArticle	POST	Автор	Публікує статтю
/article/editArticle/:articleId	PUT	Користувач	Редагує статтю,
/article/rejectArticle	POST	Автор	Відмовляє у публікації і додає зауваження
/article/getArticlesByAuthor	GET	Гість	Повертає список статей, написаних конкретним користувачем
/article/getArticle/:id	GET	Гість	Повертає одну статтю за ідентифікатором
/article/delete/:articleId	DELETE	Автор	Видаляє статтю
/article/comment	POST	Користувач	Додає коментар до статті
/article/deleteComment/:commentId	DELETE	Модератор	Видаляє коментар
/article/reportComment/:commentId	POST	Користувач	Додає скаргу на коментар
/article/getReports	GET	Модератор	Повертає список скарг

/article/deleteReport/:reportId	DELETE	Модератор	Видаляє скаргу
/article/getComments/:articleId	GET	Гість	Повертає список коментарів до статті
/article/searchArticles	GET	Гість	Повертає список статей за текстом, заголовком або ключовими словами
/article/getCommentsByAuthor	GET	Гість	Повертає список коментарів за ідентифікатором автора

Також нижче представлена маршрутизація, що стосується користувачів

Таблиця 2.8

Кінцеві точки для користувачів

Кінцева точка	HTTP-метод	Роль	Опис
/users/login	POST	Гість	Вхід у профіль
/users/register	POST	Гість	Реєстрація нового акаунта
/users/refresh	POST	Гість	Оновлює токени для аутентифікації
/users/auth	GET	Гість	Аутентифікує користувача
/users/:userId	GET	Користувач	Повертає користувача
/users/changeAvatar	PUT	Користувач	Змінює аватар користувача
/users/changeNickname	PUT	Користувач	Змінює нікнейм користувача

/users/changePassword	PUT	Користувач	Змінює пароль
/users/clearTokens	DELETE	Користувач	Очищає токени
/users/deleteUser/:userId	DELETE	Користувач	Видаляє користувача
/users/getUsersByRole	GET	Адміністратор	Повертає список користувачів за роллю і нікнеймом
/users/changeRole	PUT	Адміністратор	Змінює роль користувача

Крім того, варто розглянути роботу Telegram-бота

Почати варто з ініціалізації команд бота. За це відповідає функція `initCommands()`

```
export default async function initCommands(telegramToken: string) {
  type SessionData = ScenesSessionFlavor & {
  }

  type BotContext = Context & SessionFlavor<SessionData> & ScenesFlavor

  const bot = new Bot<ParseModeFlavor<BotContext>>(telegramToken);
  bot.api.config.use(parseMode("HTML"));

  bot.use(hydrateReply);
  bot.use(
    session({
      initial: () => ({}),
    })
  )

  const scenes = new ScenesComposer<BotContext>(getObjectByIdScene)
  bot.use(scenes.manager())

  bot.command("start", async (ctx) => {
    await ctx.reply(getText(TextShortcut.Greeting), {
      reply_markup: {
        inline_keyboard: [
          [{text: getText(TextShortcut.IdSearch), callback_data:
BotCommand.StartSearchById}],
        ]
      }
    });
  });

  bot.callbackQuery(BotCommand.StartSearchById, async (ctx) => {
    await ctx.answerCallbackQuery();
    await ctx.scenes.enter(BotScene.GetObjectById);
  });

  bot.use(scenes);

  bot.start();
}
```

Відбувається ініціалізація основних команд і сцен, і безпосередньо запуск бота

Далі продемонстровано сцену `GetObjectById`. Вона відправляє запит до мікросервісу `simbad`, який робить запит у базу даних `SIMBAD` [16], після чого запитує у сервісу `article` щодо статей, які можуть бути пов'язані з цим об'єктом.

Бот отримує статті і дані про об'єкт, і відсилає їх користувачу

```
import {Scene, ScenesFlavor, ScenesSessionFlavor} from "grammy-scenes"
import {Context, InlineKeyboard, SessionFlavor} from "grammy";
import {BotCommand, BotScene} from "../commands/command";
import {getText, getMessageFromSimbadObject} from "../text";
import {TextShortcut} from "../text/textShortcuts";
import {App} from 'package-app'
import {SimbadActionName, GetObjectByIdPayload, GetObjectByIdResult, ServiceName} from
"package-types";
import {getArticleLinks, getArticleTitles} from "../helpers/getArticleLinks";

type SessionData = ScenesSessionFlavor & { };
type BotContext = Context & SessionFlavor<SessionData> & ScenesFlavor
export const getObjectByIdScene = new Scene<BotContext>(BotScene.GetObjectById);

getObjectByIdScene.use(async (ctx, next) => {
  console.log("Entering main scene...");
  await ctx.reply(getText(TextShortcut.IdSearchStart));
  return next();
})

getObjectByIdScene.wait().on("message:text", async (ctx) => {
  const identifier = ctx.message.text;
  const lang = ctx.from.language_code;
  const searchStarted = getText(TextShortcut.IdSearchStarted);
  await ctx.reply(searchStarted);
  const {object, articles} = await App.call<GetObjectByIdPayload,
GetObjectByIdResult>(ServiceName.Simbad, SimbadActionName.GetObjectById, {identifier});
  if(Object.keys(object).length === 0) {
    await ctx.reply('Об'єкт не знайдено!');
    const keyboard = new InlineKeyboard();
    keyboard.text(getText(TextShortcut.IdSearch), BotCommand.StartSearchById);
    await ctx.reply('Шукати ще?', {reply_markup: keyboard});
    return;
  }
  const message = getMessageFromSimbadObject(object);
  await ctx.reply(message);
  const articleLinks = getArticleLinks(articles);
  const articleText = getArticleTitles(articles);
  if(articleLinks.length > 0 && articleText) {
    const keyboard = new InlineKeyboard();
    articleLinks.forEach((link, index) => {
      keyboard.url(`${index}`, link);
    });
    keyboard.text(getText(TextShortcut.IdSearch), BotCommand.StartSearchById);
    await ctx.reply(articleText, {reply_markup: keyboard});
  } else {
    const keyboard = new InlineKeyboard();
    keyboard.text(getText(TextShortcut.IdSearch), BotCommand.StartSearchById);
    await ctx.reply('Шукати ще?', {reply_markup: keyboard});
  }
});
```

Клієнтська частина системи розроблена за допомогою бібліотеки `React.js`, а також менеджера станів `Redux` [10] і маршрутизатора `react-router-dom` [12]. Комунікація з сервером відбувається за допомогою `axios`. Також було створено проміжні дії перед і після запиту, які необхідні для коректної роботи

аутентифікації А за візуальну частину відповідає фреймворк Chakra UI [14]. Було створено файл App.tsx, в якому підключаються інші сторінки, маршрути, і ініціалізуються необхідні залежності

Файл App.tsx виглядає наступним чином:

```
import * as React from "react"
import {
  BrowserRouter,
  Route,
  Routes
} from "react-router-dom";
// @ts-ignore
import {
  ChakraProvider,
} from "@chakra-ui/react"
import theme from './theme'
import Header from "./components/header";
import Footer from "./components/footer";
import Home from "./views/home";
import Login from "./views/login";
import Register from "./views/register";
import {Provider} from 'react-redux';
import {store} from "./store";
import Logout from "./components/logout";
import Admin from "./views/admin";
import WriteArticle from "./views/writeArticle";
import ReviewArticles from "./views/reviewArticles";
import ReviewArticle from "./views/reviewArticle";
import MyArticles from "./views/myArticles";
import ArticleView from "./views/article";
import User from "./views/user";
import GetReports from "./views/getReports";
import EditArticle from "./views/editArticle";
import page404 from "./views/errors/page404";
import page403 from "./views/errors/page403";
import page401 from "./views/errors/page401";
import ManageRoles from "./views/manageRoles";
import {useEffect} from "react";

export const App: React.FC = (): JSX.Element => {

  useEffect(() => {
    document.title = 'gamma';
  })

  return (
    <ChakraProvider theme={theme}>
      <BrowserRouter>
        <Provider store={store}>
          <Header/>
          <Routes>
            <Route path="/" Component={Home}/>
            <Route path="/articles/:id" Component={ArticleView}/>
            <Route path="/users/login" Component={Login}/>
            <Route path="/users/register" Component={Register}/>
            <Route path="/users/logout" Component={Logout}/>
            <Route path="/users/:userId" Component={User}/>
            <Route path="/admin" Component={Admin}/>
            <Route path="/admin/manageRoles" Component={ManageRoles}/>
            <Route path="/articles/writeArticle" Component={WriteArticle}/>
            <Route path="/admin/reviewArticles" Component={ReviewArticles}/>
            <Route path="/admin/getReports" Component={GetReports}/>
            <Route path="/articles/review/:id" Component={ReviewArticle}/>
            <Route path="/articles/myArticles" Component={MyArticles}/>
            <Route path="/articles/edit/:articleId" Component={EditArticle}/>
            <Route path="/error404" Component={page404} />
          </Routes>
        </Provider>
      </BrowserRouter>
    </ChakraProvider>
  )
}
```

```

        <Route path="/error403" Component={page403} />
        <Route path="/error401" Component={page401} />
        <Route path="*" Component={page404} />
      </Routes>
    <Footer />
  </Provider>
</BrowserRouter>
</ChakraProvider>
);
}

```

- ChakraProvider – обгортка, необхідна для роботи UI-фреймворку Chakra UI
- BrowserProvider – обгортка, необхідна для роботи маршрутизатора react-router-dom
- Provider – обгортка, необхідна для роботи менеджера станів Redux
- Routes – маршрути до сторінки

Для початку, має сенс оглянути модифікований axios [17] для запитів на вже створений HTTP API

```

import axios from "axios";
import config from "../env/env";
import {AxiosResponse} from 'axios';
import {Tokens} from "../features/tokenSlicer";
import {RefreshPayload, RefreshResult} from "package-types";
import {store} from "../store";
import {removeUser, setUser} from "../features/userSlicer";

const request = axios.create({
  baseURL: `${config.beUrl}`,
  headers: {
    "Content-type": "multipart/form-data",
  },
});

const refreshTokens = async (): Promise<Tokens | undefined> => {
  try {
    const refreshToken = localStorage.getItem('refresh');
    const {data} = await request.post<RefreshPayload, AxiosResponse<RefreshResult, any>>(`/users/refresh/`, { refresh: refreshToken });
    console.log(data);
    if(!data.tokens) {
      store.dispatch(removeUser());
      return ;
    }
    const { access, refresh } = data.tokens;
    localStorage.setItem('access', access);
    localStorage.setItem('refresh', refresh);
    return data.tokens;
  } catch (err) {
    console.log("Error", err);
  }
};

request.interceptors.request.use(
  async (config) => {
    const token = localStorage.getItem('access');
    if (token) {
      config.headers["Authorization"] = `${token}`;
    }
    return config;
  },
);

```

```

    (error) => {
      return Promise.reject(error);
    }
  );

request.interceptors.response.use(
  (response) => {
    store.dispatch(setUser(response.data.currentUser));
    return response;
  },
  async function (error) {
    try {
      const originalRequest = error.config;
      if (error.response.status === 401 && !originalRequest._retry) {
        originalRequest._retry = true;
        const res = await refreshTokens();
        if (!res?.access || !res?.refresh) {
          throw new Error('Cannot get access and refresh token from server');
        }
        const {access} = res;

        request.defaults.headers.common[
          "Authorization"
        ] = `${access}`;
        return request(originalRequest);
      }
      switch (error.status) {
        }
        return Promise.reject(error);
      } catch (err) {
        console.log(err);
      }
    }
  );
export default request;

```

Важливим елементом є `request.interceptors`. Перший застосовується для додання токенів, що відповідають за аутентифікацію до об'єкта запита, а другий обробляє ситуацію, коли термін дії одного з токенів вийшов, і треба їх оновити, а після цього-повторити запит.

Роботу цього модуля можна побачити наприклад на сторінці `editArticle`, що відповідає за редагування статті. Буде наведено частини цієї сторінки, тому що сама сторінка досить велика Повний код буде наведено в додатку Б

```

const fetchData = async () => {
  const {data} = await request.get<undefined,
  AxiosResponse<GetArticleResult>>(`/article/getArticle/${articleId}`);
  const {article: fetchedArticle} = data;
  setArticle(fetchedArticle);
  if(!fetchedArticle) {
    return (<Navigate to='/users/login' replace/>);
  }
  if (fetchedArticle) {
    setImageURL(fetchedArticle.image);
    setTitle(fetchedArticle.title);
    setText(fetchedArticle.text);
    setTags(fetchedArticle.tags ?? []);
  }
}

```

					ІПЗ.КР.Б – 121 – 23 – ІЗ	
						47

```

const {data: remarksData} = await request.get<undefined,
AxiosResponse<GetRemarksResult>>(`/article/getRemarks/${articleId}`);
setRemarks(remarksData.remarks);
}

```

Ця частина сторінки відповідає за отримання даних про статтю з сервера. Тут використовується показаний вище модель request. У випадку якщо не вдалось знайти статтю-імовірно що користувач не має до неї доступу, тому йому варто зайти в інший профіль. А якщо дані про статтю прийшли-вона записуються в стан компонента. І також, тут отримуються вже існуючі зауваження, які теж відображаються на сторінці, для того щоб користувачеві було простіше розуміти що саме варто змінити в цій статті

Далі буде наведено механізм роботи підтвердження оновлення статті

```

const onClick = async () => {
  const sanitizedTitle = sanitize(title);
  const sanitizedText = sanitize(text, {
    allowedTags: sanitize.defaults.allowedTags.concat([ 'img' ]),
    allowedAttributes: { 'img': [ 'src' ] },
    allowedSchemes: [ 'data', 'http', 'https' ]
  });
  let chosenImage;
  if(image) {
    chosenImage = image;
  } else {
    chosenImage = imageURL;
  }
  try {
    if (sanitizedText && sanitizedTitle) {
      const {data} = await request.put<EditArticlePayload,
AxiosResponse<EditArticleResult>>(`/article/editArticle/${articleId}`, {
        title: sanitizedTitle,
        text: sanitizedText,
        authorId: currentUser.id,
        tags,
        image: chosenImage
      });
      if (data.success) {
        return navigate('/articles/myArticles');
      } else {
        throw new Error('Не вдалось відредагувати статтю');
      }
    }
  } catch (err) {
    console.log(err);
  }
}

```

Варто звернути увагу на декілька речей. Текст і заголовок обробляються з метою унеможливлення надіслання на сервер шкідливого програмного коду. У випадку якщо зображення було змінено, воно буде надіслано в запиті, а якщо змін не було-замість нього буде надіслано посилання на вже існуюче зображення. Після цього надсилається запит, що оновлює дані про статтю, і у

випадку успіху-користувач буде направлений на сторінку з його статтями. Якщо не вдалось оновити статтю-буде відображено повідомлення про помилку

Для використання Redux необхідно було створити декілька модулів, що описують взаємодію з змінними стану

Наприклад, `userSlice`

```
interface UserState {
  value?: SafeUser;
}

const initialState: UserState = {

}

export const userSlice = createSlice({
  name: 'user',
  initialState,
  reducers: {
    setUser: (state, action: PayloadAction<SafeUser>) => {
      state.value = action.payload;
    },
    removeUser: (state) => {
      state.value = undefined;
    },
  },
})

export const { setUser, removeUser } = userSlice.actions

export const selectUser = (state: RootState) => state.user.value

export default userSlice.reducer
```

`userSlice` описує взаємодію з користувачем в стані. Його можна як отримати, якщо він існує, так і видалити

Також, варто продемонструвати принцип роботи компонента `Article`, що відповідає за відображення статті не як елемента списку, а для повноцінного перегляду з доступом до коментарів.

```
import React from "react";
import {Box, Text, Heading, Image, Flex, Link} from '@chakra-ui/react';
import {ViewIcon} from '@chakra-ui/icons';
import {ArticleWithUsers} from "package-types";
import {Link as ReactLink} from "react-router-dom";

interface ArticleProps {
  article?: ArticleWithUsers;
}

const Article: React.FC<ArticleProps> = (props): JSX.Element => {
  const {article} = props;
  return (
    <Box m={10}>
      {article &&
        <Box>
          <Flex>
            <Image
              src={article.image ? article.image :
`$${process.env.PUBLIC_URL}/images/image-placeholder.png`}
              w={400} borderRadius={10} m={5} alt={'article title pic
```

```

here'}/>

article.title}}/>

<Box>
  <Heading m={5} dangerouslySetInnerHTML={{__html:
  <Flex>
    <ViewIcon m={1}/>
    <Text>{article.viewCount}</Text>
  </Flex>
  <Flex m={5}>
    {
      article.tags && article.tags.map((tag, key) => {
        return (<Text m={1} key={key}>#{tag}</Text>)
      })
    }
  </Flex>
</Box>
</Flex>
<Box>
  <Text dangerouslySetInnerHTML={{__html: article.text}}/>
</Box>
{article.author
  ? <Box>
    <Text>Author: </Text>
    <Link as={ReactLink} to={`/users/${article.authorId}`}>
      <Image w={100} borderRadius={50}
        src={article.author.avatar ? article.author.avatar
: `${process.env.PUBLIC_URL}/images/avatar-placeholder.jpg`}
        alt={'no avatar here'}/>
      <Text>{article.author.nickname}</Text>
    </Link>
  </Box>
  : <Box>
    <Image w={100} borderRadius={50}
      src={`${process.env.PUBLIC_URL}/images/avatar-
placeholder.jpg`}
      alt={'no avatar here'}/>
    <Text>Deleted author</Text>
  </Box>
}
{article.reviewer
  ? <Box>
    <Text>Reviewer: </Text>
    <Link as={ReactLink} to={`/users/${article.authorId}`}>
      <Image
        w={100} borderRadius={50}
        src={article.reviewer.avatar ?
article.reviewer.avatar : `${process.env.PUBLIC_URL}/images/avatar-placeholder.jpg`}
        alt={'no avatar here'}/>
      <Text>{article.reviewer.nickname}</Text>
    </Link>
  </Box>
  : article.reviewerId
    ?
    <Box>
      <Image w={100} borderRadius={50}
        src={`${process.env.PUBLIC_URL}/images/avatar-
placeholder.jpg`}
        alt={'no avatar here'}/>
      <Text>Deleted reviewer</Text>
    </Box>
    : <Box>
      <Text>No reviewer</Text>
    </Box>
  }
</Box>
}
)
}

export default Article;

```

Дана сторінка відображає головне зображення статті, заголовок, текст, автора, рецензента, а також коментарі, і, якщо користувач авторизований, текстове поле для додавання коментаря.

Висновки до розділу 2

Було створено клієнтську і серверну частину системи. Для сервера це мікросервіси article, users, simbad, telegram-bot, gateway, filesю Таким чином, вдалось реалізувати перегляд статей, коментарів, їх написання, зміна написаних статей і власного профілю, а також отримування даних з відкритої бази даних SIMBAD за допомогою Telegram-бота. А для адміністративних ролей були реалізовані перегляд і рецензування не опублікованих статей, видалення статей і їх публікація, взаємодія зі скаргами і коментарями, користувачами, в тому числі зміна їх ролей.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ

3.1. Порядок встановлення та налаштування системи.

Оскільки система є міросервісною, і використовується архітектура MVC, то серверна частина системи може розміщуватись як на одному сервері, так і на декількох, за потребою, і клієнтська частина може бути розміщена окремо. Вимоги можуть змінюватись в залежності від навантаження, але мінімальні вимоги можна розрахувати, однозначно варто зробити певний запас на випадок більшого навантаження ніж було розраховано. Тому, розгорнути серверну частину системи можливо на сервері з такими мінімальними характеристиками:

- Чотириядерний процесор з тактовою частотою 3 ГГц або вище;
- 6 ГБ оперативної пам'яті;
- 5 ГБ вільного місця на жорсткому диску;

Для запуску серверної частини додатку потрібно виконати наступні дії:

1. Встановити Node.js версії 16.18.0.
2. Виконати команду `ssh-keygen`, і додати отриманий публічний `ssh-ключ` на сервер.
3. Виконати команду `yarn`, що встановить залежності системи, які знаходяться в файлі `package.json`.
4. Виконати команду `yarn build` для генерації коду мовою JavaScript.
5. Виконати команду `yarn start`, що запустить систему.

Для розгортання клієнтської частини системи потрібно виконати наступні дії:.

1. Встановити Node.js версії 16.18.0.
2. Виконати команду `yarn`, що встановить всі необхідні залежності додатку, що знаходяться в файлі `package.json`
3. Виконати команду `yarn start`, що запустить додаток

Для взаємодії з системою підійде будь-який сучасний веб-браузер такий як Google Chrome версія 45+, Mozilla Firefox версія 38+ і інші. Для взаємодії з Telegram – треба встановити месенджер Telegram останньої версії.

3.2. Структура інтерфейсу. Інтерфейс та порядок роботи з системою.

При відкритті клієнтської частини додатку користувач побачить список статей. У верхній частині сторінки будуть посилання на вхід та реєстрацію

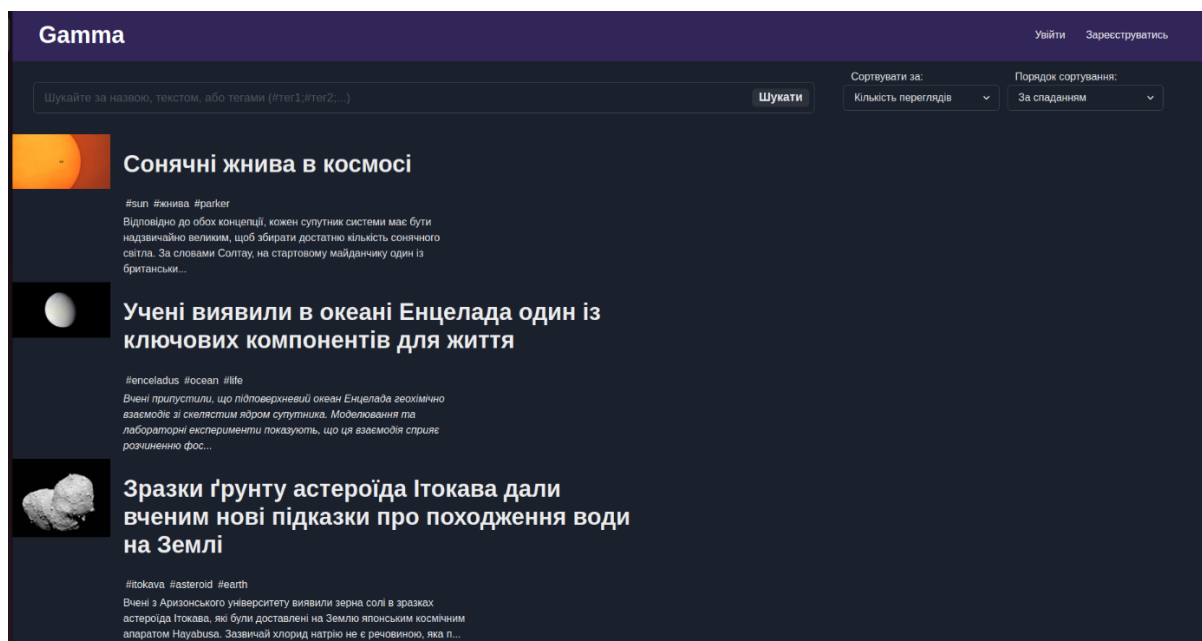


Рис. 3.1. Домашня сторінка

Якщо перейти за посиланням зі списку, можна ознайомитись зі статтею детальніше

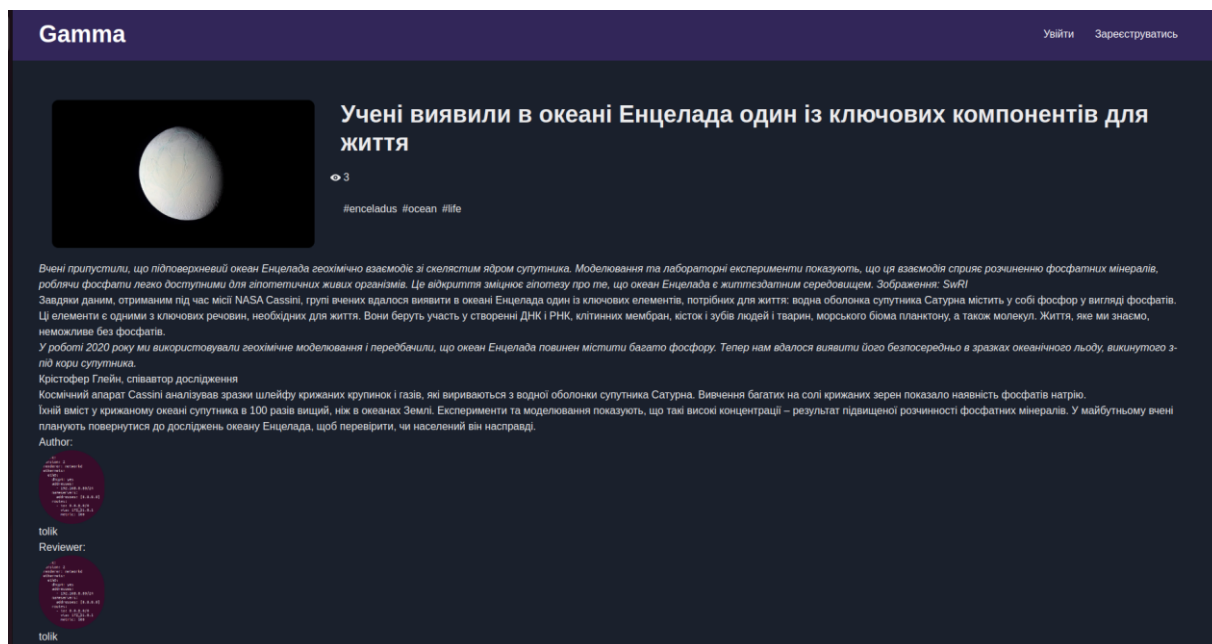


Рис. 3.2. Читання статті

Також, доступна функція пошуку статті, рис.3.3 демонструє пошук за КЛЮЧОВИМ СЛОВОМ

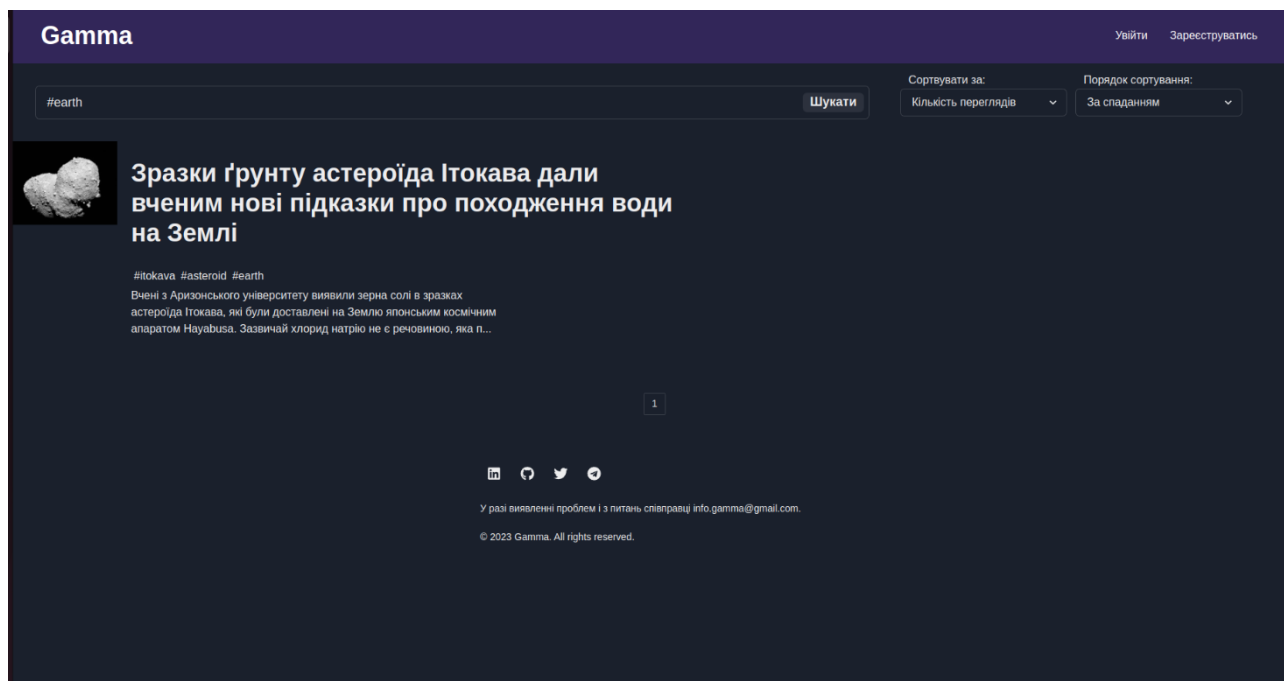
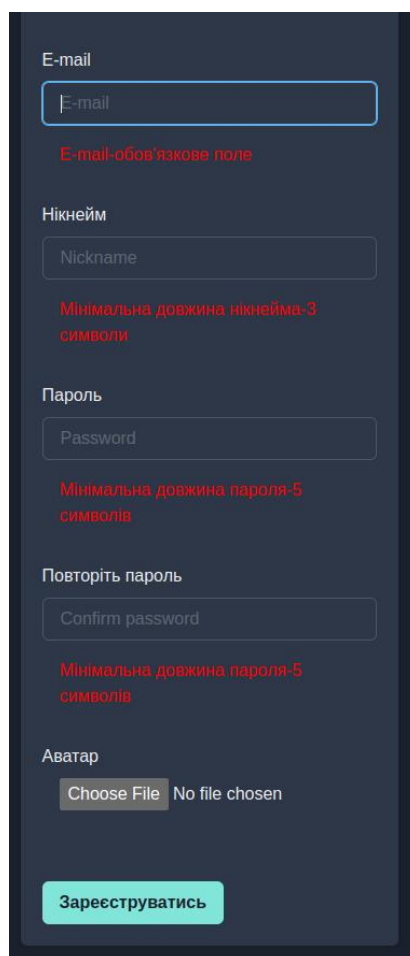


Рис. 3.3. Пошук за ключовими словами

Перейдемо до реєстрації. Для реєстрації треба вказати e-mail, нікнейм, пароль, підтвердження пароля, і, за бажанням, аватар профіля

Рис. 3.4. Форма реєстрації

У разі введення некоректних даних з'являються повідомлення з підказками (рис.3.5)

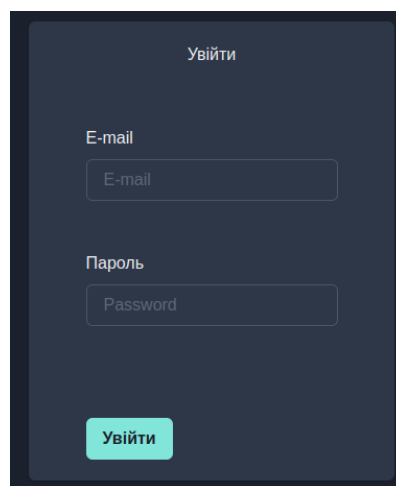


The image shows a registration form on a dark background. It contains the following fields and messages:

- E-mail:** A text input field with the placeholder "E-mail". Below it is a red error message: "E-mail-обов'язкове поле".
- Нікнейм:** A text input field with the placeholder "Nickname". Below it is a red error message: "Мінімальна довжина нікнейма-3 символи".
- Пароль:** A text input field with the placeholder "Password". Below it is a red error message: "Мінімальна довжина пароля-5 символів".
- Повторіть пароль:** A text input field with the placeholder "Confirm password". Below it is a red error message: "Мінімальна довжина пароля-5 символів".
- Аватар:** A file selection area with a "Choose File" button and the text "No file chosen".
- Зареєструватись:** A large teal button at the bottom.

Рис. 3.5. Форма реєстрації з повідомленнями про неправильність введених даних

Форма входу виглядає схожим чином. Для входу достатньо електронної пошти і пароля

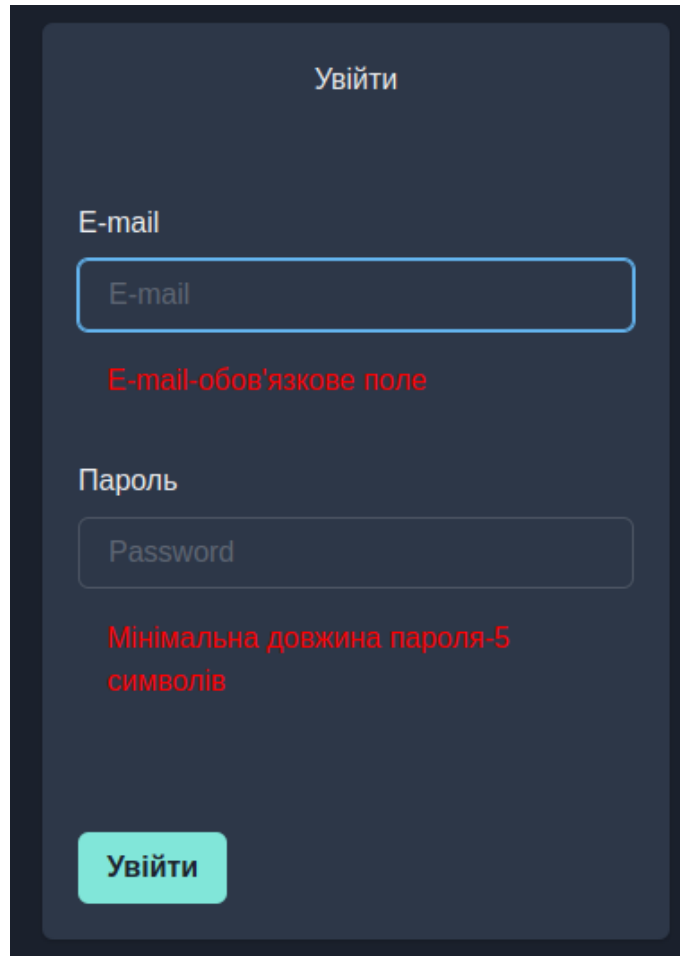


The image shows a login form on a dark background. It contains the following fields and elements:

- Увійти:** The title of the form.
- E-mail:** A text input field with the placeholder "E-mail".
- Пароль:** A text input field with the placeholder "Password".
- Увійти:** A large teal button at the bottom.

Рис. 3.6. Форма для входу у свій профіль

Форма для входу також перевіряє введені дані на коректність



Увійти

E-mail

E-mail

E-mail-обов'язкове поле

Пароль

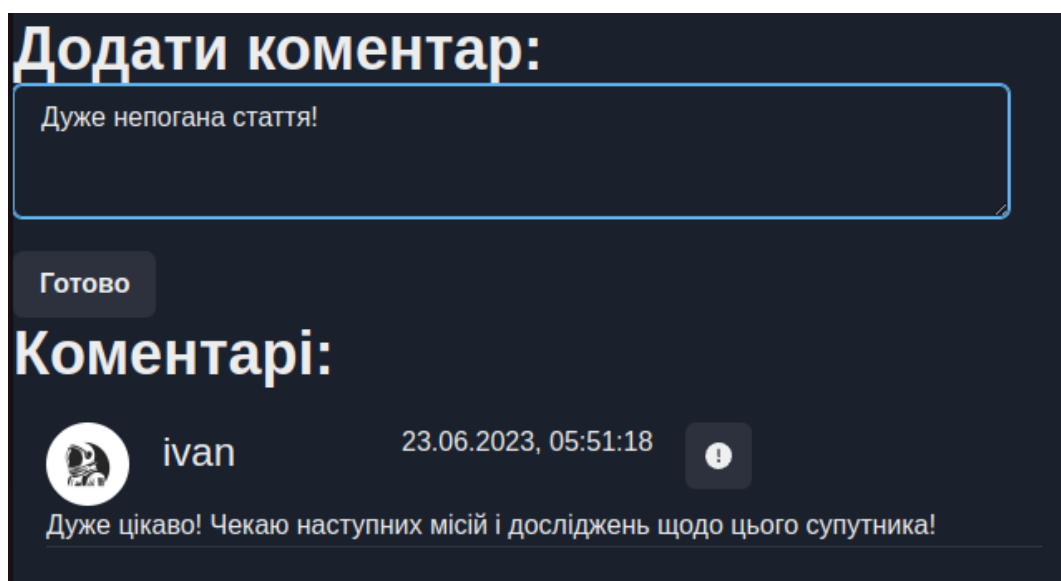
Password

Мінімальна довжина пароля-5 символів

Увійти

Рис. 3.7. Форма входу і повідомлення про некоректні дані

Перегляд коментарів доступний і без авторизації, але для додання коментаря треба обов'язково авторизуватись. Додання коментаря (рис.3.8)



Додати коментар:

Дуже непогана стаття!

Готово

Коментарі:

 ivan 23.06.2023, 05:51:18 

Дуже цікаво! Чекаю наступних місій і досліджень щодо цього супутника!

Рис. 3.8. Додання коментаря, і один вже існуючий коментар

Користувач також може спробувати написати статтю. Якщо її схвалить користувач з роллю «Автор», вона буде опублікована. Сторінка написання статті продемонстрована на рис.3.9

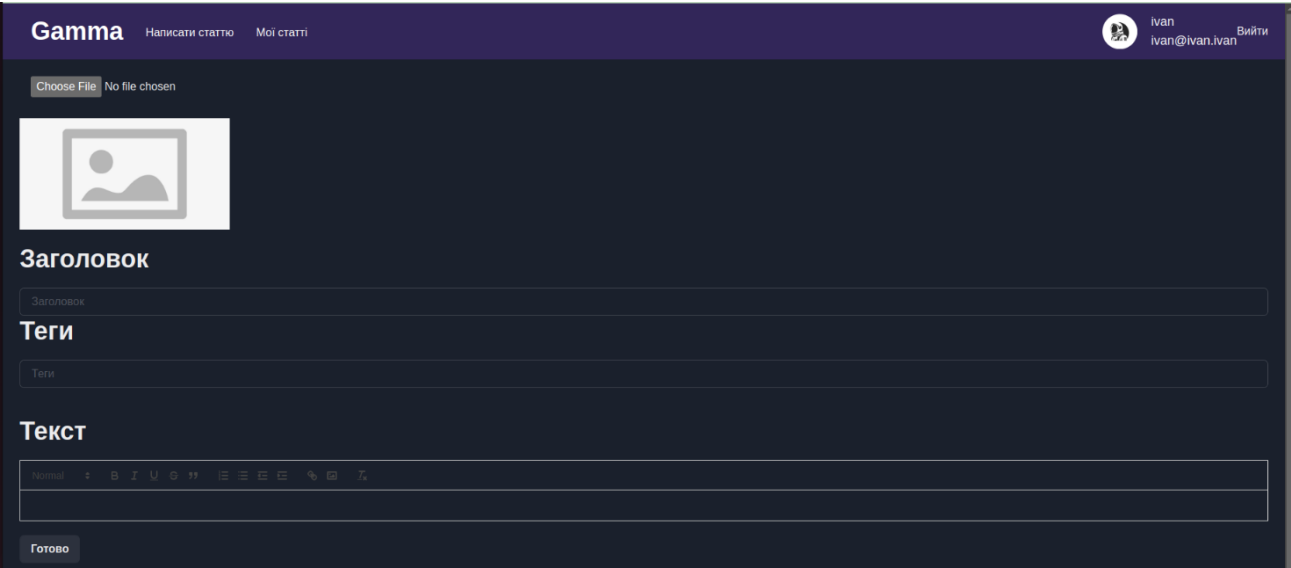


Рис. 3.9. Написання статті

Користувач може переглядати власний профіль і виконувати певні дії, наприклад, змінити аватар, нікнейм, пароль, вийти з акаунта на всіх пристроях, і видалити акаунт

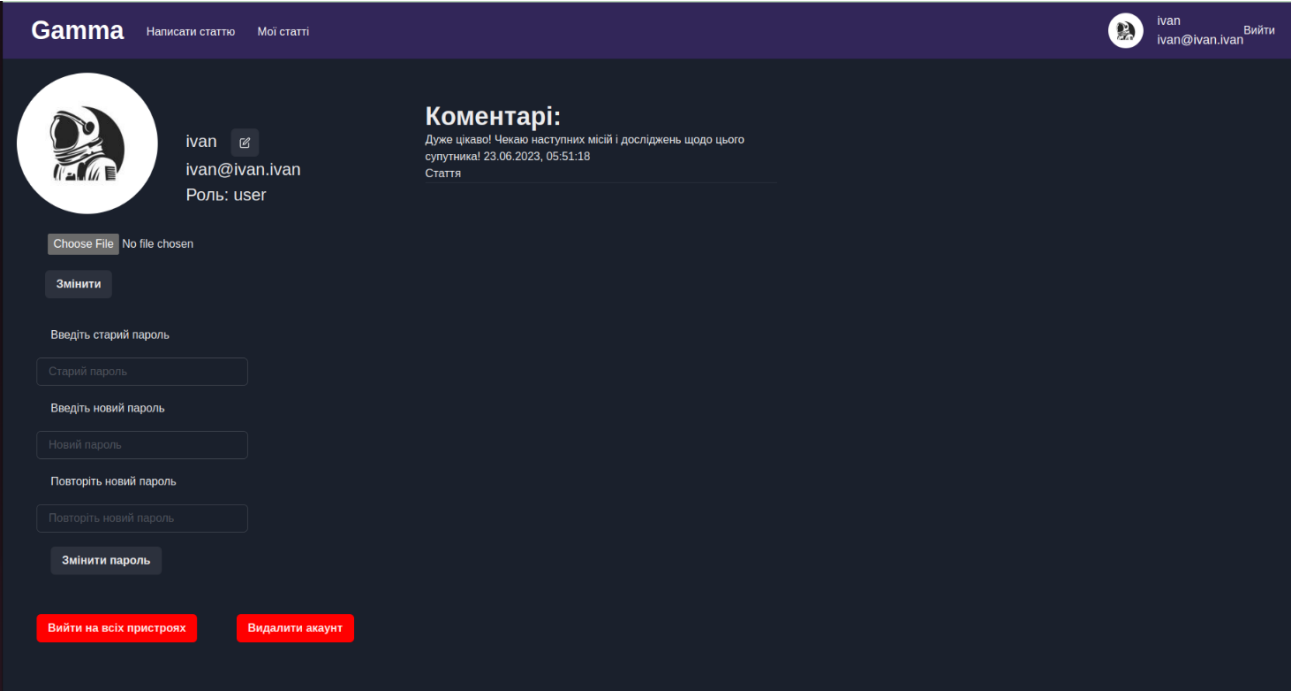


Рис. 3.10. Власний профіль користувача

Для подальшої демонстрації було використано іншого користувача, з роллю «Автор». Він може переглянути запропоновану тільки що статтю

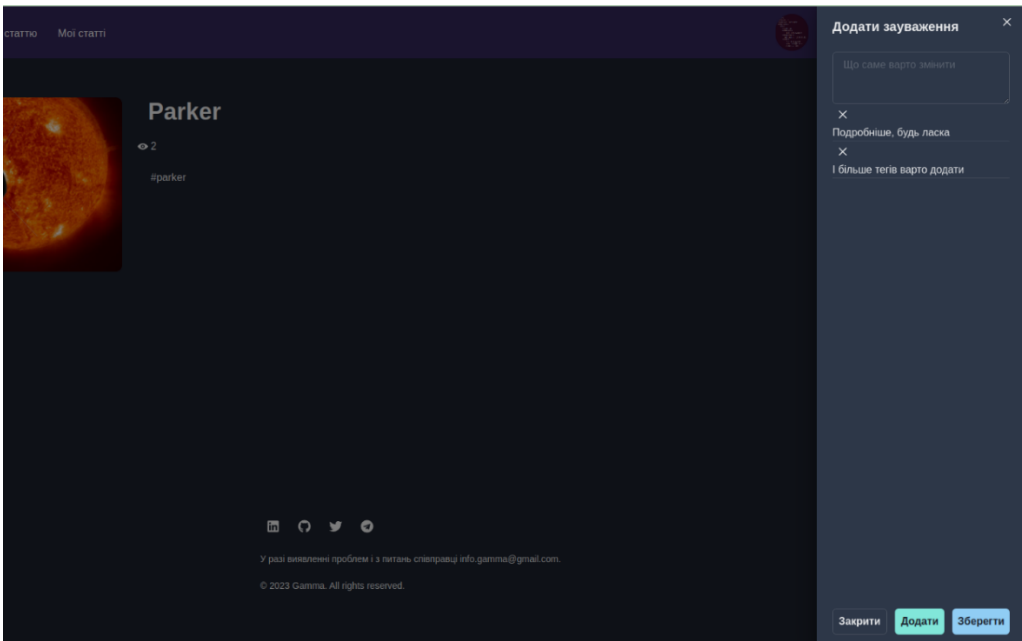


Рис. 3.11. Огляд статті з точки зору автора, і збереження зауважень до неї

Для перегляду зауважень варто знову зайти в профіль звичайного користувача і зайти в розділ «Мої статті». Він містить список, аналогічний до того, що на домашній сторінці. Якщо перейти за посиланням на статтю, побачимо наступне (рис.3.12):

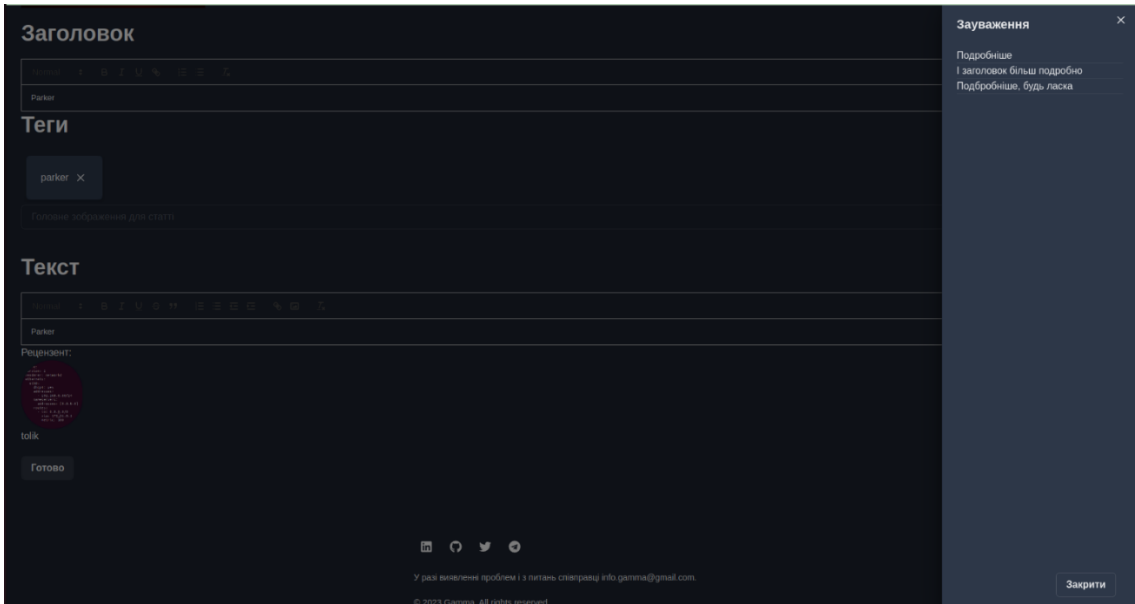


Рис. 3.12. Перегляд зауважень і можливість оновити статтю відповідно до них

Далі було здійснено вихід з профіля користувача і вхід до профіля модератора. Він може переглядати сторінку зі скаргами (рис.3.13), видаляти коментар, якщо він є некоректним, скаргу, якщо вона є недоречною, або користувача, якщо він є серйозним порушником

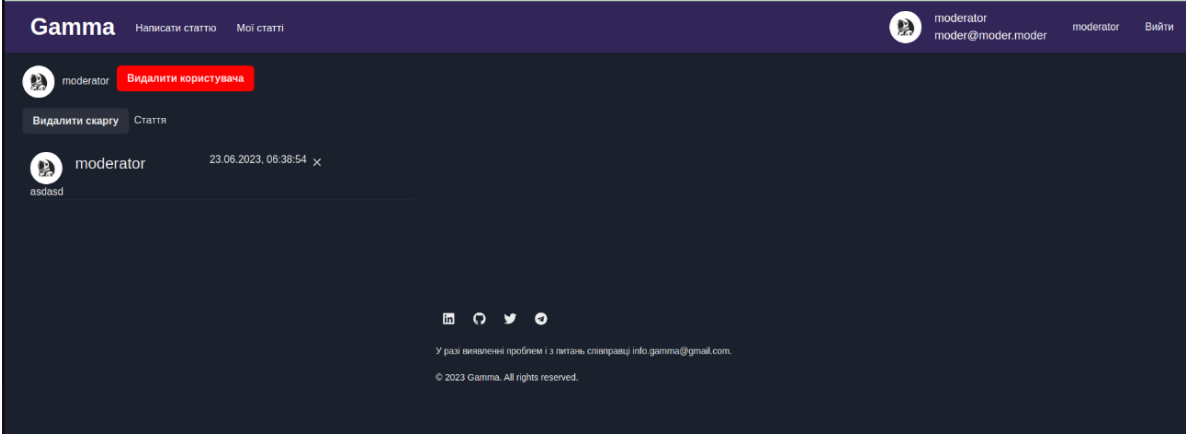


Рис. 3.13. Сторінка для перегляду скарг

Після цього було здійснено вхід в акаунт адміністратора, і перехід на сторінку управління ролями користувачів. Можна здійснити пошук за роллю користувача і нікнеймом

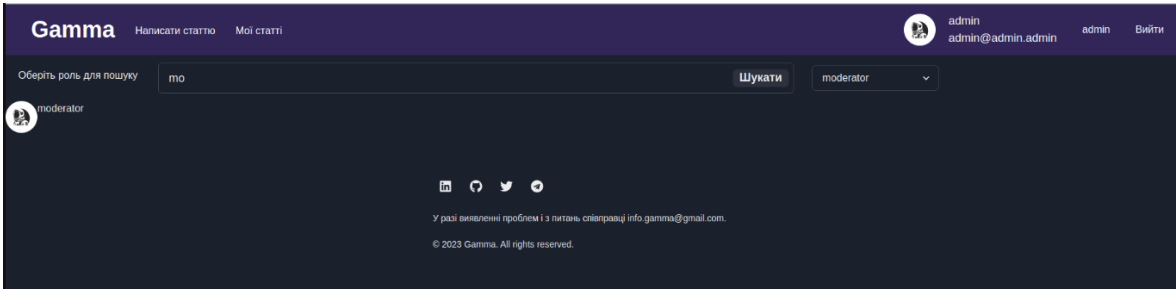


Рис. 3.14. Управління ролями

З цієї сторінки можна перейти на профіль користувача. Тут можливо змінити його роль, якщо є потреба

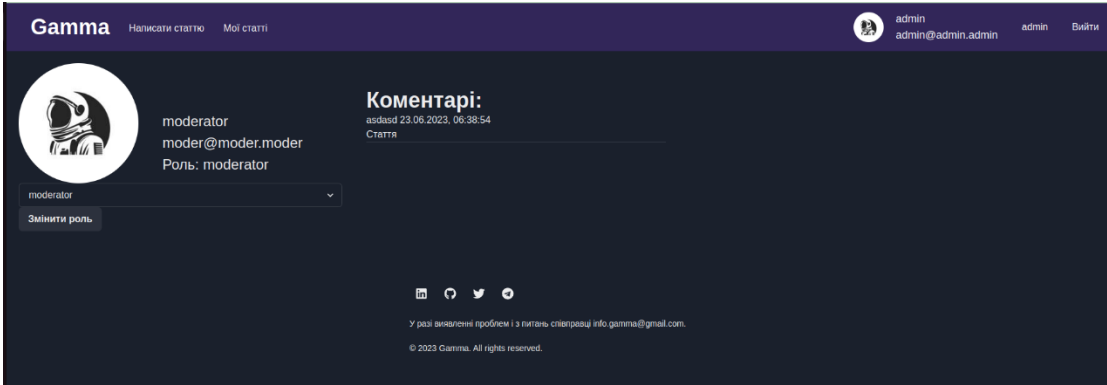


Рис. 3.15. Перегляд профілю користувача з акаунта адміністратора

На цьому має сенс перейти до Telegram-бота і продемонструвати його функціонал

Бот пропонує нам знайти об'єкт в базі даних SIMBAD за ідентифікатором. Варто ввести ідентифікатор, і бот виведе інформацію з бази, перекладену українською і оптимізовану для зручного перегляду. Також, бот шукатиме статті в базі даних, використовуючи ідентифікатор як ключове слово, і якщо такі є- відправить посилання

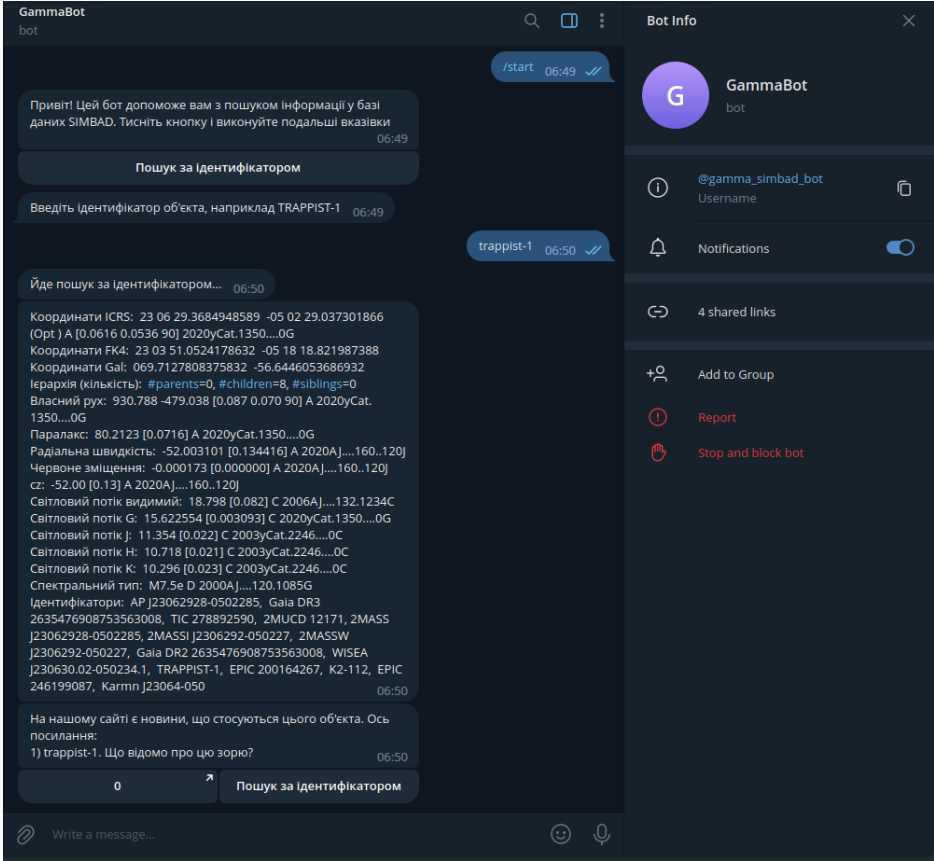


Рис. 3.16. Демонстрація роботи Telegram-бота

3.3.Тестування роботи системи

Після завершення розробки було проведено тестування всіх компонентів системи з метою виявлення проблем у роботі, а також для того щоб переконатись чи відповідає система вимогам, що були поставлені раніше. Для тестування використовувався метод ручного тестування, в результаті якого були виявлені помилки і недоліки, які пізніше було виправлено

Веб-додаток і Telegram-бот тестувались окремо. Telegram-бот не потребує сторонніх інструментів для тестування, цілком достатньо функціоналу

месенджера Telegram. А для тестування веб-сайту варто було обрати зручний інструмент

Було обрано Postman [20]. Це досить відомий і популярний інструмент для розробки серверної частини сайту, який дозволяє імітувати запити будь-якої складності з клієнтської частини. Має всі необхідні поля запиту, такі як параметри, тіло запиту, заголовки, cookies, і так далі. Крім того, Postman має велику кількість налаштувань для спрощення роботи.

На рис. 3.17 наведено приклад надсилання запиту за допомогою програми Postman.

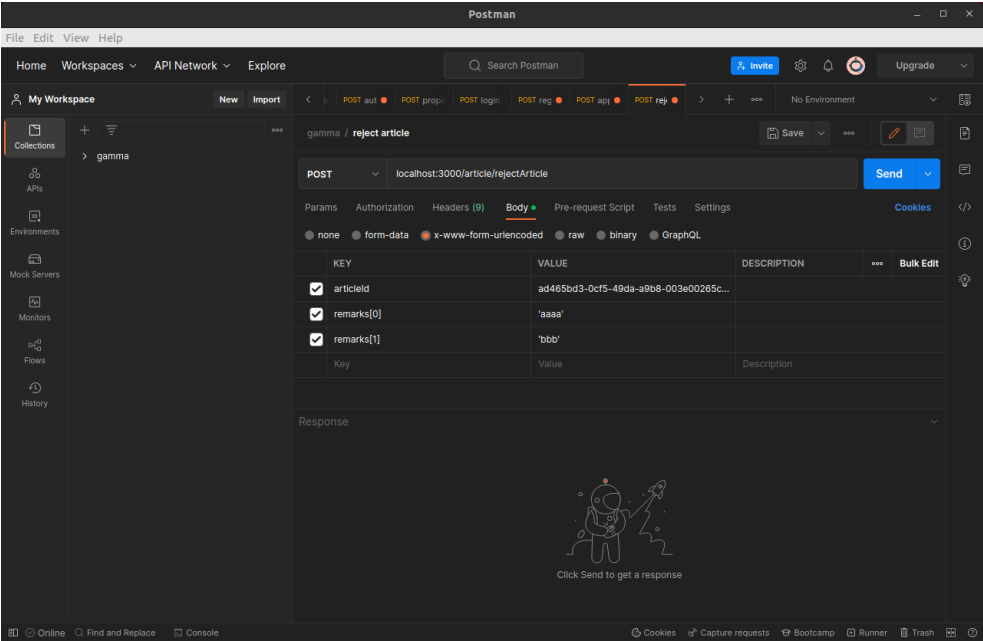


Рис. 3.17. Приклад запиту в програмі Postman

Висновки до розділу 3

В даному розділі було сформовано мінімальні системні вимоги до сервера, на якому буде розгортатись розроблена екосистема, і описано порядок розгортання системи.

Також, було продемонстровано роботу системи у вигляді зображень клієнтської частини системи, і телеграм боту

Крім того, було протестовано роботу системи, і продемонстровано на прикладі принцип тестування HTTP API системи

ВИСНОВКИ

В процесі виконання кваліфікаційної роботи бакалавра було створено інформаційну екосистему у складі веб-додатку і Telegram-бота на космічну тематику “Gamma”. Результатів було досягнуто, оскільки були виконані дані кроки:

1. Виконано аналіз предметної області та існуючих аналогів з метою виявлення потреб користувачів і визначення основних особливостей веб-додатку. Було обрано базу даних PostgreSQL, мову TypeScript і фреймворк Moleculer.js.

2. Встановлено технічне завдання, спроектовано основні принципи роботи, створено базу даних, пакети з основними типами даних і з ядром системи, після чого було створено всі потрібні мікросервіси для серверної частини системи і клієнтську частину веб-додатку

3. Розраховано і наведено мінімальні характеристики сервера для розгортання системи, проведено демонстрацію роботи клієнтської частини системи і Telegram-бота, після чого проведено тестування і усунення помилок.

В результаті виконання роботи було спроектовано і створено інформаційну екосистему на космічну тематику “Gamma”. Були розроблені всі необхідні елементи системи, такі як статті, коментарі, зауваження, скарги, користувачі, і вся взаємодія з ними, яка потрібна для роботи такої системи. Ці елементи повністю відповідають визначеним раніше вимогам. Було забезпечено високу якість системи. Система готова до використання, і лишає багато можливостей для подальшої еволюції і доробки, оскільки проектувалась з розрахунком на подальше розширення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Moleculer.js [Електронний ресурс] – Режим доступу до ресурсу: <https://moleculer.services/docs/0.14/>
2. TypeScript [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org/docs/>
3. Mary Boyd. Single Page Applications: A Powerful Design Pattern for Modern Web Apps – Medium [Електронний ресурс] / Mary Boyd – 9 березня 2018 р. – Режим доступу до ресурсу: <https://medium.com/a-lady-dev/single-page-applications-a-powerful-design-pattern-for-modern-web-apps-ec3590bb7e7a>
4. Node.js [Електронний ресурс] – Режим доступу до ресурсу: <https://nodejs.org/uk/docs>
5. Advantages of PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: <https://www.cybertec-postgresql.com/en/postgresql-overview/advantages-of-postgresql/>
6. Pros and Cons of ReactJS [Електронний ресурс] – Режим доступу до ресурсу: <https://www.javatpoint.com/pros-and-cons-of-react>
7. Sequelize [Електронний ресурс] – Режим доступу до ресурсу: <https://sequelize.org/docs/v6/>
8. Grammy [Електронний ресурс] – Режим доступу до ресурсу: <https://grammy.dev/guide/>
9. What Is A RESTful API? [Електронний ресурс] – Режим доступу до ресурсу: <https://aws.amazon.com/what-is/restful-api/>
10. Redux.js. Usage with TypeScript [Електронний ресурс] – Режим доступу до ресурсу: <https://grammy.dev/guide/>
11. Redux Toolkit [Електронний ресурс] – Режим доступу до ресурсу: <https://redux-toolkit.js.org/introduction/getting-started>
12. React-router-dom [Електронний ресурс] – Режим доступу до ресурсу: <https://reactrouter.com/en/main/start/tutorial>
13. React Hook Form [Електронний ресурс] – Режим доступу до ресурсу: <https://react-hook-form.com/get-started>

14. Chakra UI [Электронный ресурс] – Режим доступа до ресурсу:
<https://chakra-ui.com/getting-started>
15. useEffect [Электронный ресурс] – Режим доступа до ресурсу:
<https://react.dev/reference/react/useEffect>
16. SIMBAD. How to query SIMBAD with URLs [Электронный ресурс] – Режим
доступу до ресурсу: <http://simbad.u-strasbg.fr/Pages/guide/sim-url.htm>
17. Axios [Электронный ресурс] – Режим доступа до ресурсу: [https://axios-](https://axios-http.com/docs/intro)
[http.com/docs/intro](https://axios-http.com/docs/intro)
18. Node.js documentation. Http [Электронный ресурс] – Режим доступа до
ресурсу: <https://nodejs.org/api/http.html>
19. Relational vs. Non-Relational Database: Pros & Cons [Электронный ресурс] –
Режим доступа до ресурсу: [https://aloo.co/blog/relational-vs-non-relational-](https://aloo.co/blog/relational-vs-non-relational-database-pros-cons)
[database-pros-cons](https://aloo.co/blog/relational-vs-non-relational-database-pros-cons)
20. Postman overview [Электронный ресурс] – Режим доступа до ресурсу:
<https://learning.postman.com/docs/introduction/overview/>

ДОДАТКИ

Модель статті

```

import Comment from './comment'
import {Column, PrimaryKey, Table, Model, DataType, Default, AllowNull, Index, HasMany}
from "sequelize-typescript";

@Table({tableName: 'articles'})
class Article extends Model<Article> {
  @PrimaryKey
  @Column({
    type: DataType.UUIDV4,
    defaultValue: DataType.UUIDV4
  })
  id: string;

  @Index('article_search_index')
  @Column(DataType.CHAR(200))
  title: string;

  @Index('article_search_index')
  @Column(DataType.TEXT)
  text: string;

  @Column(DataType.ARRAY(DataType.STRING(25)))
  tags: string[];

  @Column(DataType.UUIDV4)
  authorId: string;

  @Default(false)
  @Column(DataType.BOOLEAN)
  posted: boolean;

  @AllowNull
  @Column(DataType.UUIDV4)
  reviewerId?: string;

  @AllowNull
  @Column(DataType.DATE)
  firstPublishedAt?: Date;

  @Column(DataType.CHAR(400))
  image: string;

  @Column(DataType.NUMBER)
  viewCount: number;

  @Column(DataType.DATE)
  createdAt: Date;

  @AllowNull
  @Column(DataType.DATE)
  updatedAt?: Date;

  @HasMany(() => Comment)
  comments?: Comment[];
}

export default Article

```

Модель коментаря

```
import Article from "../article";
import {
  Column,
  PrimaryKey,
  Model,
  DataType,
  Table,
  ForeignKey,
  AllowNull,
  BelongsTo, HasMany
} from "sequelize-typescript";
import Report from "../report";

@Table({tableName: 'comments'})
class Comment extends Model<Partial<Comment>> {
  @PrimaryKey
  @Column({
    type: DataType.UUIDV4,
    defaultValue: DataType.UUIDV4
  })
  id: string;

  @Column(DataType.UUIDV4)
  @ForeignKey(() => Article)
  articleId: string;

  @BelongsTo(() => Article)
  article: Article;

  @Column(DataType.UUIDV4)
  authorId: string;

  @Column(DataType.CHAR(1000))
  text: string;

  @Column(DataType.DATE)
  createdAt: Date;

  @AllowNull
  @Column(DataType.DATE)
  updatedAt?: Date;

  @HasMany(() => Report)
  reports: Report[];
}

export default Comment;
```

Модель зауваження до статті

```
import {Column, PrimaryKey, Table, Model, DataType, ForeignKey, BelongsTo} from
"sequelize-typescript";
import Article from "../article";

@Table({ tableName: 'remarks' })
class Remark extends Model<Remark> {
  @PrimaryKey
  @Column({
    type: DataType.UUIDV4,
    defaultValue: DataType.UUIDV4
  })
  id: string;

  @Column(DataType.UUIDV4)
  authorId: string;

  @Column(DataType.UUIDV4)
```

```

@ForeignKey(() => Article)
articleId: string;

@BelongsTo(() => Article)
article: Article;

@Column(DataType.CHAR(500))
text: string;
}

export default Remark

```

Модель скарги на коментар

```

import Comment from "../comment";
import {
  Column,
  PrimaryKey,
  Model,
  DataType,
  Table,
  ForeignKey,
  AllowNull,
  BelongsTo
} from "sequelize-typescript";

@Table({tableName: 'reports'})
class Report extends Model<Report> {
  @PrimaryKey
  @Column({
    type: DataType.UUIDV4,
    defaultValue: DataType.UUIDV4
  })
  id: string;

  @Column(DataType.UUIDV4)
  @ForeignKey(() => Comment)
  commentId: string;

  @BelongsTo(() => Comment)
  comment: Comment

  @Column(DataType.UUIDV4)
  userId: string;

  @Column(DataType.DATE)
  createdAt: Date;

  @AllowNull
  @Column(DataType.DATE)
  updatedAt?: Date;
}

export default Report;

```

Модель користувача

```

import {Role} from "package-types";
import {Model, Table, Column, DataType, AllowNull, PrimaryKey, Default, Unique} from
"sequelize-typescript";

@Table({tableName: 'users'})
export default class User extends Model<User>{
  @PrimaryKey
  @Column({
    type: DataType.UUIDV4,
    defaultValue: DataType.UUIDV4

```

```

    })
    id: string

    @Default(Role.USER)
    @Column(DataType.CHAR(10))
    role: string

    @Unique
    @Column(DataType.CHAR(100))
    email: string

    @Unique
    @Column(DataType.CHAR(50))
    nickname: string

    @Column(DataType.CHAR(300))
    password: string

    @AllowNull
    @Column(DataType.CHAR(400))
    avatar: string

    @Column(DataType.DATE)
    createdAt: Date

    @AllowNull
    @Column(DataType.DATE)
    updatedAt: Date
}

```

Модель токена

```

import {Role} from "package-types";
import {Model, Table, Column, DataType, AllowNull, PrimaryKey, Default, Unique} from
"sequelize-typescript";

@Table({tableName: 'tokens'})
export default class Token extends Model<Token>{
  @PrimaryKey
  @Column({
    type: DataType.UUIDV4,
    defaultValue: DataType.UUIDV4
  })
  id: string;

  @Column(DataType.CHAR(600))
  token: string;

  @Column(DataType.CHAR(20))
  type: string;

  @Column(DataType.UUIDV4)
  userId: string;

  @Column(DataType.DATE)
  createdAt: Date;

  @Column(DataType.DATE)
  updatedAt: Date;
}

```

Допоміжний метод що обробляє картинки в тексті статті

```

import {FilesActionName, ServiceName, UploadBuffersPayload, UploadBuffersResult} from
"package-types";
import {App} from "package-app";

```

					ІПЗ.КР.Б – 121 – 23 – ПЗ	69

```

const allowedMimetypes = ['image/png', 'image/jpg', 'image/jpeg'];

export async function setImagesToArticleText(text: string): Promise<string> {
  const textParts = text.split('" alt="asdas"/`;
        replacedText += croppedPart;
        replacedText += text;
      }
    } else {
      replacedText += part;
    }
  });
  if(Object.keys(result.buffers).length === 0) {
    return replacedText;
  }

  const {fileKeys} = await App.call<UploadBuffersPayload,
UploadBuffersResult>(ServiceName.Files, FilesActionName.UploadBuffers, result);
  let resultText = '';
  Object.keys(fileKeys).forEach((key) => {
    const searchValue = `<${key.split('.')[0]}>`
    resultText += replacedText.replace(`${searchValue}`, fileKeys[key]);
  });

  return resultText;
}

```

ДОДАТОК Б

Повний код сторінки зміни статті

```

import React, {useEffect, useState} from "react";
import {
  Box,
  Heading,
  Image,
  Input,
  Button,
  Card,
  CardBody,
  Text,
  CloseButton,
  Flex,
  DrawerOverlay,
  DrawerContent,
  DrawerCloseButton,
  DrawerHeader,
  DrawerBody,
  Divider,
  DrawerFooter,
  Drawer,
  useDisclosure, Link
} from '@chakra-ui/react';
import request from "../helpers/request";
import {

```

```

    ApproveArticlePayload, ArticleWithUsers,
    EditArticlePayload, EditArticleResult,
    GetArticleResult, GetRemarksResult, RemarkWithUser,
    Role,
  } from "package-types";
import {AxiosResponse} from 'axios';
import {useAppSelector} from "../hooks";
import {Link as ReactLink, Navigate, useNavigate, useParams} from "react-router-dom";
import ReactQuill from 'react-quill';
import 'react-quill/dist/quill.snow.css';
import sanitize from "sanitize-html";

import 'quill-image-uploader/dist/quill.imageUploader.min.css';

const EditArticle: React.FC = (): JSX.Element => {
  const {articleId} = useParams();
  const [title, setTitle] = useState('');
  const [text, setText] = useState('');
  const [tag, setTag] = useState('');
  const tagsInitialState: string[] = [];
  const [tags, setTags] = useState(tagsInitialState);
  const initialRemarksState: RemarkWithUser[] = [];
  const [remarks, setRemarks] = useState(initialRemarksState);
  const [imageURL, setImageURL] = useState(`${process.env.PUBLIC_URL}/images/image-
placeholder.png`);
  const [image, setImage] = useState<File | undefined>(undefined);
  const {isOpen, onOpen, onClose} = useDisclosure();
  const [article, setArticle] = useState<ArticleWithUsers | null>(null);
  const currentUser = useAppSelector(state => state.user.value);
  const navigate = useNavigate();

  const fetchData = async () => {
    const {data} = await request.get<undefined,
    AxiosResponse<GetArticleResult>>(`/article/getArticle/${articleId}`);
    const {article: fetchedArticle} = data;
    setArticle(fetchedArticle);
    if(!fetchedArticle) {
      return (<Navigate to='/users/login' replace/>);
    }
    if (fetchedArticle) {
      setImageURL(fetchedArticle.image);
      setTitle(fetchedArticle.title);
      setText(fetchedArticle.text);
      setTags(fetchedArticle.tags ?? []);
    }
    const {data: remarksData} = await request.get<undefined,
    AxiosResponse<GetRemarksResult>>(`/article/getRemarks/${articleId}`);
    setRemarks(remarksData.remarks);
  }

  useEffect(() => {
    fetchData();
  }, []);

  if (!currentUser) {
    return (<Navigate to='/users/login' replace/>);
  }

  const handleClick = (e: React.MouseEvent) => {
    if (e.detail === 2) {
      onOpen();
    }
  }

  const onClick = async () => {
    const sanitizedTitle = sanitize(title);
    const sanitizedText = sanitize(text, {
      allowedTags: sanitize.defaults.allowedTags.concat([ 'img' ]),
      allowedAttributes: { 'img': [ 'src' ] },
      allowedSchemes: [ 'data', 'http', 'https' ]
    });
  }

```

```

let chosenImage;
if(image) {
  chosenImage = image;
} else {
  chosenImage = imageURL;
}
try {
  if (sanitizedText && sanitizedTitle) {
    const {data} = await request.put<EditArticlePayload,
AxiosResponse<EditArticleResult>>(`/article/editArticle/${articleId}`, {
      title: sanitizedTitle,
      text: sanitizedText,
      authorId: currentUser.id,
      tags,
      image: chosenImage
    });
    if (data.success) {
      return navigate('/articles/myArticles');
    } else {
      throw new Error('Не вдалось відредагувати статтю');
    }
  }
} catch (err) {
  console.log(err);
}

const onImageChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  const images = e.target.files;
  if (images) {
    const rawImage = images[0];
    const url = URL.createObjectURL(rawImage);
    setImage(rawImage);
    setImageURL(url);
  } else {
    setImageURL(`${process.env.PUBLIC_URL}/images/image-placeholder.png`);
  }
}

const onTagsChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  const {value} = e.target;
  setTag(value);
}

const onTagInputKeyPress = (e: React.KeyboardEvent<HTMLInputElement>) => {
  if (e.key === 'Enter') {
    if (tag) {
      const sanitizedTag = sanitize(tag, {
        allowedTags: [],
        allowedAttributes: {},
        allowedIframeHostnames: []
      });
      if (sanitizedTag) {
        setTags([...tags, sanitizedTag]);
        setTag('');
      }
    }
  }
}

const onRemoveButtonClick = (text: string) => () => {
  const currentTags = JSON.parse(JSON.stringify(tags));
  currentTags.splice(tags.indexOf(text), 1);
  setTags(currentTags);
}

const handlePublish = async () => {
  if (!article) {
    return;
  }
  try {

```

```

        await request.post<ApproveArticlePayload,
AxiosResponse<ApproveArticlePayload>>('article/approveArticle', {
            articleId: article.id
        });
    } catch (err) {
        console.log(err);
    }
    return navigate('/admin');
}

const quillModules = {
    toolbar: [
        [{ 'header': [1, 2, false] }],
        ['bold', 'italic', 'underline', 'strike', 'blockquote'],
        [{ 'list': 'ordered' }, { 'list': 'bullet' }, { 'indent': '-1' }, { 'indent':
'+1' }],
        ['link', 'image'],
        ['clean']
    ],
}

return (
    <Box m={6}>
        <Drawer
            isOpen={isOpen}
            placement='right'
            onClose={onClose}
        >
            <DrawerOverlay/>
            <DrawerContent>
                <DrawerCloseButton/>
                <DrawerHeader>Зауваження</DrawerHeader>

                <DrawerBody>
                    <Box>
                        {
                            remarks.map(remark => {
                                return (
                                    <Box key={remark.id}>
                                        <Text>{remark.text}</Text>
                                        <Divider/>
                                    </Box>
                                )
                            })
                        }
                        {
                            remarks.map((remark, index) => {
                                return (
                                    <Box key={index}>
                                        {remark.author
                                            ? <Link as={ReactLink}
to={`/users/${remark.authorId}`}>
                                                <Image
                                                    w={50} borderRadius={25}
                                                    src={remark.author.avatar ?
remark.author.avatar : `${process.env.PUBLIC_URL}/images/avatar-placeholder.jpg`}
                                                    alt={'no avatar here'}/>
                                                <Text>{remark.author.nickname}</Text>
                                            </Link>
                                            : <Box>
                                                <Image
                                                    w={50} borderRadius={25}
                                                    src={` ${process.env.PUBLIC_URL}/images/avatar-placeholder.jpg`}
                                                    alt={'no avatar here'}/>
                                                <Text>Deleted reviewer</Text>
                                            </Box>
                                        }
                                        <Text>{remark.text}</Text>
                                        <Divider/>
                                    </Box>
                                )
                            })
                        }
                    </Box>
                </DrawerBody>
            </DrawerContent>
        </Drawer>
    </Box>
)

```

```

    }
  </Box>
</DrawerBody>

<DrawerFooter>
  <Button variant='outline' mr={3} onClick={onClose}>
    Закрити
  </Button>
</DrawerFooter>
</DrawerContent>
</Drawer>
<Box onClick={handleClick}>
  <Input mb={5} border={'none'} type='file' placeholder='Головне
зображення для статті'
    onChange={onImageChange}/>
  <Image mb={5} src={imageUrl} alt={'no image here'} w={300}/>
  <Heading mb={5}>Заголовок</Heading>
  <ReactQuill theme="snow" value={title} onChange={setTitle}/>
  <Heading mb={5}>Теги</Heading>
  <Flex wrap={"wrap"}>
    {tags.map((tag, index) => {
      return (
        <Card maxW={'max-content'} m={2} key={index}>
          <CardBody>
            <Flex>
              <Text m={1}>{tag}</Text>
              <CloseButton onClick={onRemoveButtonClick(tag)}/>
            </Flex>
          </CardBody>
        </Card>
      )
    })}
  </Flex>
  <Input mb={5} type='text' value={tag} placeholder='Головне зображення
для статті'
    onChange={onTagsChange} onKeyDown={onTagInputKeyPress}/>
  <Heading mt={5} mb={5}>Текст</Heading>
  <ReactQuill theme="snow" value={text} onChange={setText}
modules={quillModules}/>
  <Button mt={5} onClick={onClick}>
    Готово
  </Button>
  { (currentUser && currentUser.role === Role.AUTHOR) &&
  <Button mt={5} ml={5} onClick={handlePublish}>
    Опублікувати
  </Button>
  }
</Box>
</Box>
)
}

export default EditArticle;

```

Домашня сторінка

```

import React, {useEffect, useState} from "react";
import {Box} from '@chakra-ui/react';
import request from "../helpers/request";
import {ArticleWithUsers, SearchArticlesResult} from "package-types";
import {AxiosResponse} from 'axios';
import ArticlesList from "../components/articles/articlesList";
import Paginator from "../components/paginator";

const Home: React.FC = (): JSX.Element => {
  const initialState: ArticleWithUsers[] = [];
  const [articles, setArticles] = useState(initialState);

```

```

const fetchData = async () => {
  const {data} = await request.get<undefined,
AxiosResponse<SearchArticlesResult>>('/article/searchArticles');
  const {articles: fetchedArticles} = data;
  setArticles(fetchedArticles);
}
useEffect(() => {
  fetchData();
}, []);
const onPageClick = (page: number) => async () => {
  const {data} = await request.get<undefined,
AxiosResponse<SearchArticlesResult>>(`/article/searchArticles?page=${page}`);
  const {articles} = data;
  setArticles(articles);
}
return (
  <Box>
    <ArticlesList articles={articles} review={false} />
    <Paginator items={articles} onPageClick={onPageClick} />
  </Box>
)
}
export default Home;

```

Форма реєстрації

```

import React, {useState} from "react";
import {
  Box,
  Center,
  Button,
  Text,
  Card,
  CardBody,
  CardHeader,
  Input,
  CardFooter,
  FormLabel
} from '@chakra-ui/react';
import {useForm} from "react-hook-form";
import {yupResolver} from '@hookform/resolvers/yup';
import * as yup from 'yup';
import axios, {AxiosResponse} from "axios";
import config from "../env/env";
import {RegisterPayload, RegisterResult} from 'package-types';
import {useNavigate} from 'react-router-dom';

const schema = yup.object({
  email: yup.string().email().required(),
  nickname: yup.string().min(3, 'Мінімальна довжина нікнейма-3 символи').max(50,
'Mаксимальна довжина нікнейма-50 символів').required('Нікнейм-обов'язкове поле'),
  password: yup.string().min(5, 'Мінімальна довжина пароля-5 символів').max(30,
'Mаксимальна довжина пароля-30 символів').required('Пароль-обов'язкове поле'),
  passwordConfirmation: yup.string().oneOf([yup.ref('password'), undefined], 'Паролі
мають співпадати').required(),
  avatar: yup.mixed(),
}).required();

type FormData = yup.InferType<typeof schema>;

const Register: React.FC = () => {
  const [error, setError] = useState('');
  const {register, handleSubmit, formState: {errors}} = useForm<FormData>({
    resolver: yupResolver(schema)
  });
  const navigate = useNavigate();
  const fetchData = async (registerData: RegisterPayload) => {
    const {data} = await axios.post<RegisterPayload,

```

```

AxiosResponse<RegisterResult>>(`${config.beUrl}/users/register`, {...registerData}, {
  headers: {
    'Content-Type': 'multipart/form-data'
  }
});
if (!data.success) {
  setError(data.message ?? 'Помилка реєстрації');
  return;
}
return navigate("/users/login");
}

const onSubmit = (data: FormData) => {
  const { email, password, nickname, avatar } = data;
  fetchData({email, password, nickname, avatar});
}

return (
  <Center>
    <Card m={5} w={350} align='center'>
      <CardHeader>
        Log in
      </CardHeader>
      <form onSubmit={handleSubmit(onSubmit)} encType='multipart/form-data'>
        <CardBody>
          <Box mt={4}>
            <FormLabel>E-mail</FormLabel>
            <Input type="text" placeholder="E-mail"
{...register('email')}/>
            <Text p={4} color='red'>{errors.email?.message}</Text>
          </Box>
          <Box mt={4}>
            <FormLabel>Nickname</FormLabel>
            <Input type="text" placeholder="Nickname"
{...register('nickname')}/>
            <Text p={4} color='red'>{errors.nickname?.message}</Text>
          </Box>
          <Box mt={4}>
            <FormLabel>Password</FormLabel>
            <Input type="password" placeholder="Password"
{...register('password')}/>
            <Text p={4} color='red'>{errors.password?.message}</Text>
          </Box>
          <Box mt={4}>
            <FormLabel>Confirm password</FormLabel>
            <Input type="password" placeholder="Confirm password"
{...register('passwordConfirmation')}/>
            <Text p={4} color='red'>{errors.password?.message}</Text>
          </Box>
          <Box mt={4}>
            <FormLabel>Avatar</FormLabel>
            <Input type="file" placeholder="Select avatar"
{...register('avatar')} border={'none'}/>
          </Box>
        </CardBody>
        <CardFooter>
          <Button type="submit" colorScheme="teal" mt={4}>
            Register
          </Button>
          <Text p={4} color='red'>{error}</Text>
        </CardFooter>
      </form>
    </Card>
  </Center>
)
}

export default Register;

```