НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050103 «Програмна інженерія»

на тему: «Новинний портал із системою управління публікацією контента на основі моделі SaaS»

Виконав:
студент IV курсу, групи КП-51
Мазун Антон Ігорович _____

Керівник:
Старший викладач кафедри ПЗКС,
Гадиняк Р.А. _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н.,
Онай М.В. _____

Рецензент:
доц. кафедри ММСА інституту прикладного
системного аналізу, доц., к.т.н. Дідковська М.В. _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки – 6.050103 «Програмна інженерія»

ЗАТВЕРДЖУЮ
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2018 р.

ЗАВДАННЯ
на дипломний проект студенту

Мазуну Антону Ігоровичу

1. Тема проекту «Новинний портал із системою управління публікацією контента на основі моделі SaaS», керівник проекту Гадиняк Руслан Анатолійович, старший викладач кафедри ПЗКС, затверджені наказом по університету від «22» травня 2019 р. №1331-С

2. Термін подання студентом проекту «16» червня 2019 р.

3. Вихідні дані до проекту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - огляд існуючих програмних рішень;
   - обґрунтування вибору засобів реалізації;
   - структурно-алгоритмічна організація;
   - опис реалізації програмних засобів.

5. Перелік обов'язкового графічного матеріалу:
   - структура бази даних (креслення);
   - життєвий цикл публікації (креслення);
   - діаграма зв'язків компонентів системи (плакат);
   - діаграма прецендентів. Головний редактор (плакат).

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2018 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	Вивчення літератури за тематикою проекту	21.10.2018	
2.	Розроблення та узгодження технічного завдання	19.11.2018	
3.	Розроблення структури web-додатку	30.12.2018	
4.	Підготовка матеріалів першого розділу дипломного проекту	20.01.2019	
5.	Розроблення дизайну сторінок та графічних елементів	15.02.2019	
6.	Підготовка матеріалів другого розділу дипломного проекту	27.02.2019	
7.	Програмна реалізація web-додатку	19.03.2019	
8.	Тестування web-ресурсу	03.04.2019	
9.	Підготовка матеріалів третього розділу дипломного проекту	15.04.2019	
10.	Підготовка матеріалів четвертого розділу дипломного проекту	21.04.2019	
11.	Підготовка графічної частини дипломного проекту	26.04.2019	
12.	Оформлення документації дипломного проекту	25.05.2019	

Студент А.І. Мазун

Керівник проекту Р.А. Гадиняк

АНОТАЦІЯ

Даний дипломний проект присвячено розробленню веб-додатку для автоматизації роботи редакторського та журналістського складів новинного порталу. Створений веб-застосунок – це рішення у вигляді ресурсу, в якому реалізовано особисті кабінети користувачів з різними правами доступу до керування процесом Інтернет-видавництва.

У даній роботі проведено аналіз доступних програмних рішень, який показав, що більшість новинних порталів розроблені за допомогою CMS, оскільки останні мають готові рішення та плагіни для керування контентом сайту. Але з іншого боку, використання CMS не є зручним для даного виду продукту, адже створення та реалізація особистого кабінету користувача є не типовою задачею для новинних порталів, що змушує розробників змінювати ядро системи, а це призводить до проблем з підтримуваністю та розширюваністю продукту в майбутньому. Крім того, розглянуто існуючі технології розроблення веб-додатків та обгрунтовано обрано найбільш ефективні з них для використання при розробленні веб-ресурсу. Також проведено збір та аналіз вимог до програмного застосунку. Розроблено структуру веб-додатку та архітектуру БД. Основними складовими ПЗ є компоненти статті, та відповідні компоненти під кожен тип користувача: блогер, журналіст, редактор, головний редактор. Також працівникам порталу доступний перегляд статистики відвідувань сторінки кожної новини та активності користувачів.

Веб-додаток для керування контентом новинного порталу дозволяє ефективно взаємодіяти між собою працівникам видавництва та робить їх роботу зручнішою.

ABSTRACT

This diploma project is devoted to the development of a web-based application for the editor's and journalist's compilations of the news portal. This web application is a solution in the form of a resource, which implements personal offices of users with different rights to access the management of Internet publishing.

In this paper an analysis of available software solutions was conducted, which showed that most news portals were developed using CMS, arguing that CMS has ready-made solutions and plugins to manage the content of the site. The use of CMS is not convenient for this type of product, since the creation and implementation of a personal cabinet is not a typical task for news portals, which forces developers to change the core of the system, which leads to the difficulty of maintaining and severely expanding the product in the future. In addition, existing web development technologies are discussed and the most effective ones are rightly chosen for use when developing a web resource. Also collected and analyzed requirements for the software application. The structure of the web application and the architecture of the database are developed. The main components of the software are the components of the article, personal management, as well as for each type of user components to work with bloggers, journalists, editors, editor in chief. It is also possible for employees to view the statistics of visits to the news page and user activity.

The web-based application for managing the content of a news portal allows you to interact with publishing staff and make them more user-friendly with role-sharing in the system.

ДП.045440-01-90 Новинний портал із системою управління публікацією контента на основі моделі SaaS

Позначення	Найменування	Кіл-ть	Примітка
	Документація проекту		
ДП.045440-02-91	Новинний портал із	5	
	системою управління		
	публікацією контента на		
	основі моделі SaaS.		
	Технічне завдання		
ДП.045440-03-81	Новинний портал із	78	
	системою управління		
	публікацією контента на		
	основі моделі SaaS.		
	Пояснювальна записка		
ДП.045440-04-51	Новинний портал із	4	
	системою управління		
	публікацією контента на		
	основі моделі SaaS.		
	Програма та методика		
	тестування		
ДП.045440-05-34	Новинний портал із	11	
	системою управління		
	публікацією контента на		
	основі моделі SaaS.		
	Керівництво користувача		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-06-99	Новинний портал із	1	
	системою управління		
	публікацією контента на		
	основі моделі SaaS.		
	Структура бази даних.		
	ERD діаграма		
ДП.045440-07-99	Новинний портал із	1	
	системою управління		
	публікацією контента на		
	основі моделі SaaS.		
	Функціональні		
	Життєвий цикл		
	публікації.		
	UML-діаграма		
ДП.045440-08-98	Новинний портал із	1	
	системою управління		
	публікацією контента на		
	основі моделі SaaS.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2018 р.

НОВИННИЙ ПОРТАЛ ІЗ СИСТЕМОЮ УПРАВЛІННЯ ПУБЛІКАЦІЄЮ КОНТЕНТА НА ОСНОВІ МОДЕЛІ SAAS

Технічне завдання

ДП.045440-02-91

"ПОГОДЖЕНО"

Керівник проекту:

_____ Р.А. Гадиняк

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ А.І. Мазун

2018

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Новинний портал із системою управління публікацією контента на основі моделі SaaS.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проектування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання новинними порталами, у якості адмін-панелі для управління публікаціями та менеджменту редакторським складом.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Web-ресурс повинен забезпечувати такі основні функції:

1) можливість керування «життєвим циклом» публікації;
2) наявність особистого кабінету для редакторського та журналістських складів;
3) можливість керування ролями та даними персоналу;
4) збереження історії змін публікації;

Розробку виконати на платформі Python Django.

Додаткові вимоги:

1) наявність чотирьох типів користувачів: журналіст, редактор, головний редактор та блогер;
2) наявність інтуїтивно зрозумілого інтерфейсу;
3) попередній перегляд публікації;
4) відображення сторінки статистики публікації.

5. ВИМОГИ ДО ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
   - «Структура бази даних. EDR-діаграма»;
   - «Життєвий цикл публікації. UML-діаграма».

6. ЕТАПИ ПРОЕКТУВАННЯ

Вивчення літератури за тематикою роботи……..21.10.2018

Розроблення та узгодження технічного завдання……....19.11.2018

Розроблення структури web-ресурсу…….....30.12.2018

Розроблення дизайну сторінок та графічних елементів……..15.02.2019

Програмна реалізація web-ресурсу……....19.03.2019

Тестування web-ресурсу…….....03.04.2019

Підготовка матеріалів текстової частини проекту……..….21.04.2019

Підготовка матеріалів графічної частини проекту……...…26.04.2019

Оформлення технічної документації проекту……..…25.05.2019

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2019 р.

НОВИННИЙ ПОРТАЛ ІЗ СИСТЕМОЮ УПРАВЛІННЯ ПУБЛІКАЦІЄЮ КОНТЕНТА НА ОСНОВІ МОДЕЛІ SAAS

Пояснювальна записка

ДП.045440-03-81

"ПОГОДЖЕНО"

Керівник проекту:

_____ Р.А. Гадиняк

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ А.І. Мазун

2019

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

MTV – Model Template View;

СКБД – система керування базами даних;

MVC – Model View Controller;

ORM – Object-Relational Mapping;

SQL – Structured Query Language;

БД – база даних;

CRUD – create, read, update, delete;

ПЗ – програмне забезпечення.

ВСТУП

За останні двадцять років Інтернет набрав дуже великої популярності серед людей різних груп, зокрема: учні молодшої/середньої/старшої шкіл, люди середнього та похилого віку, а також студенти. На мою думку, найбільш популярним Інтернет-простір є серед прогресивної молоді. Телебачення та радіо відходить на другий план, а все ж таки Інтернет посідає перше місце по пошуку інформації, по можливості спілкування користувачам в онлайн-режимі за допомогою соціальних мереж, дає змогу, не виходячи з дому, придбати ту чи іншу річ або послугу, надає розважальний контент та багато інших переваг. Ще однією вагомою паревагою Інтернету є те, що кожна охоча людина може слідкувати за новинами своєї та інших країн перед своїм улюбленим гаджетом, будь то телефон, планшет або ноутбук. Мас-медіа заполонили майже весь Інтернет-простір, дуже багато різних новинних порталів, але в кожному з них є свої переваги та недоліки. Серед переваг Інтернет-видавництва можна виокремити наступні:

- публікація останніх новин окремої країни та світу;
- наявність контенту різного характеру;
- простота у використанні;
- доступність.

Але є і недоліки, зокрема:

- найчастіше, відсутність особистого кабінету автора публікацій;
- невідлагоджена система керування «життєвим циклом» статті;
- застарілі технології розроблення, що призводить до повільної роботи порталу.

В даній роботі пропонується розглянути питання розроблення сучасного новинного порталу з дотриманням всіх вимог сучасного веб-розроблення, управлінням персоналу та керуванням публікаціями.

1. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

1.1. Аналіз існуючих програмних рішень

1.1.1. *Ukr.net*

На початок 2019 року ukr.net – найпопулярніший український Інтернет-сервіс новин. Щодня в новинну стрічку на порталі ukr.net надходить близько тридцяти тисяч новин від майже півтори тисячі джерел з усієї України. Серед них – популярні і авторитетні новинні сайти, міжнародні та українські інформаційні агентства, онлайн-версії ЗМІ, регіональні онлайн-ресурси [2]. Новини збираються щохвилини і розміщуються у відповідних тематичних рубриках «Політика», «Економіка», «Події», «Суспільство», «Технології», «Наука», «Авто», «Спорт», «Здоров'я», «Шоу-бізнес», «За кордоном», «Цікавинки», «Фоторепортаж», «Відео», а також новини кожного регіону України. При цьому в новинній стрічці на головній сторінці ukr.net за допомогою алгоритму утримуються найбільш свіжі та актуальні новини. Редакція порталу ukr.net не виробляє новини, а контролює автоматичну рубрикацію і кластеризацію, формування новинних сюжетів, виключає матеріали, що порушують чинне законодавство і морально-етичні норми, принцип недоторканності приватного життя, які містять образи і ненормативну лексику, контент для дорослих, матеріали, що представляють загрозу розвитку дітей, очевидно замовні матеріали і інше у відповідності з редакційною політикою. Новини на порталі ukr.net представлені незалежно від політичної позиції та ідеології Інтернет-сайтів, які їх публікують. Таким чином, читач може вибирати і читати новини, які відображають різні точки зору на одну і ту ж подію.

Отже, серед переваг можна виділити наступне:

- автоматичне збирання новин з простору Інтернет; загалом, ukr.net – це агрегатор новин;
- надання поштового сервісу своїм користувачам.

Недоліки:

- відсутність власного персоналу для роботи з публікаціями, а також журналістського та редакторського складу;
- неоптимізований медіа-контент, що робить роботу сайту дещо повільнішою. Аналіз роботи знаходиться на рис. 1.1.

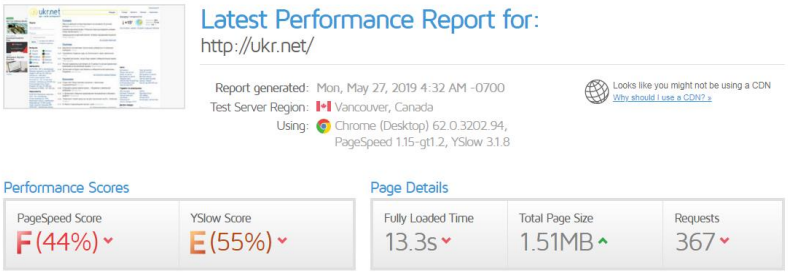


Рис. 1.1. Ілюстрація швидкості завантаження сторінки ukr.net

1.1.2. *112.ua*

112.ua – ще один популярний український Інтернет-сервіс новин. На відміну від ukr.net, розглянутого вище, сервіс не є агрегатором новин. Цей веб-портал позиціонує себе, як авторське видавництво, в якому працює журналістський та редакторський склад. 112 публікує новини та статті на різні тематики та категорії, вони працюють над створенням новин групами людей, що відповідають кожний за свою частину роботи [3]. Основний недолік даного ресурсу – це його швидкість. Через старі інструменти розроблення, на сьогодні можна спостерігати явне уповільнення роботи сайту. Дані щодо аналізу швидкості роботи сайту знаходяться на рис. 1.2.

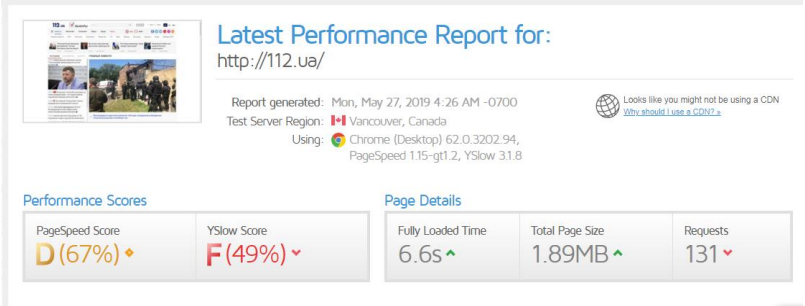


Рис. 1.2. Ілюстрація швидкості роботи 112.ua

1.1.3. *Focus.ua*

focus.ua – українське Інтернет-видавництво, що знаходиться на ринку політичних та соціальних новин/статей з 2000 року. Посідає 40 місце серед всіх інших новинних порталів [1]. Однією з причин такої низької оцінки читачів є те, що було помічено неспівпадіння дати публікацій з поточної датою, були опубліковані новини «з майбутнього». Це призвело до падання рейтингів даного сайту. Можливою причиною помилок даного типу є те, що процес публікації є недостатньо автоматизованим, тому система не побачила і не попередила користувача-адміністратора про помилку публікації. Всі дати публікацій вносились в систему мануально безпосередньо працівниками редакторського відділу і в день публікації статті потрапляли до уваги читачів порталу. Дана помилка відбулась через неуважність редактора, що є звичайним людським фактором. Спостерігається аналогічний недолік, як і у вищерозглянутих – швидкість завантаження сторінки, що можна переглянути на рис. 1.3.

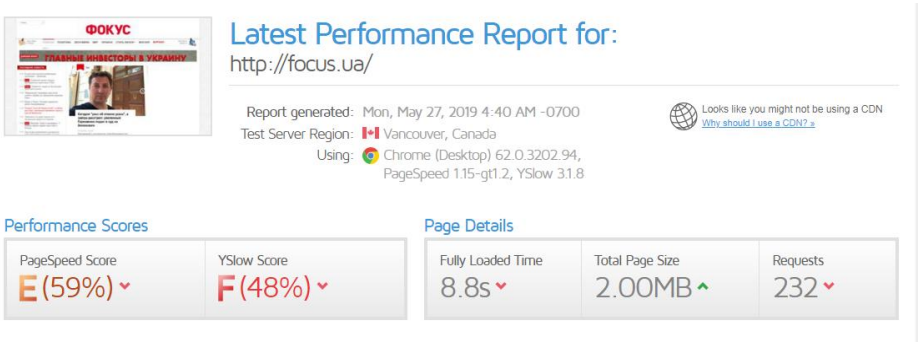


Рис. 1.3. Ілюстрація швидкості завантаження контенту на focus.ua

1.1.4. *Telekritika.ua*

telekritika.ua – це ще один інформаційний портал України, який набирає популярність, в основному серед молодої верстви населення. На відміну від своїх конкурентів та аналогів, які розглядались в попередніх пунктах цього розділу, він відрізняється глобально тим, що на ньому є така роль співробітника як блогер [4]. Блогер – це людина, яка веде свій

власний блог. Блог – це сайт для обговорення подій або інформаційний веб-сайт, опублікований у Всесвітній павутині, що складається з дискретних, часто неформальних, текстових записів у форматі щоденника. Повідомлення, як правило, відображаються в зворотному хронологічному порядку, так що найновіша публікація з'являється спочатку у верхній частині веб-сторінки. До 2009 року блоги, як правило, були роботою однієї особи, іноді невеликої групи, і часто охоплювали одну тему. У 2010-х роках з'явилися «багатоавтоматичні блоги» (MABs), над роботою яких працювали кілька авторів. На блогах розміщують контент у вигляді текстових заміток (постів), аудіо і відеофайлів (відеоблог), а також зображення та фотоматеріали (фотоблоги).

У класичному блозі люди пишуть свої думки, виносять на обговорення громадськості різні проблеми, розповідають про своє життя і т.д.

Одним з найважливіших факторів успішного блогу є особистість автора, автор спілкується з читачами, які в свою чергу коментують його пости, висловлюють свою думку, задають питання, обговорюють. Саме цим блоги відрізняються від звичайних сайтів, де це все не так важливо. Однією з найважливіших функцій блогів є створення довірчих відносин між автором і читачами. Виходячи із вищезазначеного, можна зробити висновок, що успішність та популярність даного сервісу прямопропорційно залежить від контенту, що там публікується. Авторами більш ніж 60% цього веб-додатку є саме блогери. Тому доречно було б реалізувати модуль який дозволяє змінювати порядок виведення блогів кінцевому користувачу, адже, якщо людина буде бачити серед статей, авторами якого є блогери свого фаворита в цій достатньо новій професії, то вона буде читати цей портал. Серед недоліків – швидкість завантаження сторінок. Дані щодо аналізу швидкості знаходяться на рис. 1.4.

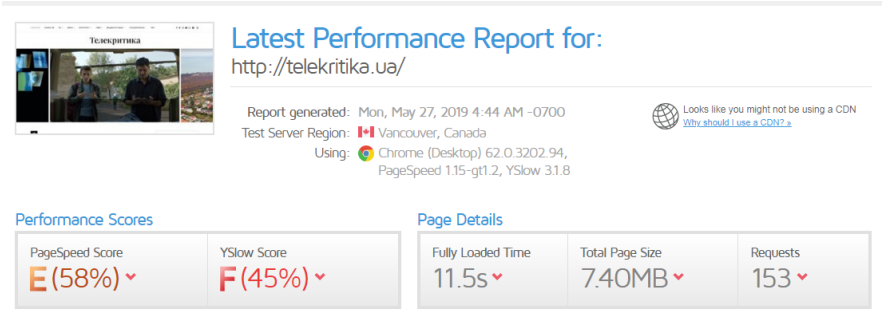


Рис. 1.4. Ілюстрація швидкості завантаження сторінок telekritika.ua

1.2. Опис процесу розроблення новинного порталу

Більшість Інтернет-видавництв повністю або частково зроблено на CMS. CMS – це система управління контентом, набір скриптів для створення, редагування і управління контентом сайту. Відбувається це через те, що CMS підтримують багато готових шаблонних рішень, таких як додавання, редагування, видалення тексту, реєстрація/авторизація користувачів на сайті. Продивившись найпопулярніші CMS на ринку та їх можливостей, можна зробити висновок, що деякі модулі могли б бути корисними для цілей даного дипломного проекту. Але у CMS є свій ряд недоліків, один з яких – дуже велика кількість залежностей різних модулів для відносно невеликого функціонального наповнення, що і призводить до збільшення часу відклику від сервера та недоречності використання [5].

Також пропонується розглянути, на які види поділяються Інтернет-видавництва:

- вузько-регіональні (ознайомитись з регіональними новинами, так наприклад і новинами Київської області);
- широко-регіональні (Україна, Казахстан, Америка і тд.);
- вузько-тематичні (один веб-проект видає матеріали тільки про культурне життя або будівництво, ринок нерухомості і т.д.);
- широко-тематичні (на яких публікуються новини за різною тематикою).

Види Інтернет-видавництв за способом наповнення контентом:

- парсинг (новинні портали – агрегатори);

- користувацький контент (веб-проект наповнюють відвідувачі);
- модерування (новини розміщують редактори та журналісти. При цьому вони можуть брати з інших джерел зі своїми уточненнями, писати самостійно або отримати від акредитованих СМІ);
- змішаний вид.

1.3. Результати аналізу

Огляд існуючих рішень показав, що в новинних порталах є деякі модулі з SaaS – продуктів. Проблемами, з якими зустрічаються розробники новинних порталів, є те, що частіше за все готові модулі та плагіни CMS не дозволяють до кінця розробити бізнес-модель продукту, яку потрібно. Через це програмісти часто змінюють ядро, вже розробленої системи своїми функціями, що призводить до неможливості підтримувати продукт.

Завданням даної роботи є розроблення проекту із можливостями створення сучасного швидкого веб-порталу із зручним користувацьким інтерфейсом, особистим кабінетом для редакторського та журналістських складів, модулем керування «життєвим циклом» публікації для більш зручної роботи редакторського складу, а також модулем керування даними персоналу.

2. ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

Для програмного застосунку, розроблюваного в даному дипломному проекті, необхідно обрати мову програмування для реалізації серверної та клієнтської частини, а також обрати фреймворки, які будуть допомогати у розробленні. Крім того потрібно обрати СКБД для зберігання даних.

2.1. Вибір фреймворку та мови програмування

Розглянемо найпопулярніші мови та фреймворки, які використовуються для розроблення серверної частини web-сервісів.

2.1.1. *Python*

Python є інтерпретованою мовою програмування високого рівня загального призначення. Його мовні конструкції та об'єктно-орієнтований підхід спрямовані на те, щоб допомогти програмістам написати чіткий, логічний код для малих і великих проектів. Python має динамічну типізацію, що дозволяє розробникам користуватись всіма перевагами даного підходу. Він підтримує багато парадигм програмування, включаючи процедурні, об'єктно-орієнтовані та функціональні парадигми. Python часто описується як «батарея включена» через його всеосяжну стандартну бібліотеку. Інтерпретатори Python доступні для багатьох операційних систем в тому числі: MacOS, windows, linux. На сьогодення є дві актуальні версії мови – це Python 2.x та Python 3.x. Python використовує «duck typing» [6]. В ООП-мовах – визначення факту реалізації певного інтерфейсу об'єктом без явної вказівки або успадкування цього інтерфейсу, а просто по реалізації повного набору його методів. Обмеження типу не перевіряються під час інтерпретації; швидше, операції над об'єктом можуть бути невдалими, що означає, що даний об'єкт не є відповідним типом даних. Незважаючи на динамічну типізацію, Python є сильнотипізованою мовою, забороняючи операції, які не є чітко визначеними (наприклад, додаючи число до рядка) і видає на це спеціальну помилку. Python дозволяє програмістам визначати власні типи даних зі своїми властивостями та методами по роботі з ними за допомогою

класів, які найчастіше використовуються для об'єктно-орієнтованого програмування. Нові екземпляри класів створюються викликом конструктора класу, наприклад, Article (title=‘Остання новина’), User (first_name=‘Антон’, last_name= ‘Мазун’, date_birth=‘04.06.1998’).
До версії 3.0, Python мав два види класів: старий стиль і новий стиль.

Синтаксис обох стилів однаковий, відмінність полягає в тому, чи об'єкт класу успадковується від базового типу Python «object», прямо чи опосередковано. Всі класи нового стилю успадковуються від об'єкта і є екземплярами типу. У версіях мови починаючи від Python 2 до Python 2.2, можна використовувати обидва типи класів. Класи старого стилю були вилучені в Python 3.0. Python використовує принцип DRY (do not repeat yourself) [6].

DRY – це принцип розроблення програмного забезпечення, націлений на зниження повторення інформації різного роду, особливо в системах з безліччю шарів абстрагування. Принцип DRY формулюється як: «Кожне рішення повинне мати єдине, несуперечливе уявлення в рамках системи». Він був сформульований одними з розробників. Вони застосовували цей принцип до схем баз даних, планів тестування, збірок програмного забезпечення, навіть до документації. Коли принцип DRY застосовується успішно, зміна в одному елементі системи не вимагає внесення змін в інші, логічно непов’язані, елементи.

Використання мови:

- веб-розроблення;
- Data Science – включаючи машинне навчання, аналіз даних і візуалізацію даних;
- скріптинг.

Так як дана робота присвячена саме веб-розробці, зупинимось на цьому напрямку більш детально. Веб-фреймворки, розроблені на Python такі як Django та Flask останнім часом стали дуже популярними для розроблення.

Django

Django – це високорівнева веб-інфраструктура Python, яка дозволяє швидко створювати безпечні і підтримувані веб-сайти. Побудований досвідченими розробниками, Django піклується про багато проблем веб-розроблення, тому ви можете зосередитися на написанні свого застосування без необхідності вигадувати рішення задач, які розв'язує кожний веб-програміст щодня. Це безкоштовний і відкритий фреймворк, має активну спільноту та чудову документацію. Django реалізує такий архітектурний шаблон, як MTV [8]. Фреймворк допомагає розробнику писати програмне забезпечення, яке:

«Все включає»

Django слідкує філософії «все включено» і надає майже всі основні функції, якими повинен забезпечувати сучасний веб-додаток своїх користувачів, тому багато речей реалізовано «з коробки», наприклад, авторизація/реєстрація користувачів в системі. Оскільки все, що вам потрібно, є частиною одного продукту, все це працює без проблем, слідкує послідовним принципам проектування та є дуже легким у розширені та модернізації уже існуючого продукту [8].

Гнучке

Django може бути використаний для створення практично будь-якого типу веб-сайту – від соціальних мереж і новинних порталів. Він може працювати з будь-якої клієнтської платформою і може доставляти контент практично в будь-якому форматі (включаючи HTML, RSS-канали, JSON, XML). Можливості фреймворку надає вибір практично для будь-якої функціональності, наприклад, кілька популярних баз даних, шаблонізатор, за необхідності він також може бути розширений для використання інших компонентів.

Безпечне

Django допомагає розробникам уникати багатьох поширених помилок безпеки, надаючи інфраструктуру, яка була розроблена для того,

щоб автоматично захистити сайт. Наприклад, Django забезпечує безпечний спосіб управління обліковими записами користувачів і паролями, уникаючи поширених помилок, таких як включення інформації про сеанс в файли cookie, де вона вразлива (замість цього файли cookie містять тільки ключ, а фактичні дані зберігаються в тимчасовій базі даних), або зберігання паролів у відкритому вигляді, замість їх хеш-значення.

Хеш паролю – це значення фіксованої довжини, створене шляхом обробки пароля через криптографічну хеш-функцію. Django може перевірити правильність введеного паролю, пропустивши його через хеш-функцію і порівнявши висновок зі збереженим значенням хеша. Завдяки «однобічного» характеру функції, навіть, якщо збережене хеш-значення скомпрометовано, зловмисникові буде складно дізнатися вихідний пароль. Django забезпечує захист від багатьох вразливостей за замовчуванням, включаючи SQL-ін'єкцію, міжсайтовий скриптинг, підробка запитів і клікджекінг.

Масштабоване

Django використовує компонентну «shared-nothing» архітектуру (кожна частина архітектури не залежить від інших, і отже, може бути замінена або змінена при необхідності).

Зручне у супроводі

Код Django написаний з використанням принципів і шаблонів дизайну, які заохочують створення підтримуваного і багаторазового коду. Зокрема, він використовує принцип Do not Repeat Yourself (DRY), тому немає непотрібного дублювання, що зменшує кількість коду. Django також сприяє угруповуванню пов'язаних функцій в багаторазові «додатки» і на більш низькому рівні групує пов'язаний код модулів відповідно до MTV шаблону.

Переносне

Django написаний на Python, що працює на багатьох платформах. Це означає, що ви не прив'язані до якої-небудь конкретної платформи і

можете запускати додатки в багатьох ОС Linux,Windows і MacOSX. Крім того, Django добре підтримується багатьма постачальниками веб-хостингу, які часто надають певну інфраструктуру і правила розміщення сайтів Django.

Django ORM

Особлива перевага над іншими фреймворками.

Object-Relational Mapping – технологія програмування, яка пов’язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних». Django надає інтерфейс керуванням баз даних через систему класів. Будь-яка сутність в базі даних описується класом, в якому є властивості та методи екземплярів класу. Для того, щоб робити запити в базу даних розробнику не є обов’язковим знати мову SQL. Приклад запиту в базу даних мовою SQL та за допомогою Django ORM: «SELECT * FROM article» та Article.objects.all().

Flask

Мікрофреймворк Flask, як і Django, використовується для розроблення веб-додатків. Ці два, мабуть, найпопулярніші веб-фреймворки Python, і на то є свої вагомі причини, але вони дуже відрізняються в своїх філософіях.

Flask унікальний тим, що він сам позиціонується як «мікрофреймворк», а це означає, що філософія полягає в тому, щоб забезпечити функціональну і лаконічну основу для вашої програми, а залишити все інше для розробників. Приклад, Django буде поставляється з ORM, про яку згадувалось вище для SQL баз даних, що дає вам безпосередню функціональність для запуску екземпляра бази даних, тоді як Flask вимагатиме додаткових кроків для запуску бази даних [10].

Flask вважається набагато більш легким і орієнтованим на швидкий запуск проетку. Це тип фреймворків, які люди люблять використовувати в хакатонах, де ви зосереджені на створенні робочої програми з нуля за короткий проміжок часу і, можливо, менш зацікавлені в довгостроковому

керуванні цією програмою. Між тим, Django є всеохоплюючим фреймворком на відміну від мікро-фреймворку Flask, він поставляється з безліччю нестандартних інструментів, таких як адміністративна панель та шаблони сторінок [10].

Переваги та недоліки Flask.

Неоднозначно можно виділити одну з переваг фреймворку – це те, що він простий у розумінні. Але це не означає, що на ньому якісно розробити продукт. Все залежить від складності системи.

Інщі переваги:

- вбудований сервер розроблення і швидкий відладчик;
- вбудована підтримка модульного тестування;
- відправка RESTful-запиту;
- шаблонізатор Jinja2;
- підтримка безпечних файлів cookie (сеанси на стороні клієнта);
- відповідний WSGI 1.0;
- легкий у розширенні;
- існує багато модулів, з відкритим вихідним кодом.

Недоліки:

- багато глобальних змін;
- не є зручним інтерфейс побудування sitemap, так як, всі роути (Інтернет-адреси) є по суті, декораторами мови і виглядають так – @route(‘/personal-info/’), у випадку, коли проект є дуже великим і планується розширення функціональних можливостей, то дуже важко буде підтримувати його з такою архітектурою.

2.1.2. *PHP*

PHP (Hypertext Preprocessor) – скриптова мова загального призначення, інтенсивно застосовується для розроблення веб-додатків.

В даний час підтримується переважною більшістю хостинг-провайдерів і є одним з лідерів серед мов, що застосовуються для створення динамічних веб-сайтів [11].

PHP замислювався авторами, як простий і доступний інструмент для створення динамічних сторінок. Весь написаний код на PHP, в момент запуску завантаження сторінки, завантажується в пам'ять сервера і виконується інтерпретація коду.

Однією з особливих рис мови є те, що вона має слабку типізацію, що призводить до несподіваної поведінки коду. Так як, мова є цілком без строгого ведення архітектури проекту, то це може призвести до купи повторюваного коду, відсутність модульності в системі, неможливості розширити функціональність якогось компоненту системи. Мова дуже стрімко розширюється і вдосконалюється, але часто програмісти не слідкують за основними простими принципами програмування, наприклад, імена функцій, класів, змінних, відступи в коді, що робить великі проекти часто важкопідтримуваними.

Так, як PHP є дуже популярною мовою серед розробників та Інтернет-провайдерів, є велика кількість готових рішень та наборів функцій для швидкої розроблення, зокрема: Symfony та Laravel.

Symfony

Symfony – вільний фреймворк, написаний на PHP, який використовує патерн MVC [12]. Symfony пропонує швидке розроблення і управління веб-додатками, дозволяє легко вирішувати рутинні завдання веб-програміста. Працює тільки з PHP 5 і вище. Має підтримку безлічі баз даних (MySQL, PostgreSQL, SQLite або будь-яка інша PDO-сумісна СУБД). Інформація про реляційної бази даних в проекті повинна бути пов'язана з об'єктною моделлю.

Це можна зробити за допомогою ORM інструменту. Symfony поставляється з двома з них: Propel і Doctrine. Symfony безкоштовний і публікується під ліцензією MIT. Symfony складається з набору не пов'язаних між собою компонентів, які можна використовувати повторно в проектах. Дозволяє встановлювати сторонні пакети, бібліотеки,

компоненти і налаштовувати їх за допомогою конфігурації в форматах YAML, XML, PHP, а також.env файлах [12].

Серед основних переваг фреймворку є:

- потужна екосистема навколо фреймворка, з хорошим співтовариством і безліччю розробників;
- якісна, актуальна документація для всіх версій фреймворка;
- безліч різних, не пов'язаних між собою, компонентів для повторного використання;
- пропонує механізм функціональних і модульних тестів для знаходження помилок;
- підходить для складних і навантажених веб-проектів електронної комерції.

Серед недоліків можна виділити наступні:

- незважаючи на хорошу документацію, фреймворк є складним для вивчення;
- перенасиченість різного роду сутностями;
- інтегрований аннотаціонний синтаксис, що робить проект при його збільшені важкопідтримуваним;
- повільна ORM (керування базою даних).

Laravel

Laravel – це безкоштовний PHP фреймворк загального призначення з відкритим кодом, який з'явився в 2011 році, але, завдяки стрімким темпам розвитку і величезної кількості програмістів, сьогодні він є одним з найпопулярніших PHP фреймворків. Laravel має дуже багатий набір функцій, які підвищують швидкість веб-розроблення [13].

Переваги Laravel:

- веб-додаток стає більш масштабованим завдяки інфраструктурі Laravel;

- при проектуванні веб-додатку економиться чимало часу, оскільки Laravel повторно використовує компоненти з інших середовищ при розробці [13].

Наведемо особливості використання фреймворку.

Composer – це пакетний менеджер, який включає всі залежності та бібліотеки [13]. Всі залежності зазначаються у файлі composer.json, який поміщений у вихідну папку. Но основі цього, можна зробити висновок, що розробники не мають проблем сумісності з поточними версіями бібліотек, пакетів, сторонніх модулів та фреймворків, що використовуються в проекті.

Модульність

Laravel надає 20 вбудованих бібліотек і модулів, що допомагає вдосконалювати додаток. Кожен модуль інтегрований з менеджером залежностей Composer, який полегшує оновлення [13].

Тестування

Laravel включає функції, які допомагають у тестуванні різних клієнтських помилок. Ця функція допомагає підтримувати код відповідно до вимог.

Маршрутизація

Laravel надає користувачеві гнучкий підхід до визначення маршрутів у веб-додатку. Маршрутизація допомагає краще масштабувати додаток і підвищує його продуктивність.

ORM

Laravel включає в себе конструктор запитів, який допомагає робити запити бази даних, використовуючи різні прості ланцюгові методи, що базуються на об’єктно-орієнтованому підході [13].

Аутентифікаіція

Аутентифікація користувачів є звичайною функцією в веб-додатках. Laravel полегшує розроблення аутентифікації, оскільки містить такі функції, як реєстр, забутий пароль і нагадування про пароль [13].

Управління конфігурацією

Веб-додаток, розроблене в Laravel, буде працювати в різних середовищах, це означає, що конфігурація буде постійно змінюватися. Laravel забезпечує послідовний підхід до обробки конфігурації ефективним способом.

Недоліки Laravel:

- велике функціональне наповнення працює через фасади, коли значення властивостей класів задаються динамічно в залежності від ситуації і IDE-системи не бачать методів і властивостей в деяких класах, показуючи попередження, що робить розроблення незручним [13];
- не містить вбудованих генераторів інтерфейсів, що значно зменшує швидкість розроблення, але дає розробникам можливість самими задавати архітектуру клієнтської частини [13].

2.1.3. *Платформа Microsoft.NET Core і мова програмування C#*

.NET Core – це універсальна платформа розроблення з відкритим кодом, яку підтримує корпорація Майкрософт і співтовариство .NET на сайті GitHub. Вона підтримує Windows, macOS, Linux, може використовуватися для створення локальних додатків на користувацьких комп’ютерах (десктоп) і веб-додатків [14].

.NET Core володіє наступними характеристиками:

- кросплатформеність. Підтримка операційних систем Windows, macOS і Linux;
- однакове виконання коду в різних архітектурах, включаючи x64, x86 і ARM;
- програми командного рядка. Зручні інструменти для локального розроблення і сценаріїв безперервної інтеграції [14];
- сумісність. Платформа .NET Core сумісна з .NET Framework, Xamarin і Mono завдяки .NET Standard [14];
- Відкритий код. Платформа .NET Core має відкритий код і

поширюється за ліцензіями MIT і Apache 2;

- підтримка від Майкрософт. Корпорація Майкрософт надає підтримку.NET Core;.NET Core складається з перерахованих нижче компонентів [14];
- середовище виконання .NET Core надає систему типів, функції завантаження збірок, збирач сміття, власні функції взаємодії. Бібліотеки платформи .NET Core надають примітивні типи даних, типи компоновки додатків і базові службові програми [14];
- середовище виконання ASP.NET надає платформу для створення сучасних хмарних додатків, підключених до Інтернету: веб-додатків, додатків серверної частини мобільних рішень і багато чого іншого;
- засіб dotnet використовується для запуску додатків .NET Core та інструментів командного рядка. Він вибирає середовище виконання, розміщує її, надає політику завантаження збірок і запускає програми та інструменти [14].

Деякі з них роблять суттєвий внесок в проект.NET Core на порталі GitHub і беруть участь в управлінні його розвитком [14].

.NET Core – дуже схожа на інші продукти .NET, але в той же час унікальна платформа. При її розробці ставилася мета забезпечити максимальну адаптованість до нових платформ і робочих навантажень. Платформа має порти на кілька ОС і ЦП, підтримує перенесення на багато інших платформи.

C# – це витончена об'єктно-орієнтована мова з суворою типізацією, яка дозволяє розробникам створювати різні безпечні і надійні додатки, що працюють на платформі .NET Framework і .NET Core. C# можна використовувати для створення клієнтських додатків Windows, XML-веб-служб, розподілених компонентів, додатків клієнт-сервер, додатків баз даних і т.д. Visual C# надає розвинений редактор коду, зручні конструктори користувальницького інтерфейсу, інтегрований відладчик і

багато інших засобів, які спрощують розроблення додатків на мові C# для платформи .NET Framework і .NET Core [15].

Синтаксис C# дуже багатий, але при цьому простий і зручний у вивченні. Характерні фігурні дужки C# миттєво впізнаються всіма, хто знайомий з C, C ++ або Java. Розробники, які знають будь-яку з цих мов, зазвичай дуже швидко починають ефективно працювати в C#. Синтаксис C# значною мірою спрощує складність C++, але при цьому надає відсутні в Java потужні функції, наприклад обнулення типів значень, перерахування, делегати, лямбда-вирази і прямий доступ до пам'яті. C# підтримує універсальні методи і типи, які забезпечують більш високий рівень безпеки і продуктивності, а також ітератори, що дозволяють визначати в класах колекцій власну поведінку ітерації, яке може легко застосувати в клієнтському коді. Вирази LINQ створюють дуже зручну мовну конструкцію для суворо типізованих запитів [15].

Крім цих основних принципів об'єктно-орієнтованого програмування, C# пропонує ряд інноваційних мовних конструкцій, що спрощують розроблення програмних компонентів:

- інкапсульовані сигнатури методів, іменовані делегатами, які дозволяють реалізувати безпечне для типів повідомлення про події;
- атрибути, що надають декларативні метадані про типи під час виконання;
- внутрішні коментарі для XML-документації;
- LINQ для створення запитів до різних джерел даних.

2.2. Вибір СКБД

СКБД – система керування базами даних. СКБД – програми засновані на базах даних і варіанти використання даних (вставка, видалення, редагування, вибірки) [16].

Система є досить безпечною та відповідає всім критеріям цілісності даних.

Основні функції реляційної СКБД:

- безпосередньо управління даними в пам’яті;
- керувати операціями буфера;
- керувати транзакціями;
- підтримка мови SQL.

Класифікація баз даних:

- ієрархічні;
- мережеві;
- реляційні;
- об’єктно-реляційні;
- об’єктно-орієнтовані.

Проаналізувавши вищезазначене, було обрано дві основні СКБД – MySQL та PostgreSQL, які підтримують всі основні функції систем керування баз даних. MySQL – реляційна, PostgreSQL – об’єктно-реляційна. Розглянемо іх можливості. Обидві використовують мову SQL в якості мови запитів до баз даних/таблиць.

SQL – декларативна мова запитів, що застосовується для створення, модифікації та управління даними в реляційній БД, керованої відповідною системою управління базами даних. Є, перш за все, інформаційно-логічною мовою, яка призначена для опису, зміни і вилучення даних, що зберігаються в базах даних [17]. Базові функції мови:

- створення в базі даних нової таблиці;
- зміна структур таблиць;
- додавання в таблицю нових записів;
- вибірка записів з однієї або декількох таблиць (відповідно до заданого умовою);
- зміна записів;
- видалення записів.

2.2.1. *MySQL*

MySQL – є системою управління реляційними базами даних з відкритим вихідним кодом. MySQL базується на моделі клієнт-сервер [18]. Основним моментом використання MySQL є можливості обробки даних. При виконанні запиту MySQL завантажує всю відповідь сервера в пам'ять клієнта, при великих обсягах даних це може бути не зовсім зручно.

У більшості випадків для організації роботи з базою даних в MySQL використовується таблиця InnoDB, ця таблиця представляє з себе B-дерево з індексами [18]. Індекси дозволяють швидко отримати дані з диска, це призводить до зменшення використання обчислювальної потужності. Але сканування дерева вимагає знаходження двох індексів, а це вже повільно.

2.2.2. *PostgreSQL*

PostgreSQL – вільна об'єктно-реляційна система управління базами даних [19].

Основною характеристикою PostgreSQL є використання індексів. У PostgreSQL є підтримка індексів таких як: хеш, B-дерево. При необхідності можна створювати нові типи індексів [19]. Індекси в PostgreSQL мають властивості:

- можливий перегляд індексу не тільки в прямому, а й у зворотному порядку;
- можливе створення індексу над декількома стовпцями таблиці, в тому числі над стовпцями різних типів даних;
- індекси можуть бути функціональними, тобто будуватися не на базі набору значень якогось стовпця/стовпців, а на базі набору значень функції від набору значень;
- планувальник запитів може використовувати кілька індексів одночасно для виконання складних запитів.

Розглянемо основні моменти використання PostgreSQL.

Можливості обробки

PostgreSQL підтримує використання курсорів для переміщення по отриманим даним – отримує тільки покажчик, а вся відповідь зберігається в пам'яті сервера баз даних. Цей покажчик можна зберігати між сеансами. Тут підтримується побудова індексів відразу для декількох стовпців таблиці.

Продуктивність

Вся заголовна інформація таблиць PostgreSQL знаходиться в оперативній пам'яті. Отже, не можна створити таблицю, яка буде не в пам'яті. Записи таблиці упорядковано відповідно по індексу, а тому можно їх дуже швидко витягти.

Для більшої зручності прийнято застосовувати кілька індексів до однієї таблиці. В цілому PostgreSQL працює швидше, за виключенням використання первинних ключів [19].

2.3. Вибір технологій для розроблення клієнтської частини

Сучасний стек розроблення клієнстької частини веб-порталів повністю дає розробникам повну свободу вибору. Розглянемо основні популярні інструменти.

2.3.1. *Jquery*

jQuery – це швидка, невелика та багатофункціональна бібліотека JavaScript [20].

Роботу з jQuery можна розділити на 2 типи:

- отримання jQuery-об'єкта за допомогою функції $(). Наприклад, передавши в неї CSS-селектор, можна отримати jQuery-об'єкт всіх елементів HTML, що потрапляють під критерій і далі працювати з ними за допомогою різних методів jQuery;
- виклик глобальних методів у об'єкта $. Наприклад: $.ajax(), $.each().

Переваги бібліотеки:

- усуває безліч проблем з браузерним javascript за рахунок використання бібліотеки modernizr [20];
- простота у використанні, може виконувати складні операції Javascript в маленькому коді;
- легкий спосіб використання асинхронних запитів до сервера;
- можливість швидко зробити візуальні ефекти сторінки.

Недоліки:

- недоречне використання бібліотеки через те, що стандарти браузерного javascript мають багато функцій, які є в jquery. Завантаження бібліотеки на сайт – це програш в часі завантаження сторінки;
- відстутність реактивних даних, неможливість зручно змінювати структуру DOM.

2.3.2. *Angular2*

Відкрита платформа для розроблення клієнтських інтерфейсів написана на мові TypeScript. Розробляє та підтримує цей інструмент компанія Google. Дуже великий фреймворк, який має багато функцій по роботі з даними. Вся логіка його роботи будується на компонентах, які можуть між собою взаємодіяти, обмінюватись даними та доповнювати один одного. Кожна програма Angular2 має принаймні один компонент - кореневий компонент, який з'єднує ієрархію компонентів з об'єктною моделлю документа (page). Кожен компонент визначає клас, який містить дані та логіку програми, і пов'язаний з шаблоном HTML, який визначає вигляд, який буде відображатися в кінцевому результаті [21].

Декоратор @Component() ідентифікує клас безпосередньо під ним, як компонент і надає шаблон і пов'язані з ним компонентні метадані.

Переваги:

- реактивні дані;
- велика підтримка фреймворку;

- принцип модульності;
- Unit Testing Ready Angular2 (готовий модуль тестування, що є одним з його найголовніших переваг) [21].

Недоліки:

- велика кількість залежностей для коректної роботи;
- нестабільність бібліотек та пакетів;
- відносно довгий час відклику на зміни в коді (перезавантаження сторінки);
- неповна документація, що робить розроблення дуже довгою, тому що виникає необхідність шукати інформацію поза документацією;
- необхідність транслювання TypeScript в стандарти JS ES5, тому що браузери не розуміють мову TypeScript.[21].

2.3.3. *Vue.js*

Фреймворк з відкритим кодом, розроблений відносно нещодавно, перший реліз відбувся 2016 року. Проблему, яку вирішує фреймворк це є саме представлення даних. Добре розширюваний, не має великої кількості бібліотек та залежностей. Досить простий та зрозумілий у використанні. Його ядро в першу чергу вирішує завдання рівня представлення (перегляду), що спрощує інтеграцію з іншими бібліотеками в існуючих проектах.

Переваги:

- невеликий розмір фреймворку;
- успіх фреймворку JavaScript залежить від його розміру. Чим менший розмір, тим більше він буде використовуватися. Одним з найбільших переваг Vue.js є його невеликий розмір. Розмір становить 18–21 Кб і клієнтові не потрібно багато часу для завантаження та використання. Це не означає, що він має низьку швидкість через малий розмір;
- проста структура проекту;
- розробник може легко додати Vue.js до свого веб-проекту через

його просту архітектуру. За допомогою цієї схеми можна розробити як малі, так і великі проекти, що економить багато часу;

- проста інтеграція (можна легко інтегрувати в уже існуючі проекти, побудовані на JavaScript);
- гарна документація;
- наявність реактивних даних;
- гнучкість;
- декларативний рендеринг;
- директиви (всі директиви мають префікс «v-». В директиву передається значення стану, а в якості аргументів використовуються html атрибути або Vue JS події).

Недоліки:

- необхідність транслювання коду в ES5;
- відносно малий час на ринку, не надто велике товариство.

2.4. Результати аналізу

Отже, проаналізувавши переваги та недоліки вищезазаначених технологій та посилаючись на завдання даного дипломного проекту, зроблено наступні висновки. Мовою програмування адмінпанелі обрано Python з фреймворком Django. Ці інструменти були обрані з таких причин.

Python

До переваг даної мови програмування при її виборі саме на даний проект можна віднести:

- інтерпретована мова;
- множинне успадкування;
- динамічну типізацію;
- багата на вбудовані класи бібліотека;
- є дуже зручним для роботи з текстовими даними.

Django

До переваг даного фреймворку при його виборі саме на даний проект можна віднести: строгу структуру проекту, що дозволяє в майбутньому легко розширювати окремі модулі. Наступним плюсом є архітектурний шаблон MTV, що чітко розмежовує бізнес логіку та представлення на сайті. Також до переваг можна додати принцип модульності та DRY (кожен модуль, з яких складається проект є незалежним один від одного) та зручну маршрутизацію - всі роути знаходятся в одному місці - urls.py. Також варто відмітити ORM, яка є досить швидкою моделлю керування базою даних, в якій не використовуються мова SQL, доступ до об'єктів здійснюється за принципами ООП. Швидкий та зручний шаблонізатор.

Оцінивши можливості двох реляційних баз даних – MySQL та PostgreSQL в якості системи керування баз даних в дипломному проекті було обрано СКБД MySQL, тому що:

- дані представленні у вигляді таблиць;
- підтримувана фреймворком Django;
- повнотекстовий пошук;
- є підтримка тригерів;
- нема обмежень у розмірі таблиць;
- є підтримка кирилиці;
- досить прості налаштуванні сервера при запуску проекту в Інтернет.

Для реалізації фронт-енд частини адмін-панелі було обрано наступні технології.

Фреймворк Vue.js

Через свій малий розмір та відсутність допоміжних модулів. Компонентний підхід, що дозволяє зробити один раз компонент, а потім в потрібних місцях його підключати. Простота синтаксису та логіки робити. Зручний життєвий цикл компонента, вбудовані методи дозволяють отримувати данні та показувати їх користувачу в будь-який момент

«життєвого циклу» компонента. Реактивні дані (динамічний контент), зручна маршрутизація.

Клієнтська частина

Для реалізації клієнтської частини додатку було обрано наступні інструменти:

- для виводу даних користувачу – шаблонізатор Django template language, який є зручним, швидким, з можливістю розширити стандартні функції. Головним аргументом чому не використовувати Vue.js в якості інструменту для розроблення фронт-енд частини, є те, що фреймворк має динамічні дані, що робить даний сервіс невидимим для пошукових роботів та CEO, а це є неприпустимо для Інтернет-видавництва;
- для динаміки, що присутня на клієнтській частині було обрано мову JavaScript та бібліотеку Jquery.

3. СТРУКТУРНО-АЛГОРИТМІЧНА ОГРАНІЗАЦІЯ

Для логічної та коректної роботи ПЗ необхідно від самого початку правильно організувати його структуру. Структрурно-алгоритмічна організація впливає на швидкість розроблення та можливість майбутнього розширення системи.

3.1. Аналіз вимог до програмних засобів

Для правильного формулювання вимог до ПЗ, потрібно детально описати продукт. Даний проект пропонується розробляти, як SaaS продукт з дотриманням основних принципів проектування шаблону SaaS та реалізації окремих модулів

3.1.1. *Загальна характеристика системи*

Основне завдання даного сервісу – надати інтуїтивно простий та легкий інтерфейс користувача для керування контентом новинного порталу, оскільки люди, що працюють в журналістській сфері, як правило, не мають поглиблених знань у комп'ютерній сфері або сфері IT. Сучасне Інтернет-видавництво є повноцінним порталом для новин при наявності журналістського та редакторських складів. Система складається з адміністративної панелі керування для кожного працівника та клієнтської частини - тої, що бачить кінцевий користувач-читач. В даній системі окрім редакторів та інших авторів публікацій є роль головного адміністратора, який може створювати категорії, в які будуть публікуватись новини та блоги. Загалом принцип роботи системи та «життєвий цикл» публікації виглядає так, як зображено на рис. 3.1.

Розглянемо те, як публікація потрапляє кінцевому користувачу-читачеві в даному сервісі:

- написання публікації автором;
- відправка на редагування;
- якщо роль передбачає функцію відправки відразу головному редактору, то є можливість відправки головному редактору;

- якщо ні, то відправка до редакторів;
- далі редагування редактором та написання зауважень до автора;
- якщо публікація готова, відправка до головного редактора;
- якщо ні, редактор вносить свої зміни та може відправити до автора;
- головний редактор дивиться на статтю, яку йому відправили, і вирішує, публікувати чи ні;
- якщо стаття готова до публікації, то головний редактор її публікує;
- якщо ні, вносить свої корективи в текст та зауваження до автора або редактора і має можливість відправити статтю на доопрацювання редактору, який займався даною публікацією або безпосередньо до автора статті.

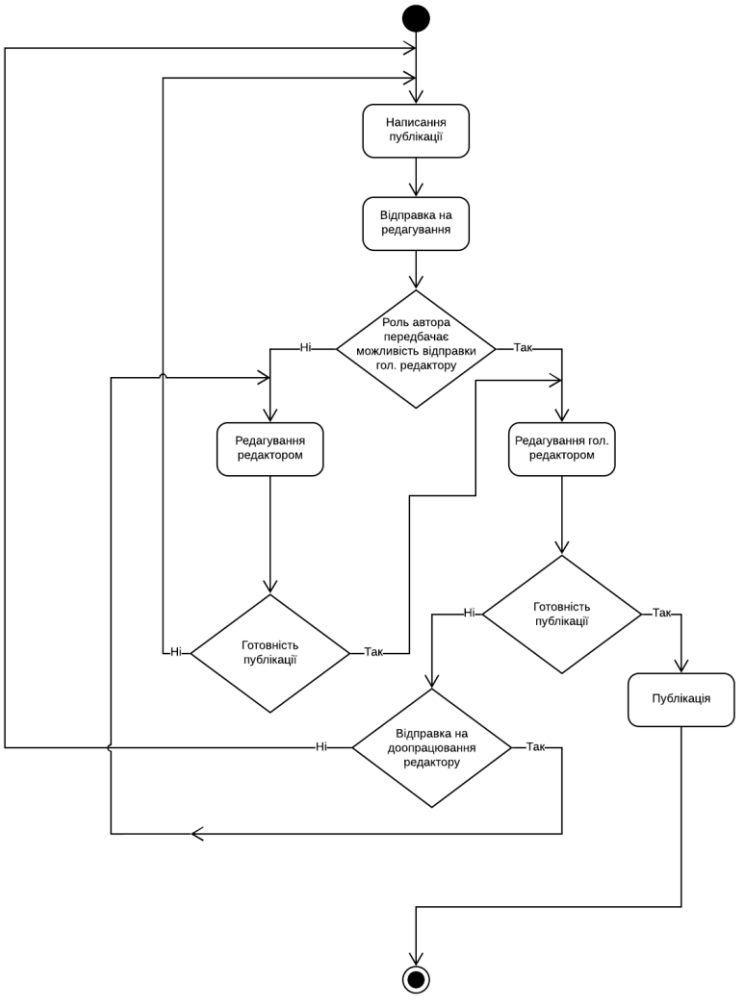


Рис. 3.1. UML-діаграма «життєвого циклу» публікації

Опис адмін-панелі авторського складу

Система повинна передбачати декілька ролей користувачів з різними рівнями доступності до керування нею, серед яких:

- головний редактор;
- редактор;
- журналіст;
- блогер.

Дані про кожну роль та її відповідні функції зберігаються в окремих компонентах системи. Це зроблено для того, щоб у випадку, якщо система зазнає розширення, наприклад з’явиться ще один тип користувача, її можна було б легко розширити. Кожний працівник, який належить тому чи іншому типу користувачів, має свій, незалежний від інших, робочий простір. У даному веб-застонку цей робочий простір є «особистим кабінетом». Перед тим, як приступити до розгляду безпосередньо кожного типу користувача та його можливостей, потрібно навести деякі базові принципи роботи розробленої системи.

Початок роботи з системою

Є декілька способів розпочати роботу з системою.

Перший спосіб – реєстрація користувачем свого акаунта. Користувач заходить під спеціальною Інтернет-адресою (/manage-user/register), яку не знають звичайні користувачі, яку надає головний редактор даного порталу, та заповнює нескладну форму реєєстрації. Після чого йому на вказану електронну пошту приходить лист з пітвердженням реєєстрації. Після цього головний адміністратор сайту або головний редактор призначає даному користувачу один з запропонованих типів користувачів потім він може зайти в особистий кабінет та розпочату роботу.

Другий спосіб початку роботи з системою – реєстрація працівника головним редактором. Головний редактор у відповідному пункті користувацького інтерфейсу може зареєструвати нового користувача, знаючи його поштову адресу. Форма реєстрації виглядає так само, як і

форма реєстрації з першого способу, за винятком однієї важливої речі – пароля. Оскільки інструментами реалізації проекту обрані Python&Django (а вони не зберігають паролі у відкритому вигляді), було прийнято рішення використовувати пароль за замовчуванням – «default_password». Після того, як головний редактор зареєстрував нового працівника, він може передати йому логін-пароль будь-яким зручним для обох способом. Але для більш зручного користування цією функцією було реалізовано можливість відправки листа по e-mail, в якому є вся необхідна інформація для входу до особистого кабінету на ту пошту, яка була вказана головним редактор під час реєстрації працівника.

Головна відмінність обох способів – це те, що у першому користувач потрапляє до системи без конкретної ролі і поки головний редактор не надасть яку-небудь роль із вищезазначених, працівник не може потрапити до особистого кабінету. Саме роль користувача при вході і визначає, в який з компонентів системи буде потрапляти юзер, і якими правами буде наділений.

3.1.2. *Організація роботи з вищеописаними типами користувачів*

У цьому підпункті пропонується розглянути, яким чином користувачі різних типів можуть взаємодіяти з системою. Так як основний принцип розроблення системи – SaaS, а також використовуючи основні вимоги розширюваності, це означає, що всі користувачі мають спільний набір можливостей для роботи з системою, а кожний користувач системи має свої функції, які отримують шляхом розширення базового функціонального наповнення.

Спільні функціональні можливості користувачів

Кожний користувач системи має пройти реєстрацію в системі або скористуватися акаунтом, зробленим головним редактором, який попередньо був надісланий на поштову адресу листом, в якому є вся необхідна інформація для входу в свій особистий кабінет – логін та пароль.

Кожен тип користувача має можливість заповнити про себе особисту інформацію: ім’я та прізвище, посилання на сторінки у соціальних мережах, а також є можливість вибору публікуватись під власним іменем чи під псевдонімом (що є дуже популярним серед журналістів та блогерів). Кожний користувач при бажанні може змінити свій пароль входу у налаштуваннях особистого кабінету.

Спільними можливостями, що наділені всі типи користувачів – є написання публікації та збереження їх в особистому кабінеті для подальшої роботи над ними. Крім цього, користувачі, які наділені правами відправляти на редактування до головного редактора, можуть відправляти безпосередньо до головного редактора системи.

У кожного користувача є можливість перегляду статусу своїх публікацій, де вони зараз знаходяться та хто над ними працює, а також можливість спостерігати за активністю користувачів (у випадку, якщо стаття знаходиться у публікації).

Також для всіх реалізовано функція «Історія змін» новини/статті/блогу. В історії змін користувачі можуть дивитись як саме змінювався контент статті, хто змінював та коли змінював протягом всього життєвого циклу статті. Крім цього, спільною функцією для всіх користувачів є попередній перегляд статті для того, щоб автор міг бачити як саме буде відображена сторінка статті читачеві.

Журналіст

Крім основних функцій, про які було зазначено вище, журналіст може сам вирішувати, коли відправляти свою роботу на редагування до редакторів системи. Можливості типу користувача «журналіст» представлені у вигляді UML-діаграми на рис. 3.2.

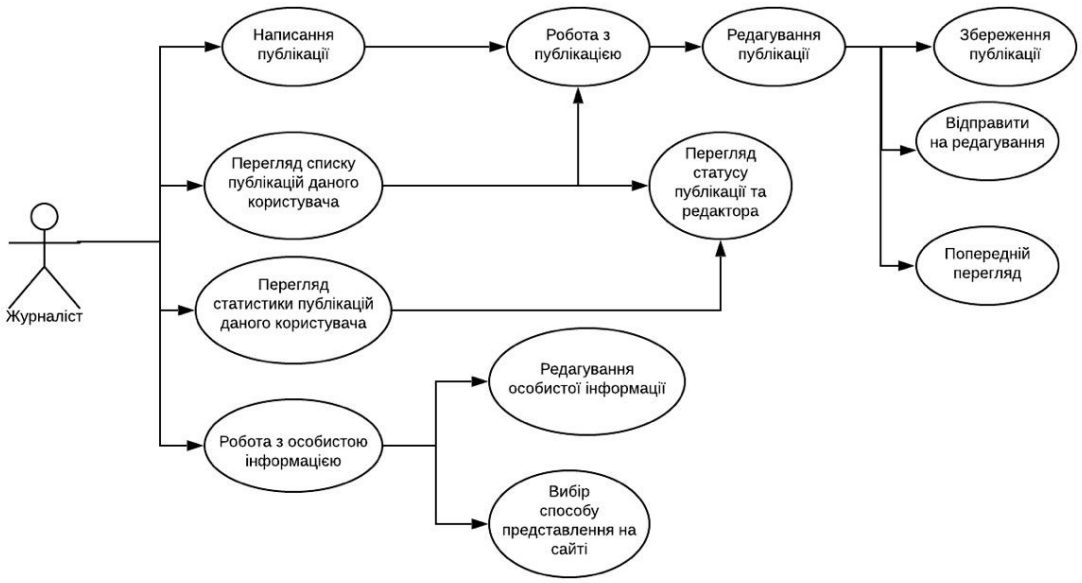


Рис. 3.2. UML-діаграма можливостей журналіста в системі

Редактор

Головною задачею редактора є створення контенту та редагування інших публікацій від журналістів або блогерів. До його функціональних можливостей входять: внесення корективів статті, написання зауважень до автора та повернення йому статті. Корективи безпосередньо будуть бачити автори повернутих публікацій в своїх особистих кабінетах. Саме редактор визначає, відправляти статтю до головного редактора чи ні. Можливості типу користувача «редактор» представлені у вигляді UML-діаграми на рис. 3.3.

Рис. 3.3. UML-діаграма можливостей редактора

Розглянемо можливості редактора більш детально.

Написання публікації

На відміну від журналіста, редактор може як зберегти у своєму кабінеті статтю на доопрацювання в майбутньому, так і відправити на редагування до інших редакторів. Крім того він може відправити статтю до головного редактора без редактури інших редакторів.

Взяти на опрацювання публікацію

Після того як редактор бере статтю в редагування, автор цієї статті бачить свою публікацію з відповідним статусом. Редактор під час корегування тексту статті може вносити туди свої корективи, не боючись, що він зітре оригінал публікації. Разом з тим, він може писати зауваження до автора статті, які будуть відображатись автору статті. Після того, як редактор закінчив редагування статті, він може або відправити її на схвалення головному редактору з урахуванням останніх правок та зауважень, або відправити назад автору із зауваженням та своїми коректівами.

Блогер

Роль блогера в даному новинному порталі є досить простою. Особливою його відмінною рисою від журналіста є те, що він може публікуватись без попереднього редагування своїх робіт, може визначити дату та час публікації, тобто без участі редакторів та головного редактора, якщо таку можливість надасть головний редактор. Також має функцію попереднього перегляду статті. Він може переглядати статистику своїх публікацій та активність користувачів (кількість переглядів, кількість коментарів). Можливості типу користувача «блогер» представлені у вигляді UML-діаграми на рис. 3.4.

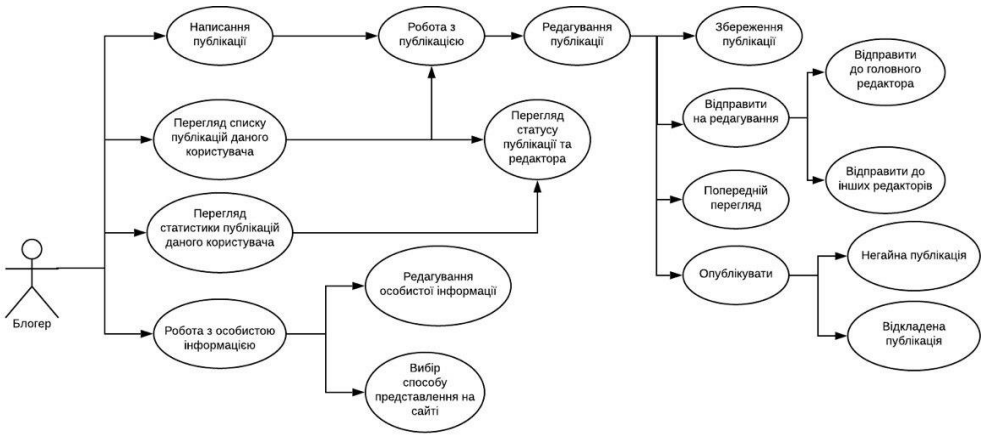


Рис. 3.4. UML-діаграма можливостей блогера

Головний редактор

Головний редактор має найбільш розширені функціональні можливості, такі як: керування даними персоналу, надання прав особам, публікація та зняття з публікації новин, встановлення дати та часу публікації, а також підтримка статті після публікації. Була реалізована можливість взяти статтю в опрацювання головним редактором, відредагувати її, написати зауваження до редактора, який її редагував або безпосередньо до автора цієї статті, а також можливість відправити на доопрацювання статтю до редактора або до журналіста (автора).

Написання публікації

Головний редактор може сам створити статтю, зберегти її у себе в особистому кабінеті, за бажанням він може відправити свою статтю на редагування до інших редакторів. Може без редагування відправити статтю в публікацію з можливістю визначити конкретний час та дату публікації або може опублікувати статтю негайно.

Перегляд списку всіх публікацій

Він має можливість переглянути список всіх публікацій, що є в системі, може проконтролювати інших користувачів, побачити хто чим зайнятий, над якими публікація введеться робота. Також має можливість змінити мануально кількість переглядів та вподобань на статті. Може взяти в опрацювання статтю, яка ще не редагувалась.

Опрацювання статті

Головний редактор може редагувати статті будь-яких авторів: редакторів, журналістів, блогерів. Опрацюванням статті головним редактором можна вважати тоді, коли публікація набуває певного статусу («на опрацюванні головним редактором»). Перший варіант – коли редактор відправляє свою статтю або ту, яка вже була ним опрацьована (наприклад після опрацювання статті журналіста редактором) головному редактору. Другий варіант – коли сам головний редактор бере публікацію на опрацювання. В обох випадках головний редактор має дві можливості розвитку подій. Перший – якщо публікація не є «сирою» та готова до того, щоб бути опублікована кінцевому читачеві, головний редактор може відправити статтю в публікацію. Для цього він має обрати одну з двох функцій – «опублікувати негайно» або «опублікувати в конкретний день та час». Друга – це відправити публікацію на доопрацювання автору або редактору (на вибір, заливши свої корективи та зауваження (за необхідності).

Керування даними про персонал

Головний редактор має сам встановити роль користувачу, який щойно був зареєстрований в системі. Поки він не призначить роль, користувач не має можливості входу в особистий кабінет. Крім того, головний редактор може сам створити нового користувача в системі, заповнивши просту форму реєстрації нового користувача. Задля більш підвищення рівня автоматизації процесу початку роботи нової людини він може відправити логін та пароль на електронну пошту новому співробітнику, яку він вказував при реєстрації. Однією з функцій головного редактора є те, що він може змінювати ролі користувачів, надаючи тим самим певні права та обмеження на користування системою. Також він може «звільняти» персонал, тобто зробити його не активним в новинному порталі, що забороняє доступ до особистого кабінету користувача, в будь-який момент він може поновити роботу цього працівника.

Керування медіа-файлами

Головним редактором може бути завантажений або видалений медіа-контент порталу, тобто фото, відео або інші файли, що пропонуються для використання у статтях. Цей контент може бути використаний усіма користувачами системи для створення своїх публікацій.

Зміна пріоритету виводу блогів

Ця функція стосується тільки блогерів системи. Головний редактор має можливість вирішувати, в якій послідовності будуть показуватися блоги тих чи інших блогерів в списку блогів.

Можливості типу користувача «головний редактор» представлені у вигляді UML-діаграми на рис. 3.5.

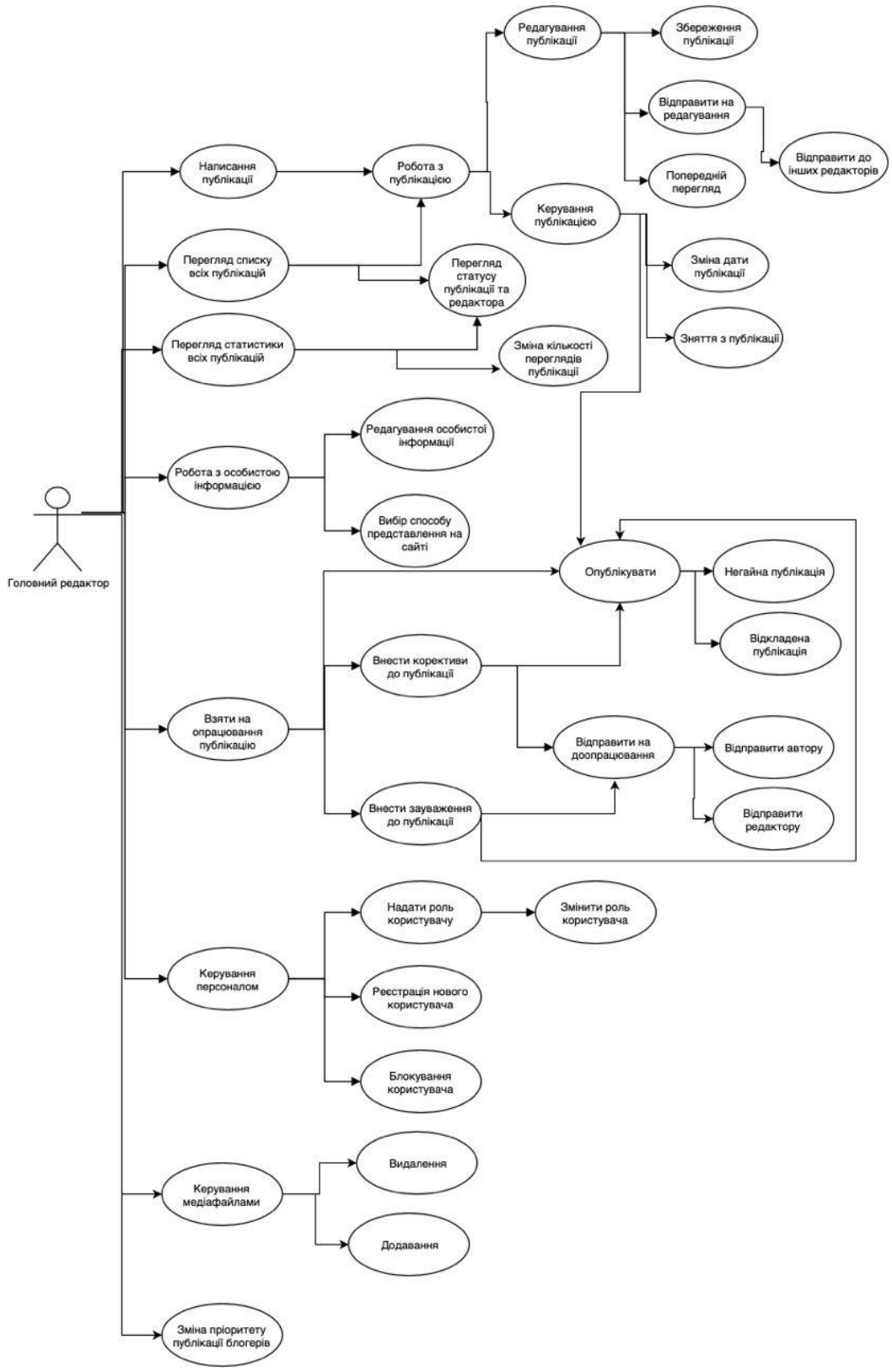


Рис. 3.5. UML-діаграма можливостей головного редактора

3.1.3. *Функції головного адміністратора сайту*

Особистий простір головного адміністратора реалізовано за допомогою вбудованої адмін-панелі, яку надає Django, що є дуже зручним для тих функцій, які потрібні головному адміністратору.

Головний адміністратор має свій особистий кабінет, в якому йому доступні функції щодо керування всім порталом, наприклад:

- створення, видалення та блокування/розблокування користувачів;
- створення статті;
- змінювати автора статті;
- призначати редактора на статтю;
- змінювати статус статті;
- видаляти статтю;
- змінювати статтю;
- створювати, редагувати та видаляти категорії, як для блогерів так і для інших авторів;
- змінювати порядок виведення категорій;
- додавати/видаляти медія-файли;
- надавати користувачам права керуванням сайту.

3.1.4. *Вимоги до новинного порталу*

Вимоги до адмін-панелі авторів

Завданням панелі адміністрування новинного порталу є забезпечення зручної та безперебійної роботи видавництва. Визначимо функціональні та нефункціональні вимоги до даного програмного засобу. Оскільки у всіх типів користувачів є спільні функції в системі, розглянемо функціональні та нефункціональні вимоги до адмін-панелі всіх користувачів та представимо їх у вигляді реєстру у табл. 3.1.

Таблиця 3.1

Реєстр спільних вимог до адмін-панелі всіх користувачів системи

FEAT	Зміст вимоги	Атрибути		
		Priority	Difficulty	Type
F 1	Аутентифікований вхід в систему.	HIGH	HIGH	REQUIREMENT
F 2	Заповнення власної інформації.	LOW	LOW	FUNCTIONAL
F 3	Зміна типу представлення під статею псевдонім/ім’я, прізвище.	MEDIUM	MEDIUM	FUNCTIONAL
F 4	Перегляд статистики.	LOW	MEDIUM	FUNCTIONAL
F 5	Відображення сторінки статистики.	LOW	MEDIUM	INTERFACE
F 6	Перегляд всіх статей конкретного користувача.	MEDIUM	HIGH	FUNCTIONAL
F 7	Зміна паролю.	MEDIUM	MEDIUM	FUNCTIONAL
F 8	Написання статті.	HIGH	HIGH	FUNCTIONAL
F 9	Зміна власних публікацій.	HIGH	HIGH	FUNCTIONAL
F 10	Збереження історії змін статті.	MEDIUM	HIGH	FUNCTIONAL
F 11	Попередній перегляд.	LOW	HIGH	INTERFACE

Вище у табл. 3.1 були описані основні вимоги до всіх типів користувачів. Далі будемо розглядати функціональні та нефункціональні вимоги для кожного з типів користувачів.

Журналіст

Розглянемо функціональні та нефункціональні вимоги до адмін-

панелі журналіста та представимо їх у вигляді реєстру у табл. 3.2.

Таблиця 3.2

Реєстр вимог до адмін-панелі для журналіста

FEAT	Зміст вимоги	Атрибути		
		Priority	Difficulty	Type
F 1	Відправка власної статті на редагування.	MEDIUM	HIGH	FUNCTIONAL

Редактор

Розглянемо функціональні та нефункціональні вимоги до адмін-панелі редактора та представимо їх у вигляді реєстру у табл. 3.3.

Таблиця 3.3

Реєстр вимог до адмін-панелі для редактора

FEAT	Зміст вимоги	Атрибути		
		Priority	Difficulty	Type
F 1	Взяття публікації на опрацювання.	HIGH	HIGH	FUNCTIONAL
F 2	Внесення корективів до статті.	HIGH	HIGH	FUNCTIONAL
F 3	Написання зауважень автору.	LOW	HIGH	FUNCTIONAL
F 4	Повернення автору на доопрацювання.	MEDIUM	HIGH	FUNCTIONAL
F 5	Переведення статті на головного редактора.	HIGH	HIGH	INTERFACE
F 6	Відправка власної статті на редагування.	MEDIUM	HIGH	FUNCTIONAL

Головний редактор

Головний редактор в даній системі є найбільш розвиненим в плані функцій користувачем, він має той самий набір функцій, які були наведені в табл 3.2 і табл. 3.3, однак він має більш розширені функціональні можливості аніж інші, реєстр вимог яких наведено у табл. 3.4.

Таблиця 3.4

Реєстр вимог до адмін-панелі для головного редактора

FEAT	Зміст вимоги	Атрибути		
		Priority	Difficulty	Type
F 1	Визначати роль нового користувача.	HIGH	HIGH	FUNCTIONAL
F 2	Змінювати роль користувача.	HIGH	HIGH	FUNCTIONAL
F 3	Створювати нового користувача.	HIGH	HIGH	FUNCTIONAL
F 4	Блокувати користувача.	LOW	MEDIUM	FUNCTIONAL
F 5	Розблоковування користувача.	LOW	MEDIUM	FUNCTIONAL
F 6	Керування порядком виводу блогів у стрічку.	LOW	HIGH	FUNCTIONAL
F 7	Публікація новини негайно.	HIGH	LOW	FUNCTIONAL
F 8	Публікація новини у конкретний день-час.	HIGH	HIGH	FUNCTIONAL
F 9	Додавання медія контенту у спільний доступ всьому редакторському складу.	MEDIUM	HIGH	FUNCTIONAL
F 10	Видалення медія контенту.	MEDIUM	MEDIUM	FUNCTIONAL

Продовження табл. 3.4

F 11	Попередній перегляд конкретного фото з медія.	LOW	MEDIUM	INTERFACE
F 12	Можливість мануально змінити кіл-сть переглядів та вподобань.	LOW	MEDIUM	FUNCTIONAL
F 13	Сортування публікацій по декільком полям.	MEDIUM	MEDIUM	INTERFACE

3.2. Структурна організація веб-застосунку

Оскільки обраними інструментами розроблення є мова Python та фреймворк Django, це передбачає певну структуру проекту. Вся структура веб-додатку є відображенням моделі MTV – Model-Template-View. Основною філософією Django є те, що у випадку розширення системи чи, навпаки, зменшення потрібно просто відключити той чи інший компонент системи. Ідея реалізації на даних інструментах – це те, що весь проект складається з незалежних додатків, в яких описані сутності, бізнес-логіка та відповідні шаблони, а також кожен модуль має свій маршрутизатор для того, щоб приймати запити від користувача та віддавати йому відповідь.

Основний проект gazeta складається з таких основних компонентів:

- gazeta – основний, кореневий компонент ПЗ;
- article – компонент по роботі з публікаціями;
- categories – компонент по роботі з категоріями;
- manage_user – компонент по роботі з даними про персонал;
- bloger – компонент по роботі з функціями блогера;
- editor – компонент по роботі з функціями редактора;
- journalist – компонент по роботі з функціями журналіста;
- main_editor – компонент по роботі з функціями головного редактора;

- client_app – компонент клієнтської частини.

Охарактеризуємо більш детально кожний з них.

Компонент «gazeta» – кореневий, в якому знаходяться файли, які відповідають за налаштування проекту, за його маршрутизацію та за процес вигрузки проекту в мережу Інтернет. Налаштування проекту включають підключення до бази даних, налаштування маршрутизації по директоріям проекту, шляхи до статичних (css, js, images) та медіа-файлів. Медія файли в контексті даного проекту слід розуміти, як ті файли, що завантажуються користувачами: фото (*.png, *.jpg, *.jpeg, *.svg), відео (*.mp4,*.mov, *.avi та інші). Крім цього, в налаштуваннях є головне – список компонентів, які входять в цей проект.

Компонент «article» – в даному компоненті описано саму модель публікації, хто і як може з нею взаємодіяти. Цей модуль має свою ізольовану бізнес логіку, свої шаблони для представлення та свій маршрутизатор. Також в ньому є описані додаткові функції по роботі з медія-контентом, наприклад, додавання водяного знаку на фото. Так як, фронт-енд частина реалізована на Vue.js, а він не може керувати даними, які було відображені за допомогою стандартних шаблонізаторів, йому потрібно робити запит на сервер та отримувати дані у форматі JSON. Тому у компоненті «article» реалізований модуль, який забезпечує інтерфейс доступу та керування даними – REST API. У цьому компоненті реалізований модуль, який відповідає за збереження життєвого циклу публікації (збереження історії змін).

Компонент «bloger» – він має модуль моделей, з якими він взаємодіє, свою бізнес логіку, свій маршрутизатор та свої шаблони представлення. Загалом компонент реалізує наступні функції:

- можливість написання публікації. Ця функція пов’язана з компонентом «article» та його модулями, так як саме в ньому знаходиться модель публікації та історія змін;

- можливість публікуватися без попередньої модерації (при необхідних правах);
- можливість перегляду та зміни особистої інформації;
- можливість перегляду своїх публікацій та перевіряти їх статуси;
- можливість перегляду статистики по своїм роботам.

Компонент «categories» – в цьому компоненті головний адміністратор має можливість створювати категорії, які будуть присутні на порталі, крім того, редактори та журналісти мають можливість вибирати тільки ту категорію, яку задав головний адміністратор. Крім категорій для редакторів та журналістів, головний адміністратор також задає список категорій для блогерів, у яких можуть бути такі ж самі категорії, як і у решти або можуть відрізнятись.

Компонент «manage_user» – тут є модулі для керування даними про персонал. Деякі модулі з цього компоненту використовуються головний редактором, зокрема, керування списком блогерів – головний редактор може змінювати пріоритети виводу статтей для блогерів (хто стоїть перший в списку- того публікації будуть з'являтись читачеві першими), призначати ролі новим користувачам системи. В цьому компоненті також реалізований модуль реєстрації/авторизації, який є спільним для двох компонентів системи - для клієнтської частини (реєстрація/авторизація читача) та для адмін-панелі персоналу. Модуль реєстрації/авторизації і реалізує функцію маршрутизатора для журналістського та редакторського складів, тобто він перевіряє який тип користувача системи зайшов та переадресовує його у відповідний компонент.

Компонент «editor» – реалізує функції для роботи редактора в системі, а саме:

- можливість написання публікації;
- можливість перегляду та зміни особистої інформації;
- можливість перегляду своїх публікацій та перевіряти їх статуси;
- можливість перегляду статистики по своїм роботам;

- взяття публікації на опрацювання;
- внесення корективів до статті;
- внесення зауваження до автора;
- відправити на доопрацюванню автору;
- відправити статтю до головного редактора.

Компонент «journalist» – описує функції для роботи журналіста в системі.

Функції, що має журналіст:

- можливість написання публікації;
- зберегти статтю для подальшого опрацювання;
- можливість перегляду та зміни особистої інформації;
- можливість перегляду своїх публікацій та перевіряти їх статуси;
- можливість перегляду статистики по своїм роботам;
- відправити статтю до редактора на опрацювання.

Компонент «main_editor» - є найбільшим із запропонованих, він включає в себе модулі з інших компонентів, наприклад, з компонента «article» – написання статті, збереження в історію змін, з компонента «manage_user» –модуль керування користувачами (надання ролей, блокування/ розблокування користувачів), зміну порядку виведення блогів. Крім цього, цей компонент реалізує такі функції для головного редактора порталу:

- можливість написання публікації;
- зберегти статтю для подальшого опрацювання;
- можливість перегляду та зміни особистої інформації;
- можливість перегляду своїх та всіх публікацій та перевіряти їх статуси;
- можливість перегляду статистики по своїм роботам;
- відправити свою статтю до редактора на опрацювання;
- взяти публікацію в опрацювання;

- написати до публікації зауваження;
- внести свої корективи до статті;
- повернути публікації до автора або редактора;
- опублікувати статтю.

Компонент «client_app» – розроблений для кінцевого користувача-читача. В ньому закладена наступна логіка роботи:

- реєстрація/авторизація користувачів (звичайних читачів);
- перегляд всіх новин та блогів;
- перегляд конкретної публікації;
- авторизований користувач має можливість коментувати публікації та відповідати на коментарі інших користувачів;
- користувач може залишити свої дані для того, щоб бути підписаним на новини.

3.3. Опис структур даних додатку

В результаті проектування було виявлено такі сутності.

Користувач (EditorUser):

- електронна пошта;
- пароль – зберігається в зашифрованому вигляді;
- логін – використовується для входу в систему;
- тип користувача – відповідна роль в системі;
- посилання на соціальні мережі;
- спосіб представлення на сайті;
- дата реєстрації.

Категорія (Category):

- назва категорії;
- порядковий номер категорії;
- тип категорії (блог чи стаття/новина);
- slug – юрл-адреса (формується на основі назви).

Стаття (Article):

- заголовок статті;
- підзаголовок статті;
- мета теги статті (для сео);
- головне зображення статті;
- текст під фото;
- текст статті;
- автор статті;
- посилання на джерело;
- інші статті (читайте також);
- виноски (пояснення деяких слів у тексті);
- хто змінював статтю;
- зауваження до автора;
- коментарі;
- дата створення;
- дата останньої зміни статті;
- дата публікацій;
- кількість переглядів.
- статус статті.

Виноски:

- слово чи словосполучення (джерело виноски);
- текст виноски (пояснення).

Коментарі:

- користувач (user);
- текст коментаря;
- дата коментаря;
- id_parent (для рекурсивного виклику);
- level – рівень вкладення;

Зауваження (Remark):

- користувач (user);
- текст зауваження.

Фото для статей:

- дата завантаження;
- файл.

Список для користувачів (user_list):

- тип користувача;
- порядок виводу публікацій;
- користувач (user).

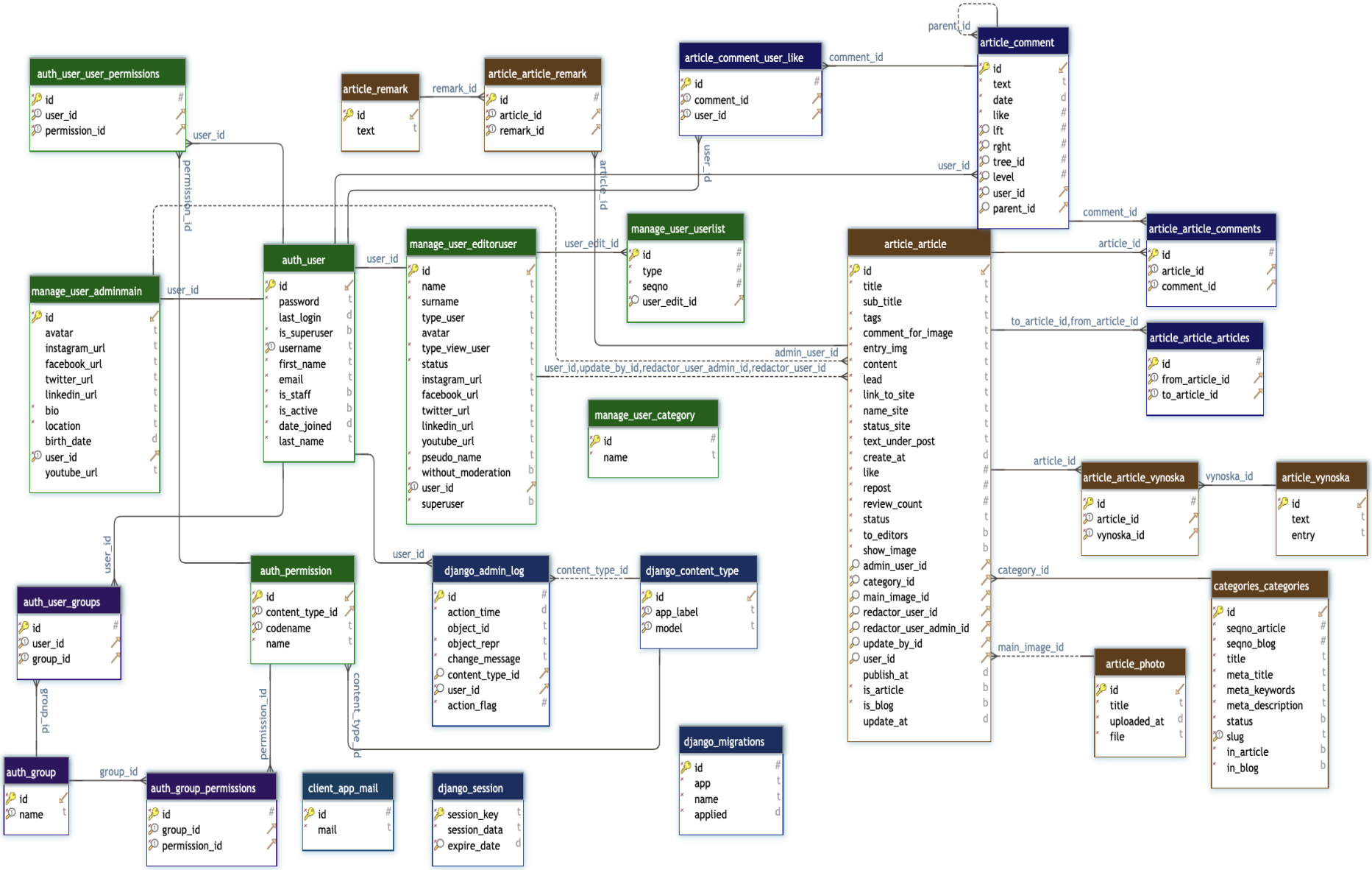


Рис. 3.6. Схема бази даних проекту

4. ОПИС РЕАЛІЗАЦІЇ ПРОГРАМНИХ ЗАСОБІВ

В цьому розділі пропонується розглянути деталі налаштування проекту, а також деталі реалізації окремих компонентів й модулів системи.

4.1. Організаційні налаштування для роботи системи

Для розроблення веб-сервера було використано мову Python фреймворк Django. Серверна частина використовується для обробки запитів, які надходять від користувача, являє собою програмний модуль, що містить засоби:

- авторизації та аутентифікації користувачів;
- обробки HTTP-запитів;
- генерації HTML-сторінок;
- обробки даних, що надходять до користувача;
- взаємодії з базою даних;
- доступ до особистого кабінету користувача;
- зв’язки між компонентами системи.

Оскільки для розроблення фронт-енд частини був обраний фреймворк Vue.js, який включає в себе компонентний підхід та реактивність даних, задля клієнт-серверного зв’язку по API був використаний django-rest-framework, який повертав дані на запит користувача у вигляді json-об’єкта, що потрапляли в якості відповіді у компонент фреймфорку Vue.js, який відображає користувачу у вигляді кінцевої сторінки HTML.

Фреймворк Vue.js має свою екосистему незрозумілу для браузера, було використано специфікацію мови javascript ES6, задля більш строгого формату написання коду з використанням класів та зручного імпорту/експорту даних з інших компонентів. Приклад використання ES6 можна переглянути на рис. 4.1.

```
import Register from './js/Register';
import SetRole from './js/SetRole';
import EditUser from './js/EditUser';
import EditorTab from './js/EditorTab';


class Root {
    constructor() {
        this.register = new Register();
        this.setRole = new SetRole();
        this.editUser  = new EditUser();
        this.editorTab = new EditorTab();

    }
}

var App = new Root();
window.App = App;
```

Рис. 4.1. Приклад використання ES6

Для того, щоб підключити скрипт до сторінки html, необхідно транслювати код фреймворка у зрозумілий для браузера формат – у специфікацію мови javascript ES5. Таким чином у даному веб-проекті файли збираються за допомогою webpack та babel.

Webpack – система збору фронт-енд модулів. Babel-транслятор з ES6 в ES5. Для роботи Webpack потрібен пакетний менеджер npm та конфігураційний файл webpack.config.js, в якому описані команди для запуску та правила збирання модулів. Приклад налаштувань webpack знаходиться на рис. 4.2.

```
'use strict';
const {VueLoaderPlugin} = require('vue-loader');
const MiniCssExtractPlugin = require("mini-css-extract-plugin");
const OptimizeCssAssetsPlugin = require('optimize-css-assets-webpack-plugin');
const path = require('path');

module.exports = {
    mode: 'development',
    devtool: 'source-map',
    entry: {
        admin: ['./front-end/admin/app.js', './front-end/admin/scss/style.scss'],
        client: ['./front-end/client/app.js', './front-end/client/scss/style.scss']
    },
    output: {
        filename: "[name].bundle.js",
        path: path.resolve(__dirname, './gazeta/static')
    },
```

Рис. 4.2. Фрагмент налаштування конфіг-файлу webpack.config.js

Всі залежності для фронт-енд частини знаходяться в папці node_modules, список яких знаходиться у файлі package.json, схематичне зображення структури яких знаходяться на рис. 4.3 та рис. 4.4 відповідно.

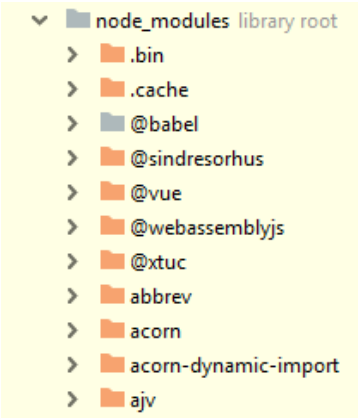


Рис. 4.3. Фрагмент залежностей для фронт-енд частини

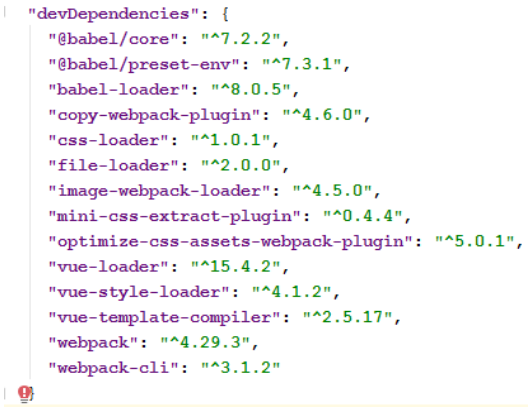


Рис. 4.4. Фрагмент файлу package.json

4.2. Реалізація компоненту «article»

В даному компоненті реалізовані такі спільні модулі для всіх типів користувачів, як models, api_urls, views, watermark, write_article_to_file, serializers.

Модуль models

В даному модулі знаходяться моделі статті та всі інші, які пов’язані із статтею: категорії, автор, статті «читайте також», виноски, зауваження до авторів, ким змінено, фото, коментарі. Зв’язки публікації з іншими сутностями наведені на рис. 4.5.

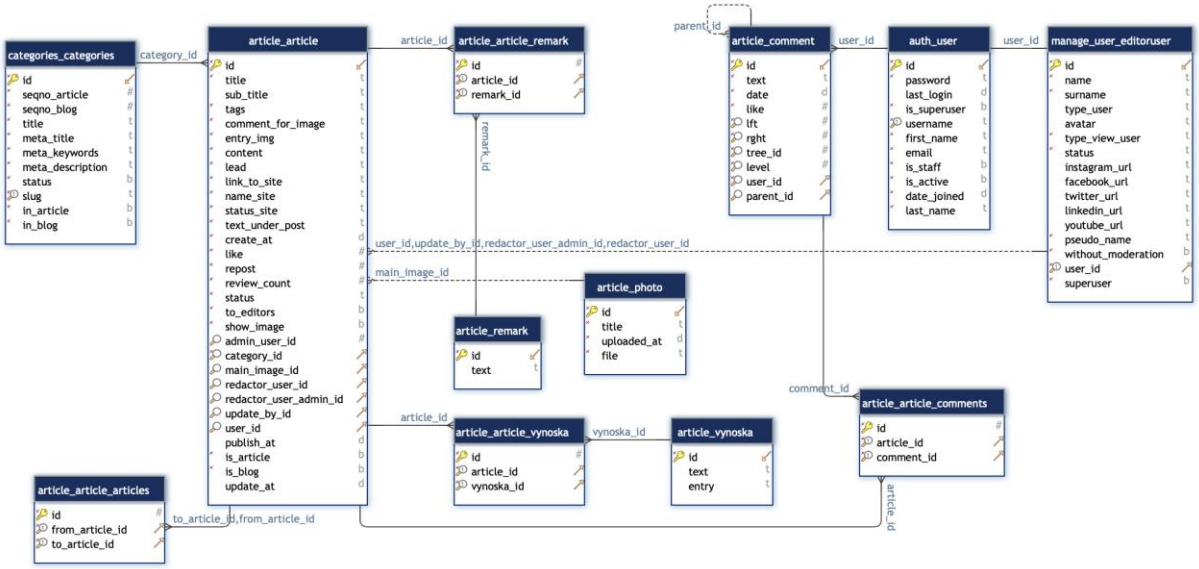


Рис. 4.5. Зв’язки публікації з іншими сутностями

Публікація має декілька статусів, що відповідають за її поточний стан:

- не опрацьована;
- на опрацюванні;
- у головного редактора;
- повернено автору;
- в публікацію.

Перелік всіх статусів представлені у вигляду типу даних кортеж та знаходиться на рис. 4.6.

```
STATUS_ARTICLE = (
    ('inactive', 'Не опрацьована'),
    ('edited', 'На опрацюванні'),
    ('to_main_admin', 'У головного редактора'),
    ('back_to_user', 'Повернено автору'),
    ('done', 'В публікацію')
)
```

Рис. 4.6. Статуси публікації

Однією з головних функцій данного модулю є тектовий редактор, який дає можливість авторам не обмежувати себе при написання статті. Текстовий редактор – це є модуль фреймворка django-ckeditor. Завдяки цьому автор може використовувати в тексті статті будь-які елементи:

- таблиці;
- нумеровані або марковані списки;
- додавати посилання;
- змінювати шрифт, жирність, розмір шрифта;
- вставляти медія-контент;
- вставляти цитати;
- користуватися шаблонами;
- підтримує функцію заміни слів/словосполучень у тексті;
- є функції вирівнювання тексту;
- вибирати колір фону тексту;
- підтримується функція друку;
- є підтримка збільшення вікна для роботи, для більш зручного написання.

Контент, написаний автором, зберігається в БД у вигляді html-строки.

Модуль api_urls

Цей модуль виступає в ролі маршрутизатора. В ньому зберігаються списки всіх доступних юрл-адрес, за якими відбуваються HTTP-запити та перенаправлення у відповідну функцію-контроллер. Список адрес запитів знаходиться на рис. 4.7.

Саме в «api_urls» йде більшість запитів із особистого кабінету користувача.

```
router = routers.DefaultRouter()
app_name = 'articles'
urlpatterns = [
    re_path(r'', include(router.urls)),
    path('user_articles/<int:current>/<int:offset>', views.ArticleByuser.as_view()),
    path('all_articles' , views.AllArticles.as_view()),
    path('all_articles_for_statistic/<int:current>/<int:offset>', views.AllArticlesForStatistic.as_view()),
    path('on-edit-articles/<int:current>/<int:offset>' , views.OnEditUserArticles.as_view()),
    path('length-articles/' , views.LengthArticlesByUser.as_view()),
    path('unset-artilces/<int:pk>' , views.unset_article , name='unset-artilces'),
    path('all-images/<int:current>/<int:offset>' , views.AllImages.as_view()),
    # path('all-media/' , views.all_media),
    path('delete-image/<int:id>' , views.delete_image),
    path('preview_article' , views.preview_article),
    path('on-edit-articles' , views.EditedArticle.as_view()),
    path('edited-articles' , views.ArticlesToAdmin.as_view()),
    path('set-count-type' , views.set_count),
    path('read-more-articles' , views.ReadMoreArticles.as_view()),
    path('read-more-for-article/<int:id_article>' , views.ReadMoreForArticle.as_view())
]
```

Рис. 4.7. Список адрес для запитів

Модуль views

В даному модулі описані класи по роботі з REST-API, які реалізовані за допомогою Django-rest-framework. В модулі реалізовані методи, які повертають список публікацій з різними статусами для конкретного користувача, методи по роботі з медія-контентом (додавання, видалення), методи, які відповідають за взяття статті редактором на опрацювання. Приклад реалізації REST-API знаходиться на рис. 4.8.

```
class EditedArticle(generics.ListAPIView):
    serializer_class = ArticleSerializer

    def get_queryset(self):
        query = Q(status='edited')
        query.add(Q(status='back_to_user'), Q.OR)
        articles = Article.objects.filter(query)
        return articles
```

Рис. 4.8. Приклад реалізації REST-API

Модуль serializers

В цьому модулі описано за якими правилами потрібно серіалізувати об’єкти класів Article, User, Photo, UserList, Category. Для того, щоб

фронт-енд отримував коректні дані з можливістю подальшого їх використання. Приклад класу сереалізації можна переглянути на рис. 4.9.

Серіалізація відбувається у форматі JSON.

```
class ArticleSerializer(serializers.ModelSerializer):
    category = CategorySerializer()
    user = EditorUserSerializer()
    redactor_user = EditorUserSerializer()
    main_image = PhotoSerializer()
    admin_user = MainAdminSerializer()
    redactor_user_admin = MainAdminSerializer()

    class Meta:
        model = Article
        fields = "__all__"
```

Рис. 4.9. Приклад класу сереалізації моделі Article

Модуль watermark

Даний модуль реалізує функцію додавання «водяного знаку» на фото-контент. Це потрібно для того, щоб всі фотографії, що завантажуються авторами порталу були авторськими і не підлягали копіюванню іншими Інтернет-видавництвами.

4.3. Особистий кабінет користувачів

Фреймворк Django використовує свою систему аутентифікації користувачів (auth_user), але вона досить проста – в ній міститься лише базова інформація про користувача, а саме:

- логін;
- пароль;
- електронна пошта.

Так як базової інформації не достатньо для того, щоб описати модель автора, було прийнято рішення використовувати власну модель користувача, яка є розширенням для моделі auth_user з додатковими полями. Дана модель зображена на рис. 4.10.

Роль користувача описано у відповідному полі в базі

даних («type_user») для моделі користувача, що описується спеціальним типом даних в Python – кортежем, елементами якого є кортежі (рис. 4.11), першим елементом вкладеного кортежу є тип даних str, значення якого буде записано в базу даних (рис. 4.12), а другим елементом є також тип str, який буде відображений користувачу у вигляді випадаючого списку зі значеннями елементу кортежу (рис. 4.13).

```
class EditorUser(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE, default=None)
    name = models.CharField(max_length=255, verbose_name='ІмAlt+0146я', default=None)
    surname = models.CharField(max_length=255, verbose_name='Фамілія', default=None)
    type_user = models.CharField(choices=TYPE_USER, max_length=50, verbose_name='Тип користувача', blank=True,
                                 null=True)
```

Рис. 4.10. Поле type_user з атрибутом «choices»

```
TYPE_USER = (
    ('editor', 'Редактор'),
    ('journalist', 'Журналіст'),
    ('bloger', 'Блогер'),
    ('main-editor', 'Головний редактор'),
)
```

Рис. 4.12. Ролі користувачів

```
<label for>Тип користувача</label>
<select name="type_user" id="id_type_user"> == $0
  <option value>---------</option>
  <option value="editor" selected>Редактор</option>
  <option value="journalist">Журналіст</option>
  <option value="bloger">Блогер</option>
  <option value="main-editor">Головний редактор</option>
</select>
```

Рис. 4.13. Значення атрибутів value, що записуються в БД

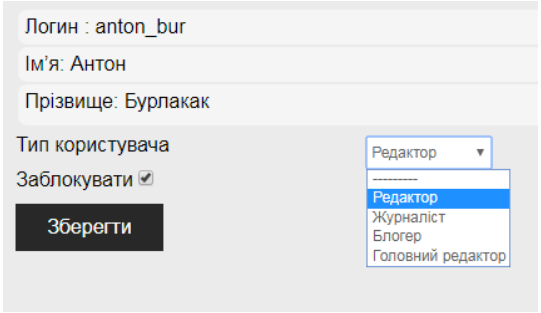


Рис 4.14. Ілюстрація списку вибору ролі користувача для головного редактора

Для того, щоб користувач мав змогу зайти в систему, головний редактор має визначити його роль в системі (рис. 4.14). У кабінеті головного редактора у відповідному пункті меню «Ролі користувачів» може вибрати користувача та назначити відповідну роль, крім цього, в цьому пункті меню головний редактор може бачити всіх активних та неактивних (ті, що не підтвердили e-mail та/або заблокованих користувачів) (рис. 4.15).

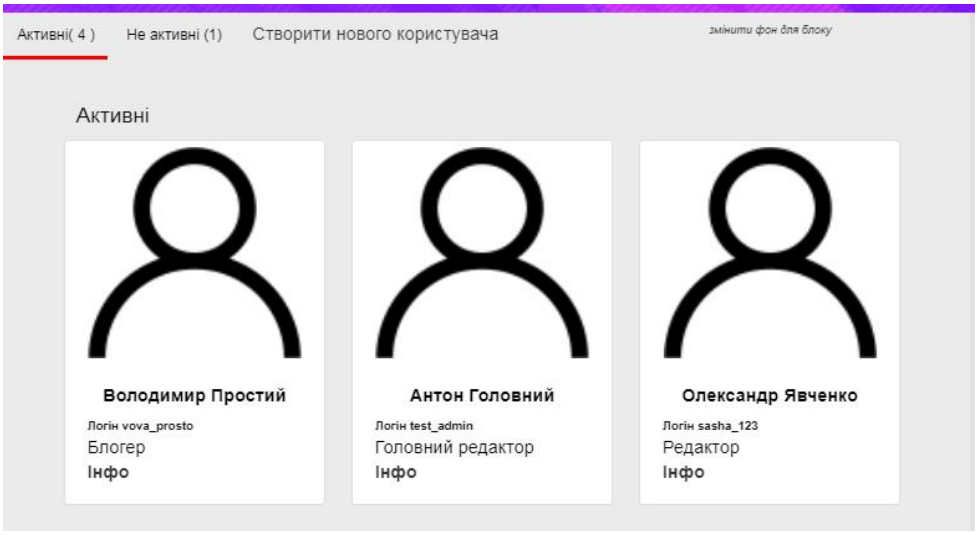


Рис. 4.15. Список всіх активних користувачів для головного редактора

Після того, як користувач має статус в системі, він може здійснити вхід та потрапити до свого робочого простору. Для цього він має заповнити просто форму входу – логін та пароль (рис. 4.16).

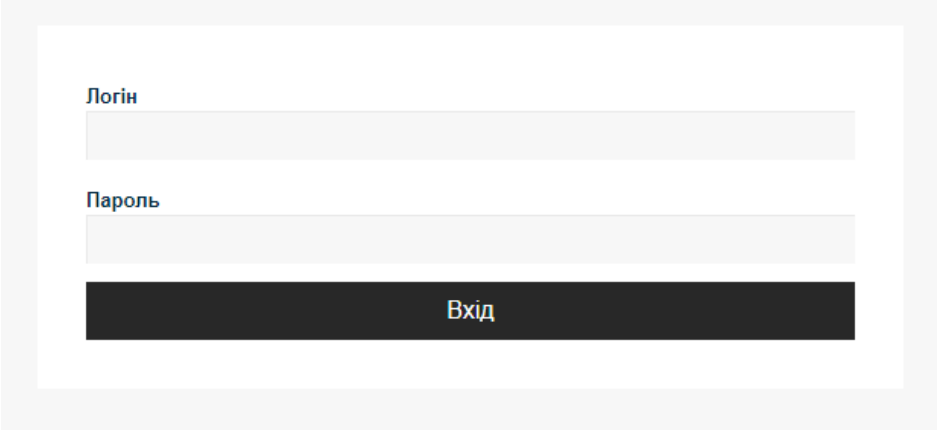


Рис. 4.16. Форма авторизації користувача

Алгоритм роботи модулю авторизації (рис. 4.17):

- користувач вводить свої дані для входу;
- виконується HTTP-запит методом POST на сервер;
- функція-контроллер перевіряє дані на коректність;
- якщо відбулося без помилок, то робиться запит в БД для того, щоб дізнатися тип користувача в системі;
- якщо в вхід у систему відбувся з помилками, користувач отримає відповідне повідомлення;
- потім генерується динамічний юрл і переправляє у відповідний кабінет користувача.

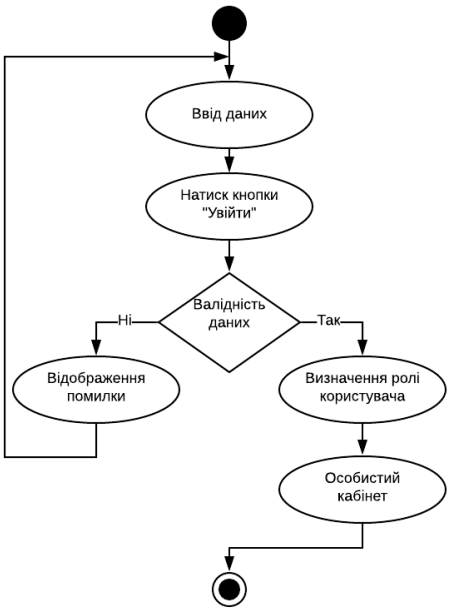


Рис. 4.17. UML-діаграма алгоритму авторизації користувача в системі

Після спішної авторизації в системі користувач потрапляє в свій особистий кабінет, в якому йому доступні пункти навігації по ньому (рис. 4.18):

- ваш профіль;
- налаштування;
- написати публікацію;
- список публікацій.

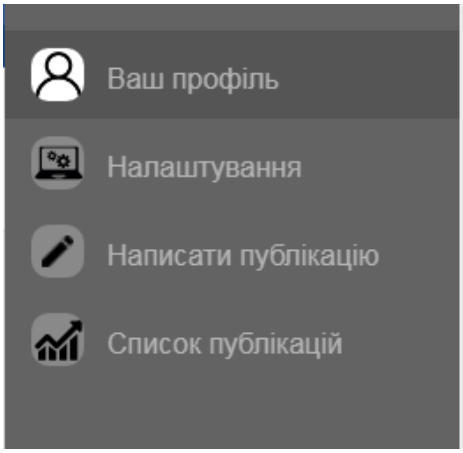


Рис. 4.18. Пункти меню для журналіста/редактора/блогера

На сторінці «Ваш профіль» користувач може бачити свою особисту інформацію та загальну статистику для всіх публікацій, що були опубліковані (рис. 4.19).

Рис. 4.19. Загальна статистика та особиста інформація користувача

Змінити особисту інформацію можна натиснувши на відповідну іконку «редагування особистої інформації». Можна заповнити посилання на соціальні мережі, змінити фото та вибрати тип представлення на сайті: публікуватися під ПІБ або під псевдонімом (рис. 4.20).

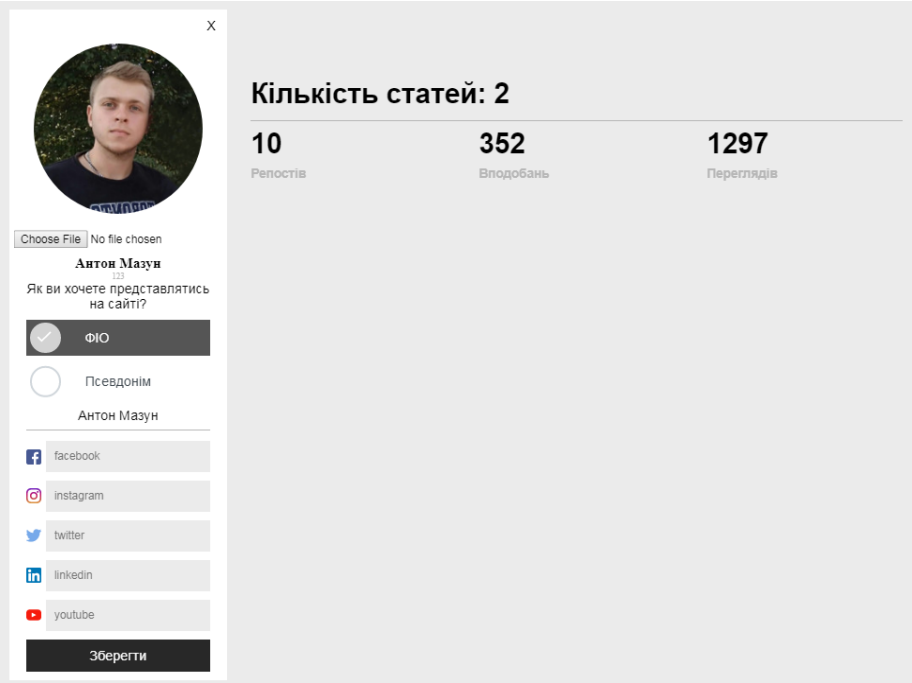


Рис. 4.20. Зміна особистої інформації користувача

Також на цій сторінці можна побачити статус своїх публікацій та скористуватися пошуком по трьом полям: заголовку статті, категорії публікації та даті створення.

На сторінці «написати публікацію» користувач потрапляє на форму написання статті, в якій він обирає одну з запропонованих категорій, пише заголовок статті, невеликий опис (лід), мета-теги – ключові слова, обирає фото для головного зображення статті, фото для репосту в соц. мережі, коментар під фото, джерело, текст публікації, посилання на джерело (у випадку, якщо стаття не є авторською), назва джерела, статус джерела (надійне, ненадійне), може додати виноски (невеликі пояснення деяких слів у тексті), текст під статтею.

Також на цій сторінці розміщені елементи управління:

- прев'ю (попередній перегляд);
- зберегти публікацію;
- можливість відправити до редакторів.

Функція попереднього перегляду збирає всі дані, що були введені користувачем та відкриває нове вікно браузера в якому відображається

публікація так, як вона буде показана читачеві.

Функція збереження статті перевіряє чи автор захотів відправити статтю до редакторів або в публікацію чи ні. У випадку якщо автор не захотів відправляти статтю далі, то стаття зберігається зі статусом «inactive», таким чином ніхто її не бачить, окрім автора в своєму кабінеті. Якщо автор позначив, що хоче відправити статтю до редакторів стаття зберігається з відповідним статусом – «edited» і буде доступна всім редакторам системи для взяття на опрацювання та не буде доступною для редагування самим автором. Якщо редактор відправляє на доопрацювання автору, то стаття буде збережена з відповідним статусом – «back_to_user», якщо головний редактор або автор, що наділений можливістю публікації без попередньої модерації захоче відправити статтю в публікацію, то стаття буде збережена зі статусом – «done».

На сторінці «Список публікацій» користувач може побачити всі свої публікації, відсортувати за певним статусом, за назвою, за кількістю переглядів, вподобань або репостів (рис. 4.21).

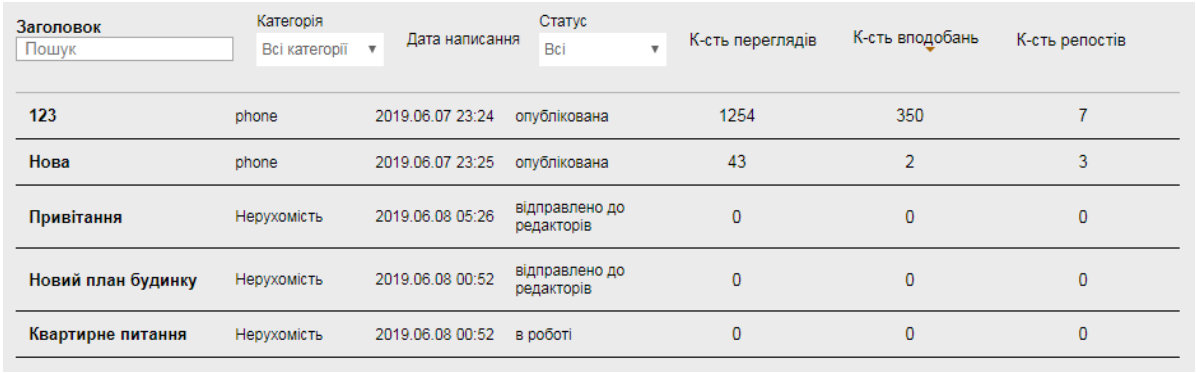


Рис. 4.21. Перегляд списку публікацій користувача

4.4. Модуль «Історія змін»

Цей модуль знаходиться в компоненті «article». Його реалізація є необхідною для авторів новинного порталу. Завдяки функціям цього модулю редактори та автори можуть бачити, які корективи, коли та хто вносив у статтю. Таким чином, можна в будь-який момент повернутись до

статті на певній стадії її «життєвого циклу».

Дані про зміни статті пропонується зберігати у файлі формату JSON, іменем якого є поле id в БД – саме він є унікальним для кожної статті, всі файли знаходяться у відповідній теці на сервері – «news-history» (рис. 4.22).

Рис. 4.22. JSON файли з історією змін

Дані, що містяться у файлі представлені у вигляді об’єкта – словника (тип даних в Python) у форматі «ключ: значення» (рис. 4.23):

- date – коли відбулися зміни;
- content – як змінилася стаття;
- title – як змінився заголовок статті;
- username – хто вносив зміни;
- usertype – який тип користувача вносив зміни.

```
{
  "date": "2019-06-08 11:39:51",
  "content": "<p style=\"text-align:center\"><span style=\"color:#e74c3c\"><strong><u><span
  "title": "\u0424\u0435\u0440\u0430\u0440\u0456",
  "username": "\u0410\u043d\u0442\u043e\u043d \u041c\u0430\u0437\u0443\u043d",
  "usertype": "journalist"
},
{
  "date": "2019-06-08 11:41:50",
  "content": "<p style=\"text-align:center\"><span style=\"color:#e74c3c\"><strong><u><span
  "title": "\u0424\u0435\u0440\u0430\u0440\u0456",
  "username": "\u0410\u043d\u0442\u043e\u043d \u0413\u043e\u043b\u043e\u0432\u043d\u0438\u0
  "usertype": "main-editor"
},
{
  "date": "2019-06-08 11:42:49",
  "content": "<p style=\"text-align:center\"><span style=\"color:#e74c3c\"><span style=\"fo
  "title": "\u0424\u0435\u0440\u0430\u0440\u0456",
  "username": "\u0410\u043d\u0442\u043e\u043d \u041c\u0430\u0437\u0443\u043d",
  "usertype": "journalist"
}
```


Рис. 4.23. Історія змін для статті

Для історії змін був виділений окремий блок у вигляді списку змін у клієнтській частині додатку (рис. 4.24), всі елементи блоку є клікабельні – відкривають діалогове вікно зі списком всіх змін статті (рис. 4.25).

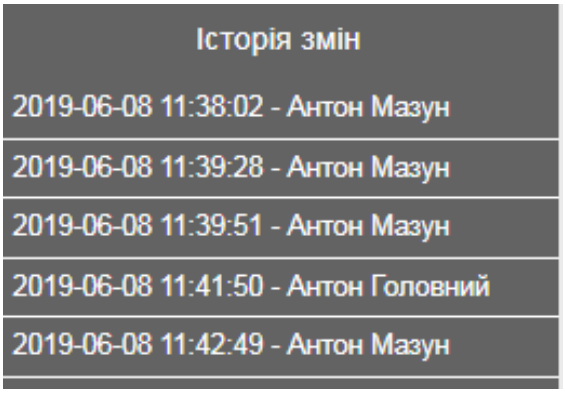


Рис. 4.24. Список змін статті

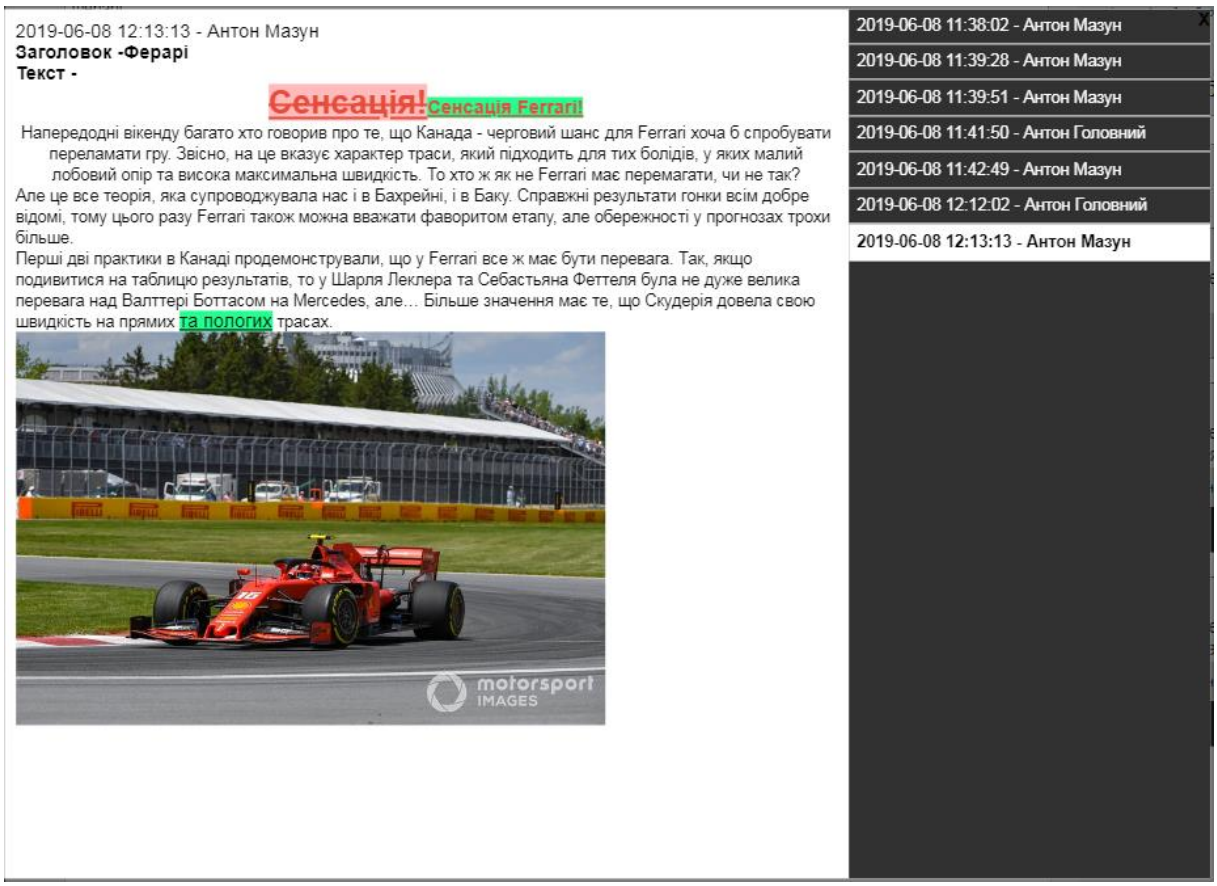


Рис. 4.25. Діалогове вікно зі списком змін публікації

Для подання різниці в тексті було обрані такі інтуїтивно зрозумілі позначення – показувати видалений текст на червоному тлі з закресленим текстом, доданий текст – на зеленому тлі (рис. 4.25).

4.5. Компонент керування даними персоналу

В даному компоненті реалізовані такі модулі:

- модуль реєстрації нового працівника в системі;
- модуль призначення ролей користувача;
- модуль блокування/розблокування користувачів;
- модуль зміни способу виводу статтей для блогерів.

Однією з функцій головного редактора є зміна пріоритету виводу блогів. Для цього був розроблений модуль «Список блогерів». Для його реалізації потрібне REST-API, що поверне дані у json-форматі у компонент vue.js. Для більш зрозумілого використання цього модулю на клієнтській частині було використано drag-n-drop підхід, який відповідає за

переміщення блоків один відносно одного і таким чином формує пріоритет виводу публікацій тих чи інших блогерів (рис. 4.26).

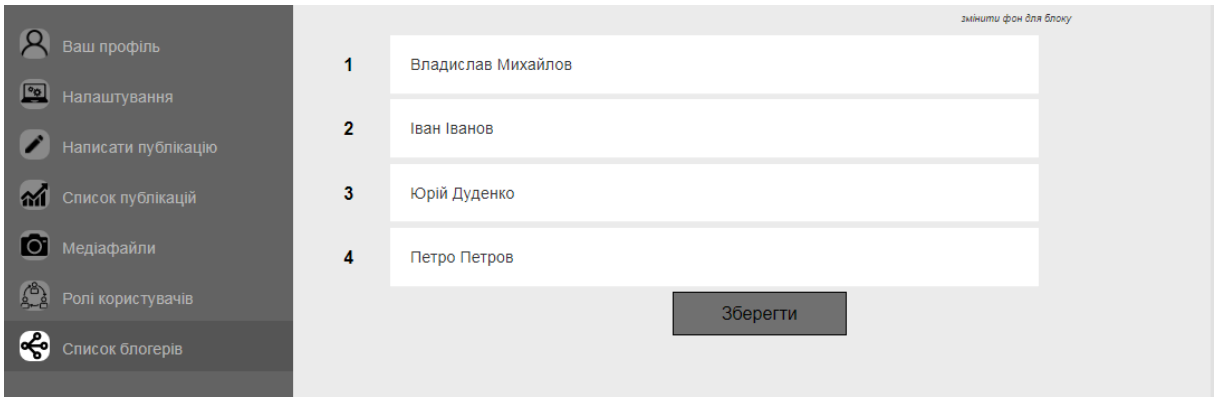


Рис. 4.26. Зміна порядку виводу публікацій блогерів

4.6. Компонент «client_app»

Даний компонент відповідає за все, з чим може працювати звичайний користувач – читач. Головні його функції:

- відображення головної сторінки користувача;
- відображення конкретної публікації;
- відображення останніх новин доби та тижня;
- реєстрація/авторизація в системі;
- вихід із системи;
- фільтрація за категоріями;
- підписка на e-mail розсилку;
- можливість коментувати публікації;
- можливість відповідати на коментарі інших користувачів.

Цей компонент системи має свій маршрутизатор (рис. 4.27) та свій модуль функцій-контроллерів (рис. 4.28). Для виводу публікацій кінцевому користувачу у вигляді html використовується стандартний шаблонізатор Django, який є зручним для цих задач. За допомогою тегів шаблонізатора {% extends ‘template.html’ %}, {% include ‘name_template.html’ %} та {% block content %} {% endblock %} можемо мати дуже зручну та гнучку

систему шаблонів. Завдяки тегам {% include %} та {% extends %} реалізовано основний принцип побудови шаблонів. Тег {% include %} використовується для підключення хедеру та футера на всі сторінки сайту, які є постійно незмінними. Тег {% block content %} {% endblock %} відповідає за змінний контент на сайті, розширюваного шаблону index.html (рис. 4.29).

```
from django.urls import path, include
from . import views
from . import user_views

# app_name='client'
urlpatterns = [
    path('', views.index),
    path('category/<slug:slug>', views.category_item, name='category'),
    path('category-blog/<slug:slug>', views.category_blog, name='category_blog'),
    path('show-article/<int:pk>/', views.show_article, name='show_article_client'),
    path('paginate_on_scroll/<int:current>/<int:offset>/', views.paginate_on_scroll),
    path('search', views.search , name='search'),

]

auth_urls = [
    path('user-login/', user_views.user_login, name='user-login'),
    path('user-register/', user_views.user_register, name='user-register'),
    path('user-logout', user_views.user_logout, name='user-logout'),
    path('subsribe', user_views.subsribe, name='subsribe'),
    path('add-comment/', user_views.add_comment, name='add_comment')
]
```

Рис. 4.27. Маршрутизатор компонента «client_app»

```
def paginate_on_scroll(request, current, offset):
    paginate_article = Article.objects \
                        .filter(category__in=base_context['categories_for_main'],
                                status='done', is_article=1, publish_at__lte=datetime.now()) \
                        .order_by(
        '-publish_at')[current + 1:offset + 1]
    if paginate_article:
        return render_to_response('client/partials/load_on_scroll_articles.html', {
            'articles': paginate_article
        })
    else:
        return JsonResponse({
            'empty': True
        })
```

Рис. 4.28. Приклад функції з модуля «views»

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>
        {% block title %}
        {{ title }}
        {% endblock %}
    </title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    {% block meta %}
    {% endblock %}

    <link rel="stylesheet" href="{% static 'client.min.css' %}">
</head>
<body>
{% include 'client/includes/header.html' %}
<div class="page-wrapper ">
    <div class="page">
        {% block body %}{% endblock %}
    </div>
    {% include 'client/includes/footer.html' %}
</div>

<script src="{% static 'client.bundle.js' %}"></script>

<script>
    App.auth_user.init();
    App.mail_send.init();
</script>
{% block scripts %} {% endblock %}
```

Рис. 4.29. Базовий шаблон сторінки

Всі статті, що бачить користувач, відповідають критеріям: статус – «done», час та дата публікації менші за поточний час.

4.6.1. *Опис клієнтської частини читача*

На кожній сторінці порталу читач має можливість перейти до публікацій певної категорії, всі категорії знаходяться у верхньому блоці сайту (рис. 4.30).

Рис. 4.30. Категорії новин та блогів

На головній сторінці сайту читач побачить у верхньому блоці чотири найсвіжіші новини.

Далі користувач побачить всі решту новин у вигляді різнотипових блоків, які діляться за наступним принципом займання ширини основного блоку:

- 3/3 всієї ширини;
- три новини по 1/3 ширини;
- одна новина на 1/3, друга на 2/3;
- три новини по 1/3 ширини;
- одна новина на 4/4 ширини;
- 6 новин по 1/3 ширині у два рядки.

Потім всі новини підгружаються циклічно за тим же принципом.

Головною особливістю даного підходу є те, що блоки публікацій не мають одного «шаблонного вигляду». Була реалізована функція «lazy load» для того, щоб коли користувач долистає до кінця сторінки, був відправлений асинхронний ajax-запит на отримання більшої кількості новин, які будуть динамічно підвантажуватись користувачу. Але ця функція має обмеження для кількості запитів на підвантаження – максимум всього 60 новин. Крім цього на головній сторінці є блок, в якому знаходяться останні події доби та тижня з можливість фільтрування (рис. 4.31).

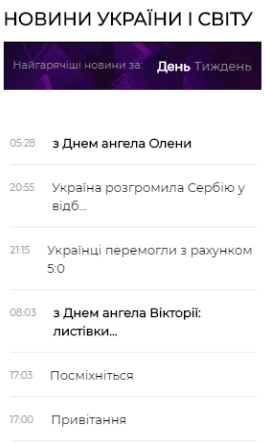


Рис. 4.31. Блок останніх новин за добу/тиждень

Натиснувши на заголовок або зображення на статті, користувач потрапляє на сторінку конкретної статті (рис. 4.31), на якій він може побачити текст статті разом з медіа-файлами, дані про автора статті, кількість перегляді та репостів, блок з новинами за добу/тиждень. Крім того авторизований користувач має можливість залишити коментар під публікацією або під іншим коментарем (рис. 4.32).

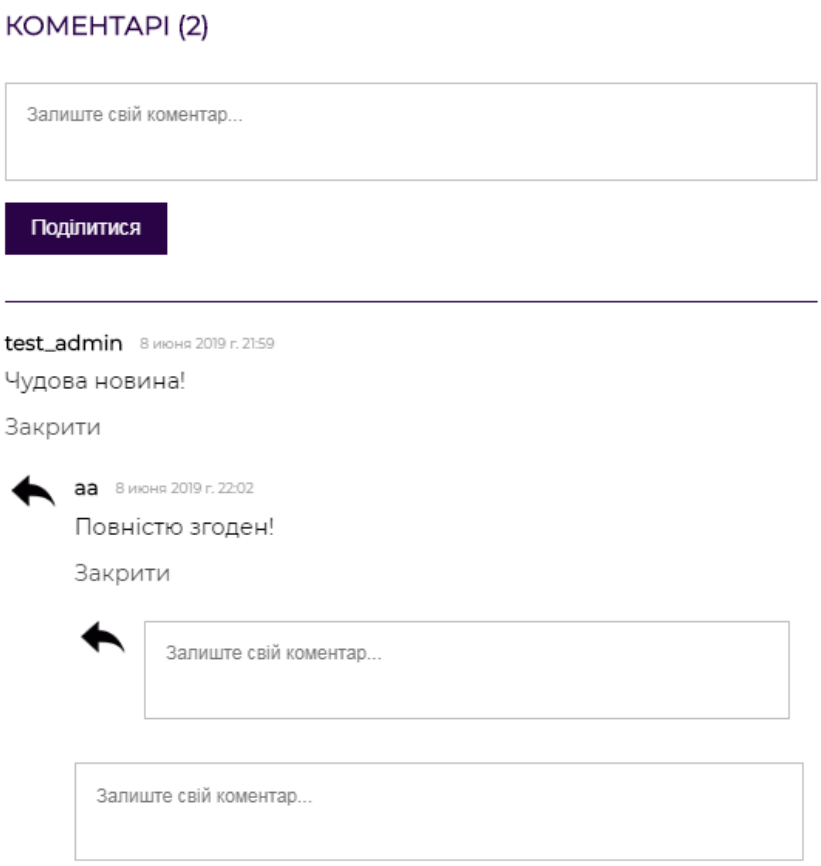


Рис. 4.32. Блок коментарів

Варто пам’ятати про те, що читачі користуються різними гаджетами для користування Інтернетом – від настільних комп’ютерів до малих смартфонів. Це також було враховано при розробці цього компоненту, забезпечуючи адаптивне розположення всіх блоків на мобільних пристроях та комп’ютерах.

ВИСНОВКИ

Метою даного дипломного проекту було розроблення веб-додатку для створення та керування контентом Інтернет-видавництва.

У даній роботі було проведено аналіз доступних програмних рішень, який показав, що більшість аналогів використовує готові модулі та плагіни з популярних CMS, що призводить до важкорозширюваності та важкопідтримуваності продукту. На основі результатів проведеного аналізу було сформовано основні вимоги до технологій розроблення, розглянуто найбільш популярні мови програмування для веб-індустрії, проведено обгрунтування вибору фреймворків для серверної та клієнтської частин, а також СУБД, та обрано оптимальні рішення у вигляді Python Django, django-rest-framework, Vue.js та MySQL. Також було проведено збір та аналіз вимог до програмного застосунку. Після чого було розроблено структуру веб-додатку та архітектуру БД. В останньому розділі даної пояснювальної записки було розглянуто основні моделі системи та їх методи, а також наведено приклади із реалізації інтерфейсу користувача, на який було зроблено великий акцент при розробленні. У результаті даний веб-додаток розроблено у повному обсязі. Також надано рекомендації щодо подальшого вдосконалення та розширення розроблюваного веб-застосунку. Веб-додаток для управління контентом та даними персоналу новинного порталу дозволяє Інтернет-видавництву використовувати його задля більш комфортної роботи з повсякденними задачами (написання та редагування статтей, публікація новин, реєстрація нових працівників тощо).

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

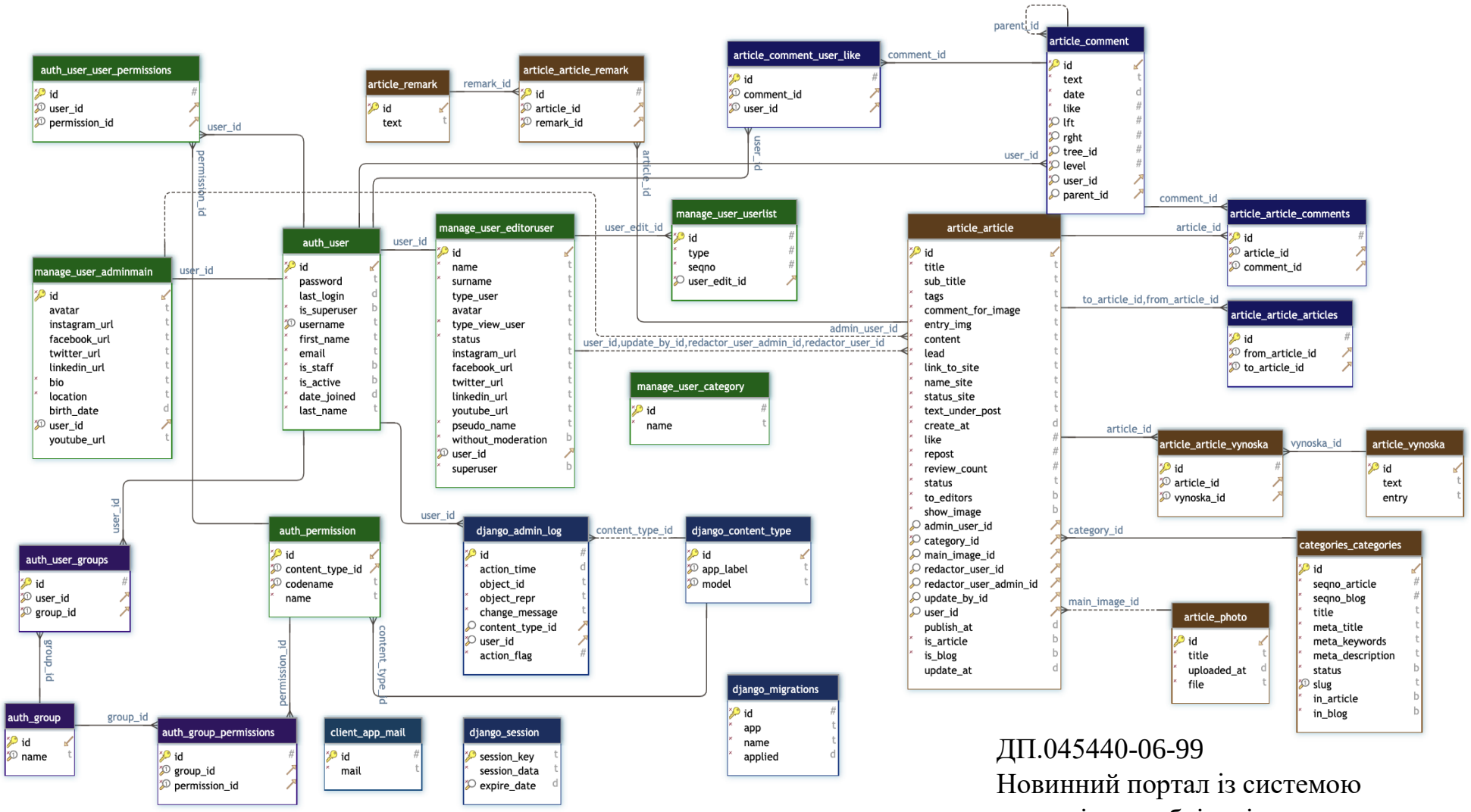
1. Рейтинг новостных сайтов украины: [Електронний ресурс]. – Режим доступу до ресурсу: https://ua-ix.biz/ru/news/ukraina. – (17.12.2018).
2. Ukr.net: [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Ukr.net. – (17.12.2018).
3. 112.ua: [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/112.ua. – (17.12.2018).
4. Телекритика: [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Телекритика. – (17.12.2018).
5. Готовая сборка новостного портала NewsModxBox: [Електронний ресурс]. – Режим доступу до ресурсу: https://habr.com/ru/post/267839/. – (17.12.2018).
6. Python: [Електронний ресурс]. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Python. – (20.01.2019).
7. Don’t repeat yourself: [Електронний ресурс]. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Don’t_repeat_yourself. – (20.01.2019).
8. Что такое Django?: [Електронний ресурс]. – Режим доступу до ресурсу: https://tutorial.djangogirls.org/ru/django/. – (20.01.2019).
9. Введение в ORM (Object Relational Mapping): [Електронний ресурс]. – Режим доступу до ресурсу: http://internetka.in.ua/orm-intro/. – (20.01.2019).
10. Flask (web framework): [Електронний ресурс]. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Flask_(web_framework). – (20.01.2019).
11. Что такое PHP?: [Електронний ресурс]. – Режим доступу до ресурсу: http://www.php.su/php/?php. – (20.01.2019).
12. Symfony: [Електронний ресурс]. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Symfony. – (20.01.2019).

13. Laravel – экосистема, а не просто PHP-фреймворк: [Електронний ресурс]. – Режим доступу до ресурсу: https://habr.com/ru/post/334776/. – (20.01.2019).
14. NET Core: возможности и перспективы: [Електронний ресурс]. – Режим доступу до ресурсу: https://dou.ua/lenta/articles/net-core/. – (20.01.2019).
15. Why Learn C#?: [Електронний ресурс]. – Режим доступу до ресурсу: http://www.bestprogramminglanguagefor.me/why-learn-c-sharp. – (20.01.2019).
16. Система управления базами данных: [Електронний ресурс]. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Система_управления_базами_данных. – (27.01.2019).
17. SQL: [Електронний ресурс]. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/SQL. – (27.01.2019).
18. MySQL: [Електронний ресурс]. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/MySQL. – (27.01.2019).
19. PostgreSQL: [Електронний ресурс]. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/PostgreSQL. – (27.01.2019).
20. What is jQuery?: [Електронний ресурс]. – Режим доступу до ресурсу: https://jquery.com. – (27.01.2019).
21. AngularJS: [Електронний ресурс]. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/AngularJS. – (27.01.2019).

ДОДАТКИ

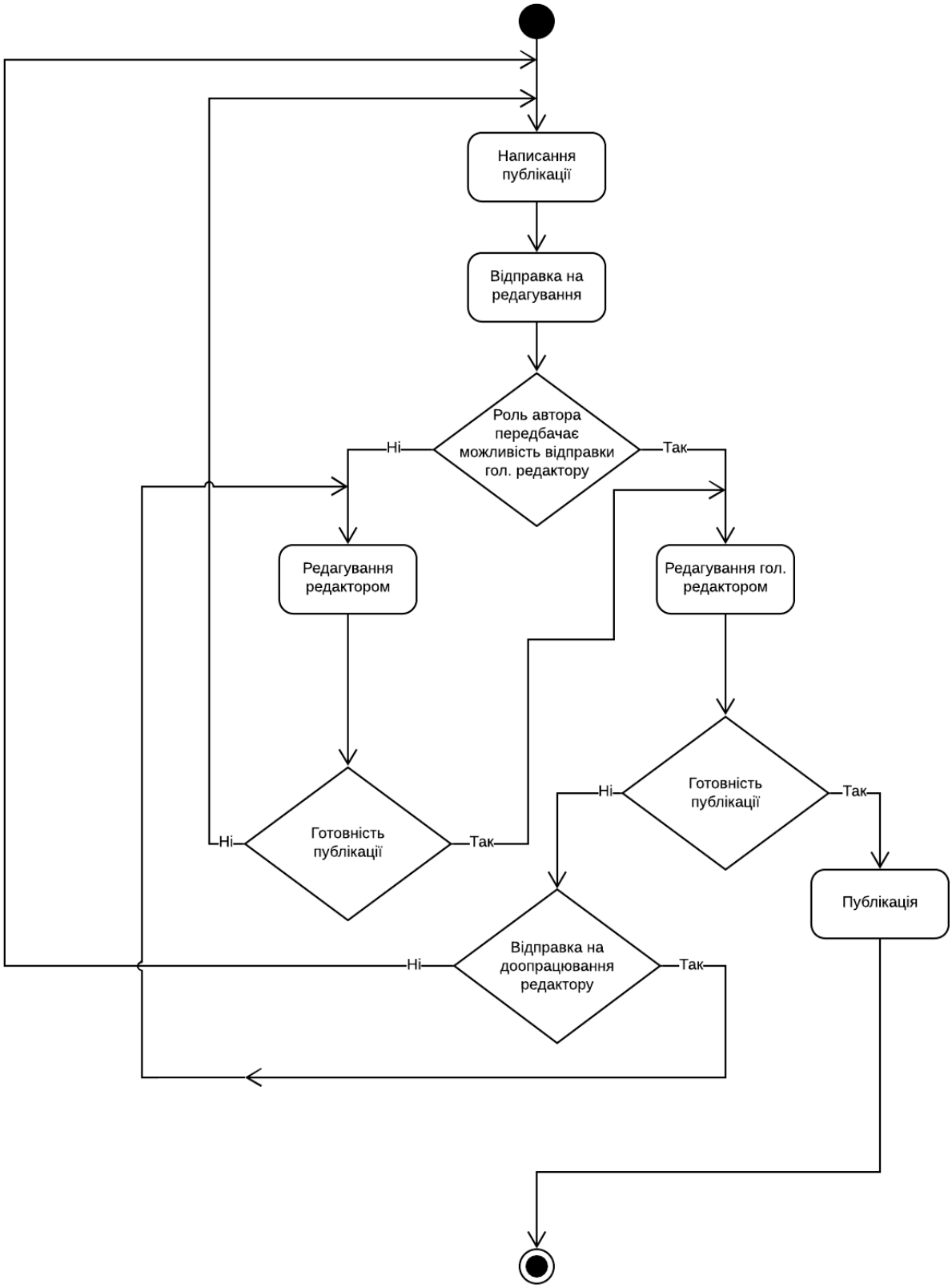
Додаток 1

Копії графічних матеріалів

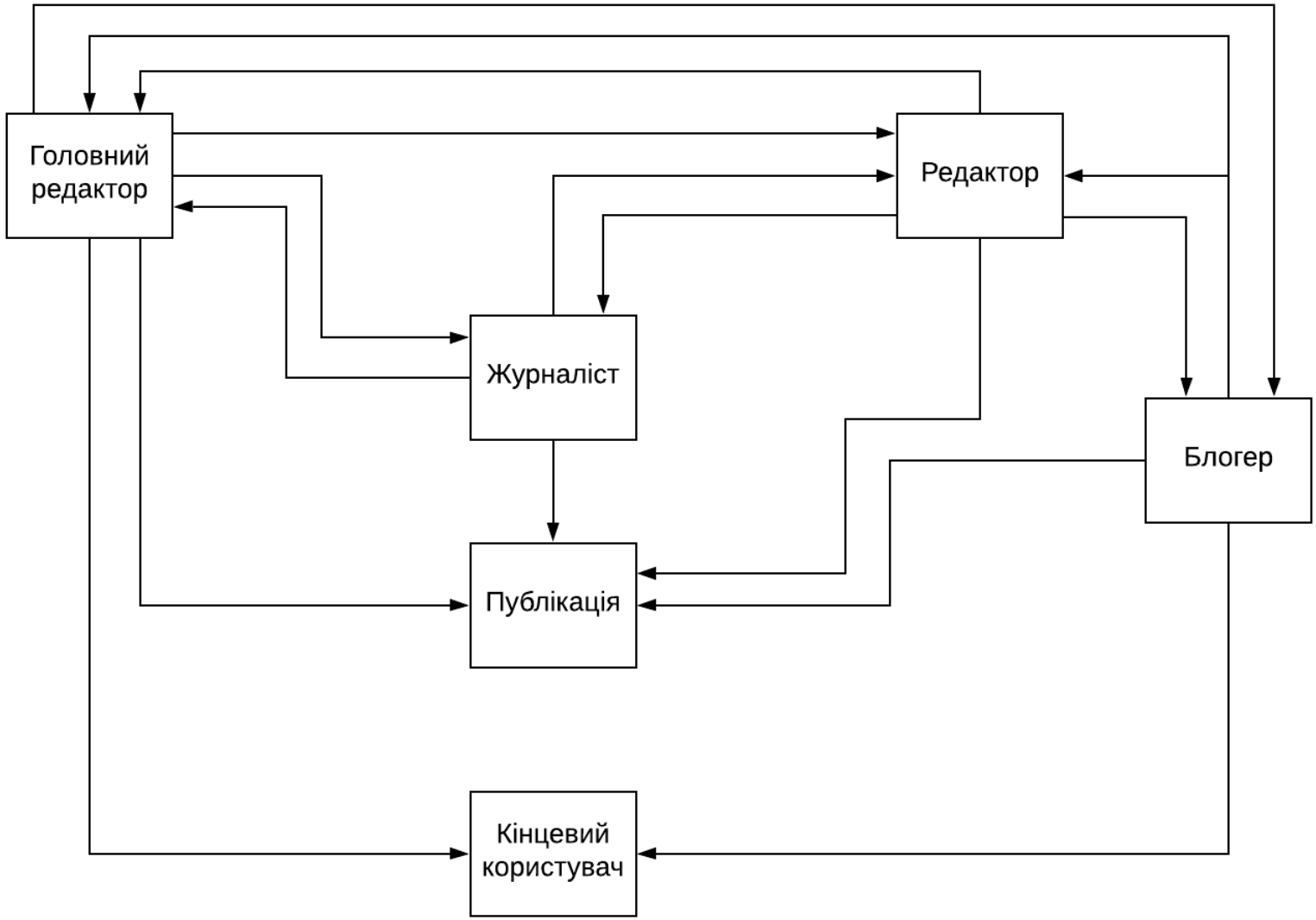


ДП.045440-06-99

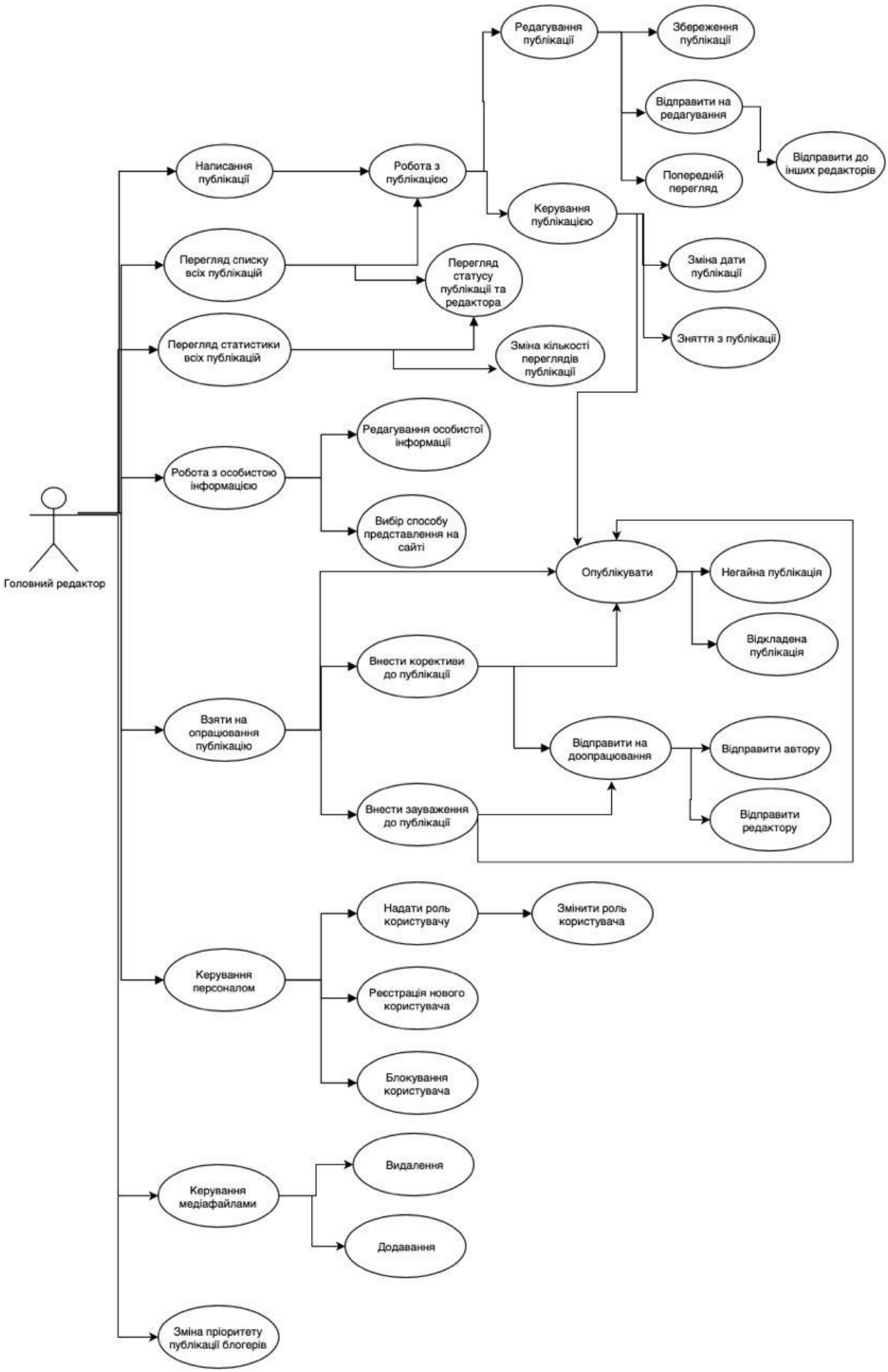
Новинний портал із системою управління публікацією контента на основі моделі SaaS. Структура бази даних. ERD діаграма



ДП.045440-07-99
Новинний портал із системою управління публікацією контента на основі моделі SaaS. Життєвий цикл публікації. UML-діаграма



Мазун А.І., група КП-51



Мазун А.І., група КП-51

Додаток 2

Копія презентації

НОВИННИЙ ПОРТАЛ ІЗ СИСТЕМОЮ УПРАВЛІННЯ ПУБЛІКАЦІЄЮ КОНТЕНТА НА ОСНОВІ МОДЕЛІ SAAS

Мазун Антон, КП-51
Науковий керівник: ст.викл. ПЗКС ФПМ Р.А. Гадиняк

Актуальність

Існуючі рішення

Недоліки

- Модулі з CMS.
- Неповнота або відстутність особистих кабінетів працівників.
- Через використання CMS є проблеми з підтримкою та розширенням продукту.

Постановка задачі

- Мета роботи

  Основною метою роботи є розроблення новинного порталу із системою управління публікацією контента на основі моделі SaaS із забезпеченням більшої гнучкості та можливості розширюваності системи.

- Постановка завдання
  - Розроблення архітектури проекту із забезпеченням можливості розширення системи.
  - Розроблення функцій особистого кабінету для користувачів різних ролей («блогер», «редактор», «головний редактор», «журналіст»).
  - Розроблення модулю «Історія змін».
  - Розроблення модулю керування даними про персонал.
  - Розрозроблення способу взаємодії між користувачами.

Засоби реалізації

Основні

Допоміжні

Вимоги до продукту

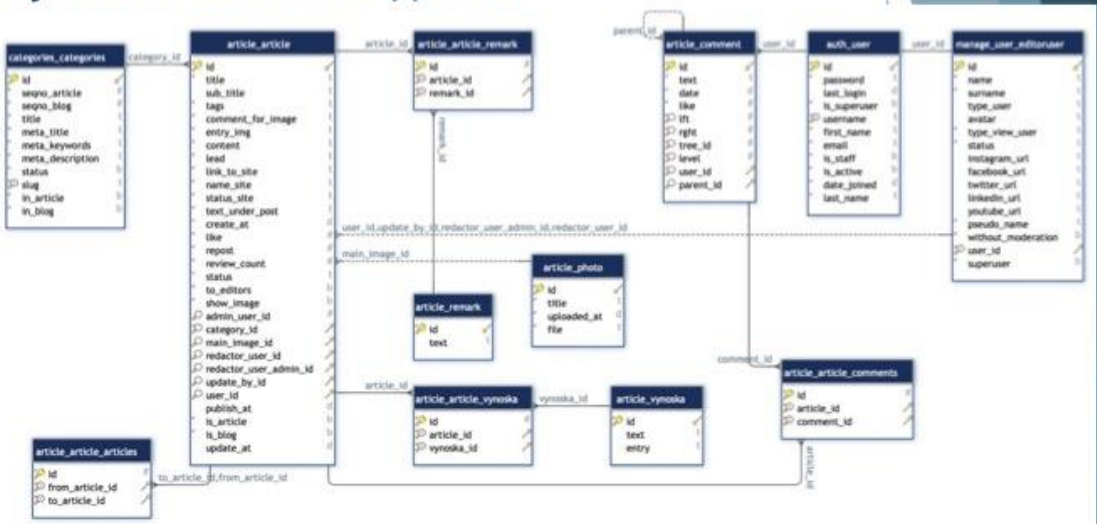
- Відстутність модулів з CMS.
- Архітектура проекту, що базується на компонентному підході.
- Наявність декількох ролей працівників в системі:
  - Журналіст;
  - Редактор;
  - Блогер;
  - Головний редактор.
- Наявність особистого кабінету користувачів з певними правами.
- Забезпечення взаємодії між користувачами.
- Можливість головного редактора працювати зі статусами публікації.

Вимоги до інтерфейсу для читача

- Інтерфейс читача повинен бути зрозумілим для нього та надавати можливість легкої навігації сайтом.
- Читач повинен мати можливість виконати вхід в систему з будь-якої сторінки порталу, щоб залишати коментарі під статтями.
- Читач повинен мати можливість залишити свій e-mail для розсилки зі свіжими публікаціями.
- Інтерфейс користувача повинен бути адаптивним.

Діаграма зв’язків компонентів системи

Зв’язки сутності «Публікація» з іншими сутностями в базі даних

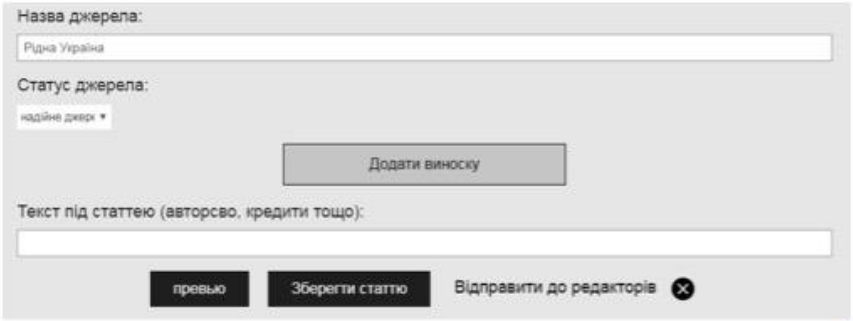


Діаграма прецендентів

«Життєвий цикл» публікації

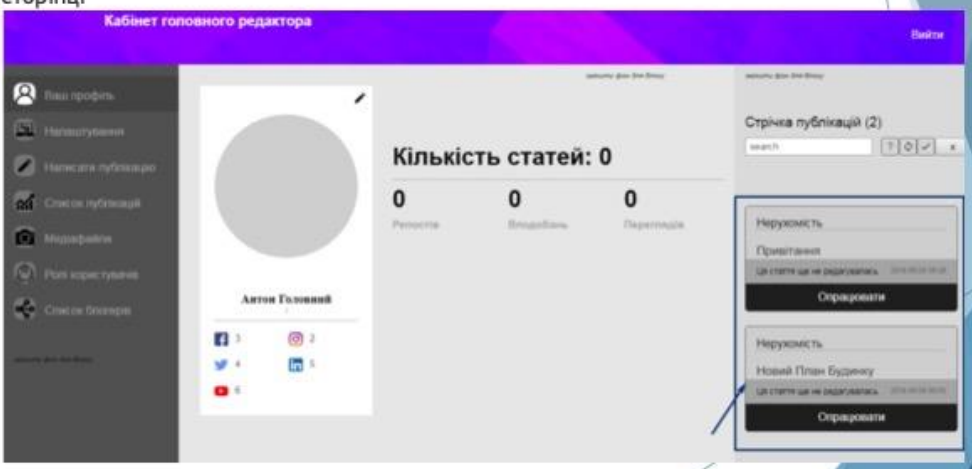
Приклади роботи

Журналіст пише публікацію та відправляє її до редакторів.



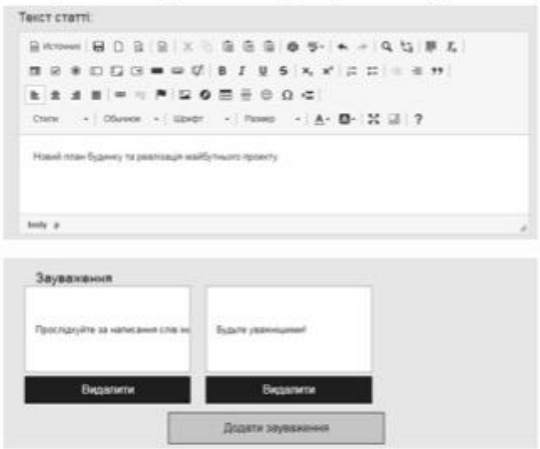
Приклади роботи

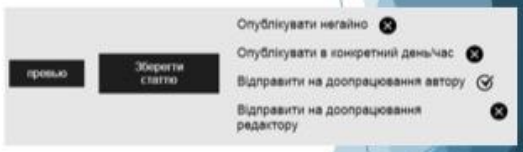
Головний редактор/редактор бачить цю статтю у відповідному блоці на сторінці



Приклади роботи

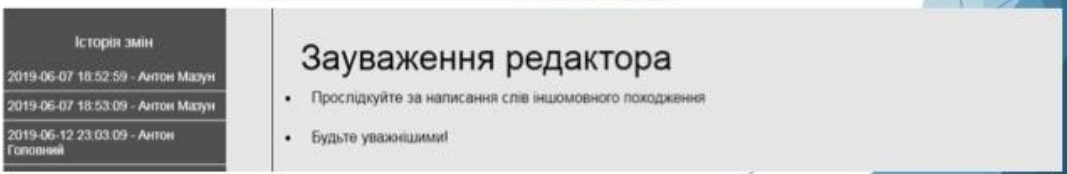
Редактор може внести правки до тексту статті, залишити зауваження до автора та має один на вибір з пунктів подальшого розвитку подій.





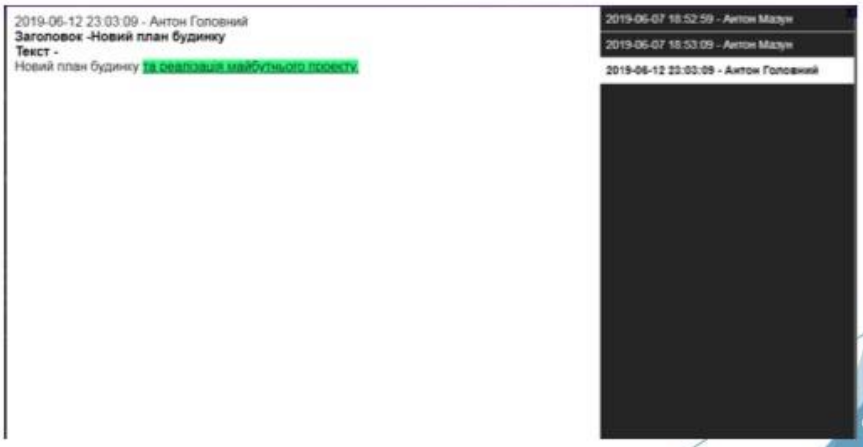
Приклади роботи

Автору приходить стаття з відповідним статусом «Вам відправили на доопрацювання» із зауваженнями та відповідними корегуванням в тексті.



Приклади роботи

Натиснувши на відповідний елемент історії буде показано вікно з усіма змінами , що вносились до тексту статті.



Висновки

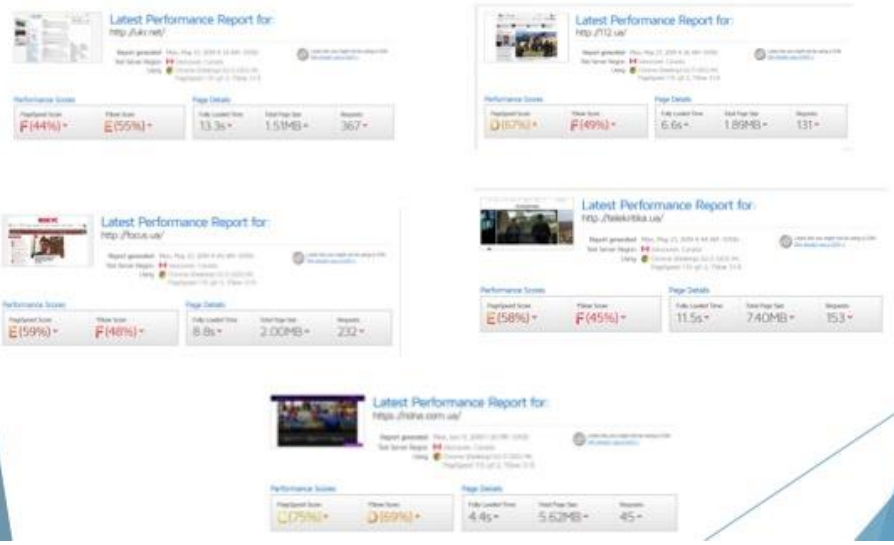
- проведено аналіз доступних програмних рішень;
- розроблено структуру веб-додатку та архітектуру БД;
- реалізовано серверну та клієнтську частини;
- реалізовано вимоги до ПЗ;
- розроблено особистий кабінет для персоналу новинного порталу;
- розроблено модуль «історії змін»

На відміну від існуючих аналогів, розроблювана система задовольняє потреби редакторського та журналістського складів видавництв через реалізацію автоматизованого менеджменту, а використання моделі SaaS забезпечує продукту більшу гнучкість та можливість розширення.

Рекомендації щодо подальшого вдосконалення

- Створення ще одного типу користувача - «Юрист»
- Створення чату всередині системи задля швидкого та зручного спілкування
- Реплікація бази даних

Порівняння швидкодії порталу з аналогами



Впровадження

Даний проект розміщений за адресою - https://ridna.com.ua/

Дякую за увагу!

Додаток 3

Лістинг програми

Модуль models.py (компонент article)

```
from django.db import models
# from categories.models import Categories
from categories.models import Categories
from django.contrib.auth.models import User
from django.contrib.auth import get_user_model
from ckeditor.fields import RichTextField
from ckeditor_uploader.fields import RichTextUploadingField
from manage_user.models import EditorUser, AdminMain
from django.conf import settings
import django
import datetime
from django.utils import timezone

# from elasticsearch_dsl import (
#     DocType,
#     Date,
#     Keyword,
#     Text,
#     Boolean,
#     Integer
# )

# Create your models here.

STATUS_SITE = (
    ('safe', 'надійне джерело'),
    ('unsafe', 'ненадійне джерело')
)
STATUS_ARTICLE = (
    ('inactive', 'Не опрацьована'),
    ('edited', 'На опрацюванні'),
    ('to_main_admin', 'У головного редактора'),
    ('back_to_user', 'Повернено автору'),
    ('done', 'В публікацію')
)


# def get_sentinel_user():
#     return get_user_model().objects.get_or_create(username='deleted')[0]


class Photo(models.Model):
    title = models.CharField(max_length=255, blank=True)
    file = models.ImageField(upload_to='photos_uploads/')
    uploaded_at = models.DateTimeField(auto_now_add=True)

    def delete(self, *args, **kwargs):
        # You have to prepare what you need before delete the model
        storage, path = self.file.storage, self.file.path
        # Delete the model before the file
        super(Photo, self).delete(*args, **kwargs)
        # Delete the file after the model
        storage.delete(path)

    def save(self, *args, **kwargs):
        print('save photo!')
        super(Photo, self).save(*args, **kwargs)


class PhotoSocial(models.Model):
    title = models.CharField(max_length=255, blank=True)
    file = models.ImageField(upload_to='photos_social_uploads/')
```

```
    uploaded_at = models.DateTimeField(auto_now_add=True)

    def delete(self, *args, **kwargs):
        # You have to prepare what you need before delete the model
        storage, path = self.file.storage, self.file.path
        # Delete the model before the file
        super(PhotoSocial, self).delete(*args, **kwargs)
        # Delete the file after the model
        storage.delete(path)

    def save(self , *args, **kwargs):
        print('save photo!')
        super(PhotoSocial, self).save(*args, **kwargs)


class Vynoska(models.Model):
    text = models.TextField(max_length=500, verbose_name='Текст Виноски',
blank=True, null=True)
    entry = models.CharField(max_length=255, verbose_name='Джерело',
blank=True, null=True)


class Remark(models.Model):
    text = models.TextField(max_length=500, verbose_name='Текст
зауваження', blank=True, null=True)


from mptt.models import MPTTModel, TreeForeignKey


class Comment(MPTTModel):
    """Модель записи блога"""
    user = models.ForeignKey(
        User,
        verbose_name="Пользователь",
        on_delete=models.CASCADE,
        related_name='twits')
    text = models.TextField("Сообщение", max_length=500)
    date = models.DateTimeField("Дата", auto_now_add=True)
    parent = TreeForeignKey(
        "self",
        verbose_name="Твит",
        on_delete=models.CASCADE,
        blank=True,
        null=True,
        related_name="child")
    like = models.IntegerField(default=0)
    user_like = models.ManyToManyField(User, verbose_name="Кто лайкнул",
related_name="users_like")

    def __str__(self):
        return "{} - {}".format(self.id, self.user)

    class MPTTMeta:
        verbose_name = "Сообщение"
        verbose_name_plural = "Сообщения"

    def get_descendant_count(self):
        """
        Returns the number of descendants this model instance has.
        """
        if self._mpttfield('right') is None:
            # node not saved yet
            return 0
```

```
        else:
            return (self._mpttfield('right') - self._mpttfield('left') - 1)
// 2

from datetime import datetime
class Article(models.Model):
    STATUS_SITE = (
        ('safe', 'надійне джерело'),
        ('unsafe', 'ненадійне джерело')
    )
    category = models.ForeignKey(Categories, on_delete=models.CASCADE,
related_name='category')
    user = models.ForeignKey(EditorUser, related_name='articles',
verbose_name='User', on_delete=models.SET_DEFAULT,
                             blank=True,
                             null=True, default=None)
    admin_user = models.ForeignKey(AdminMain, verbose_name='Admin main',
related_name='articles',
                                   on_delete=models.SET_DEFAULT,
blank=True,
                                   null=True, default=None)
    redactor_user = models.ForeignKey(EditorUser, verbose_name="Редактор
статті",
                                      default=None,
                                      on_delete=models.SET_DEFAULT,
                                      blank=True, null=True)
    redactor_user_admin = models.ForeignKey(EditorUser,
verbose_name="Редактор статті Admin main",

related_name='redactor_user_admin',
                                            default=None,
                                            on_delete=models.SET_DEFAULT,
                                            blank=True, null=True)
    title = models.CharField(max_length=255, verbose_name='Заголовок
статті')
    sub_title = models.CharField(max_length=255, verbose_name='Підаголовок
статті', blank=True, null=True)
    tags = models.TextField(max_length=1000, verbose_name='Теги')
    # main_image = models.ImageField(upload_to='uploads/articles',
verbose_name='Головне зображення статті', blank=True,
    #                                null=True)
    main_image = models.ForeignKey(Photo, default=None,
                                   on_delete=models.SET_DEFAULT,
                                   verbose_name='Головне зображення
статті',
                                   blank=True, null=True)
    social_image = models.ForeignKey(PhotoSocial, default=None,
                                   on_delete=models.SET_DEFAULT,
                                   verbose_name='Зображення для репосту
статті',
                                   blank=True, null=True)
    comment_for_image = models.CharField(max_length=255,
verbose_name='Коментар під фото')
    entry_img = models.CharField(max_length=255, verbose_name='Джерело')
    content = RichTextUploadingField(verbose_name='Текст статті')
    lead = models.TextField(max_length=2000, verbose_name='Лід(вріз)')
    link_to_site = models.CharField(max_length=1000, verbose_name='Поилання
на джерело')
    name_site = models.CharField(max_length=1000, verbose_name='Назва
джерела', default='Рідна Україна')
    status_site = models.CharField(max_length=50, choices=STATUS_SITE,
verbose_name='Статус джерела')
    text_under_post = models.TextField(max_length=255, verbose_name='Текст
під статтею (авторсво, кредити тощо)')
```

```
    create_at = models.DateTimeField(default=django.utils.timezone.now)
    update_at = models.DateTimeField(auto_now_add=False,
                                     auto_now=False,
                                     # auto_created=True,
                                     default=datetime.now,
                                     blank=True,
                                     null=True)
    publish_at = models.DateTimeField(default=datetime.now,
auto_now_add=False, blank=True,
                                      null=True)
    like = models.IntegerField(default=0, verbose_name='Вподобань')
    repost = models.IntegerField(default=0, verbose_name='Репости')
    review_count = models.IntegerField(default=0,
verbose_name='Переглядів')
    status = models.CharField(max_length=50, choices=STATUS_ARTICLE,
verbose_name='Статус',
                              default=STATUS_ARTICLE[0][0])

    is_blog = models.BooleanField(default=0)
    is_article = models.BooleanField(default=0)
    articles = models.ManyToManyField('self', related_name='articles')
    vynoska = models.ManyToManyField(Vynoska, blank=True, null=True)
    remark = models.ManyToManyField(Remark, blank=True, null=True)
    update_by = models.ForeignKey(EditorUser, related_name='update_by',
on_delete=models.SET_DEFAULT,
                                  blank=True, null=True,
                                  default=None
                                  )
    to_editors = models.BooleanField(default=False,
verbose_name='Відправити до редакторів')

    show_image = models.BooleanField(default=True, verbose_name='Показувати
в слайдері')
    comments = models.ManyToManyField(Comment)

    def __str__(self):
        return 'Стаття {}'.format(self.title)
```

Модуль urls.py (компонент article)

```
from django.urls import path , include ,re_path
from rest_framework import routers
from . import views

# urlpatterns = [
#     path('get_all/<int:id>/' , views.get_all)
# ]


router = routers.DefaultRouter()
app_name = 'articles'
urlpatterns = [
    re_path(r'', include(router.urls)),
    path('user_articles/<int:current>/<int:offset>',
views.ArticleByuser.as_view()),
    path('all_articles' , views.AllArticles.as_view()),
    path('all_articles_for_statistic/<int:current>/<int:offset>',
views.AllArticlesForStatistic.as_view()),
    path('on-edit-articles/<int:current>/<int:offset>' ,
views.OnEditUserArticles.as_view()),
    path('length-articles/' , views.LengthArticlesByUser.as_view()),
```

```
    path('unset-artilces/<int:pk>' , views.unset_article , name='unset-
artilces'),
    path('all-images/<int:current>/<int:offset>' ,
views.AllImages.as_view()),
    # path('all-media/' , views.all_media),
    path('delete-image/<int:id>' , views.delete_image),
    path('preview_article' , views.preview_article),
    path('on-edit-articles' , views.EditedArticle.as_view()),
    path('edited-articles' , views.ArticlesToAdmin.as_view()),
    path('set-count-type' , views.set_count),
    path('read-more-articles' , views.ReadMoreArticles.as_view()),
    path('read-more-for-article/<int:id_article>' ,
views.ReadMoreForArticle.as_view())
]
```

Модуль views.py (компонент article)

```
from django.shortcuts import render
from django.http import HttpResponse, JsonResponse
from manage_user.models import EditorUser, AdminMain
from django.contrib.auth.models import User
from django.core import serializers
from .models import Article, STATUS_ARTICLE
from rest_framework import viewsets, generics
from .serializers import ArticleSerializer, PhotoSerializer
from categories.models import Categories
from editor.forms import ArticleForm

from .watermark import set_watermark


# Create your views here.


class ArticleByuser(generics.ListAPIView):
    serializer_class = ArticleSerializer

    def get_queryset(self):
        current = self.kwargs['current']
        offset = self.kwargs['offset']
        try:
            editor_user = EditorUser.objects.get(user=self.request.user.id)
            articles = Article.objects.filter(user=editor_user).order_by('-
create_at')[current:offset]
            print('ArticleByuser', len(articles))
            return articles
        except Exception as e:
            editor_user = AdminMain.objects.get(user=self.request.user.id)
            articles =
Article.objects.filter(admin_user=editor_user).order_by('-
create_at')[current:offset]
            return articles

from django.db.models import Q
class AllArticles(generics.ListAPIView):
    serializer_class = ArticleSerializer

    def get_queryset(self):
        try:
            editor_user = EditorUser.objects.get(user=self.request.user.id)
            if editor_user.type_user == 'main-editor':
```

```
                articles = Article.objects\
                    .filter((Q(redactor_user=editor_user) &
Q(status='to_editors'))
                            | (Q(redactor_user__isnull=True) &
Q(status='to_editors'))
                            | (Q(redactor_user_admin=editor_user) &
Q(status='to_main_admin'))
                            | (Q(redactor_user_admin__isnull=True) &
Q(status='to_main_admin'))) \

.exclude(user=editor_user).filter(Q(status='to_editors') |
Q(status='to_main_admin')).order_by(
                    '-create_at').filter(
                    to_editors=True)
            else:
                articles =
Article.objects.filter(Q(redactor_user=editor_user) |
Q(redactor_user__isnull=True))\

.exclude(user=editor_user).filter(status='to_editors').order_by(
                    '-create_at').filter(
                    to_editors=True)
        except Exception as e:
            editor_user = EditorUser.objects.get(user=self.request.user.id)
            articles =
Article.objects.all().exclude(redactor_user_admin=editor_user).order_by('-
create_at').filter(
                to_editors=True)
        return articles


class AllArticlesForStatistic(generics.ListAPIView):
    serializer_class = ArticleSerializer

    def get_queryset(self):
        try:
            editor_user = EditorUser.objects.get(user=self.request.user.id)
            if editor_user.type_user == 'main-editor':
                print('SUPEREDITOR')
                articles = Article.objects.order_by(
                    '-create_at').filter(
                    to_editors=True)
            else:
                print('EDITOR')
                articles = Article.objects \
                    .filter((Q(redactor_user=editor_user))
                            | (Q(redactor_user_admin=editor_user))) \
                    .order_by('-create_at').filter(
                    to_editors=True)
        except Exception as e:
            editor_user = EditorUser.objects.get(user=self.request.user.id)
            articles =
Article.objects.all().exclude(redactor_user_admin=editor_user).order_by('-
create_at').filter(
                to_editors=True)
        return articles


class OnEditUserArticles(generics.ListAPIView):
    serializer_class = ArticleSerializer

    def get_queryset(self):
        current = self.kwargs['current']
        offset = self.kwargs['offset']
```

```
        try:
            editor_user = EditorUser.objects.get(user=self.request.user.id)
            if editor_user.type_user == 'main-editor':
                print(' main-editor main-editor main-editor main-editor')
                articles =
Article.objects.filter(Q(redactor_user=editor_user) |
Q(redactor_user_admin=editor_user)) \
                    .order_by('-create_at').filter(
                    to_editors=True)[current:offset]
                return articles
            else:
                articles =
Article.objects.filter(redactor_user=editor_user).order_by('-
create_at').filter(
                    to_editors=True)[current:offset]
                return articles
        except Exception as e:
            editor_user = AdminMain.objects.get(user=self.request.user.id)
            articles =
Article.objects.filter(redactor_user_admin=editor_user).order_by('-
create_at').filter(
                to_editors=True)[current:offset]
            return articles


class LengthArticlesByUser(generics.ListAPIView):
    serializer_class = ArticleSerializer

    def get_queryset(self):
        try:
            editor_user = EditorUser.objects.get(user=self.request.user.id)
            articles = Article.objects.filter(redactor_user=editor_user)
            return articles
        except Exception as e:
            editor_user = AdminMain.objects.get(user=self.request.user.id)
            articles =
Article.objects.filter(redactor_user_admin=editor_user)
            return articles


class AllImages(generics.ListAPIView):
    serializer_class = PhotoSerializer

    def get_queryset(self):
        current = self.kwargs['current']
        offset = self.kwargs['offset']
        all_photos = Photo.objects.all().order_by('-
uploaded_at')[current:offset]
        return all_photos


from django.db.models import Q


class EditedArticle(generics.ListAPIView):
    serializer_class = ArticleSerializer

    def get_queryset(self):
        query = Q(status='edited')
        query.add(Q(status='back_to_user'), Q.OR)
        articles = Article.objects.filter(query)
        return articles
```

```
class ArticlesToAdmin(generics.ListAPIView):
    serializer_class = ArticleSerializer

    def get_queryset(self):
        return Article.objects.filter(status='to_main_admin').order_by('-
create_at')


class ReadMoreArticles(generics.ListAPIView):
    serializer_class = ArticleSerializer

    def get_queryset(self):
        return Article.objects.filter(status='done').order_by('-create_at')

class ReadMoreForArticle(generics.ListAPIView):
    serializer_class = ArticleSerializer

    def get_queryset(self):
        id_article  = int(self.kwargs['id_article'])
        print()
        return Article.objects.get(pk=id_article).articles.all()

def delete_image(request, id):
    Photo.objects.get(pk=id).delete()
    return HttpResponse('ok!')


def unset_article(request, pk=None):
    article = Article.objects.get(id=pk)
    article.redactor_user = None
    article.redactor_user_admin = None
    article.status = 'inactive'
    article.save()
    if request.is_ajax():
        return JsonResponse({
            'status': 'ok'
        })
    else:
        ctx = {}
        form = ArticleForm(instance=article)
        ctx['article_obj'] = article
        ctx['form'] = form
        return render(request, 'custom_admin/editor/show_articles.html',
ctx)


import json
from .models import Photo
from django.core.files.base import ContentFile
import base64


def upload_photo(request):
    if request.is_ajax() and request.method == 'POST':
        images = json.loads(request.POST.get('image'))
        for img in images:
            base_64 = img['data_src']
            format, imgstr = base_64.split(';base64,')
            ext = format.split('/')[-1]
            base_64 = ContentFile(base64.b64decode(imgstr), name='temp.' +
ext)
            instance = Photo(
```

```
                title='test',
                file=base_64
            )
            set_watermark(instance)
    return JsonResponse({
        'upload_photo': 'ok'
    })


def preview_article(request):
    if request.is_ajax():
        article_for_preview = {}
        if request.method == 'POST':
            article_for_preview['category'] = request.POST.get('category')
            article_for_preview['title'] = request.POST.get('title')
            article_for_preview['sub_title'] =
request.POST.get('sub_title')
            article_for_preview['comment_photo'] =
request.POST.get('comment_photo')
            article_for_preview['entry_img'] =
request.POST.get('entry_img')
            article_for_preview['content'] = request.POST.get('content')
            article_for_preview['lead'] = request.POST.get('lead')
            article_for_preview['link_to_site'] =
request.POST.get('link_to_site')
            article_for_preview['name_site'] =
request.POST.get('name_site')
            article_for_preview['status_site'] =
request.POST.get('status_site')
            article_for_preview['text_under_post'] =
request.POST.get('text_under_post')
            article_for_preview['image'] = request.POST.get('image')
            article_for_preview['user'] =
EditorUser.objects.get(user=request.user)
            article_for_preview['vynoska'] = {}
            for key, value in
json.loads(request.POST.getlist('vynoska[]')[0]).items():
                article_for_preview['vynoska'][key] = value
    return render(request, 'custom_admin/partials/preview_article.html', {
        'article_obj': article_for_preview
    })


def set_count(request):
    if request.is_ajax():
        if request.method == 'POST':
            article_id = request.POST.get('article_id')
            field_db = request.POST.get('field_db')
            value = int(request.POST.get('value'))
            article_obj = Article.objects.get(id=article_id)
            if field_db:
                if field_db == 'like':
                    article_obj.like = value
                elif field_db == 'review_count':
                    article_obj.review_count = value
                elif field_db == 'repost':
                    article_obj.repost = value
                article_obj.save()
    return HttpResponse('ok')
```

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2018 р.

НОВИННИЙ ПОРТАЛ ІЗ СИСТЕМОЮ УПРАВЛІННЯ ПУБЛІКАЦІЄЮ КОНТЕНТА НА ОСНОВІ МОДЕЛІ SAAS

Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проекту:

_____ Р.А. Гадиняк

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ А.І. Мазун

2018

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Новинний портал із системою управління публікацією контента на основі моделі SaaS, який являє собою веб-застосунок, створений на платформі Python Django.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1) функціональна працездатність елементів сторінок web-ресурсу;
2) наявність доступу до бази даних;
3) взаємодію сервера з клієнтською частиною;
4) забезпечення коректної обробки запитів від користувача;
5) забезпечення належного рівня безпеки даних;
6) зручність роботи з веб-додатком;
7) відповідність дизайну вимогам Технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

1) функціональне тестування, зокрема на рівні Critical path test (базове тестування);
2) тестування продуктивності програмного забезпечення, зокрема Stability testing (тестування стабільності) та Load testing (навантажувальне тестування);
3) тестування інтерфейсу.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Працездатність web-ресурсу перевіряється шляхом:

1) динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати;
2) динамічного ручного тестування на відповідність функціональним вимогам;
3) статичного тестування коду;
4) тестування web-ресурсу в різних web-браузерах;
5) тестування при максимальному навантаженні;
6) тестування стабільності роботи при різних умовах;
7) тестування зручності використання;
8) тестування інтерфейсу.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2019 р.

НОВИННИЙ ПОРТАЛ ІЗ СИСТЕМОЮ УПРАВЛІННЯ ПУБЛІКАЦІЄЮ КОНТЕНТА НА ОСНОВІ МОДЕЛІ SAAS

Керівництво користувача

ДП.045440-05-34

"ПОГОДЖЕНО"

Керівник проекту:

_____ Р.А. Гадиняк

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ А.І. Мазун

2019

ЗМІСТ

1. Опис структури web-ресурсу

Новинний портал із системою управління публікацією контента на основі моделі SaaS складається із web-сторінок, вміст яких формується динамічно. Web-ресурс є одномовним з українською мовою.

Динамічна частина web-ресурсу включає наступні web-сторінки:

- сторінка перегляду особистої інформації користувача;
- сторінка створення та редагування публікації;
- сторінка перегляду статистики публікацій;
- сторінка керування медіафайлами;
- сторінка менеджменту користувачів;
- сторінка керування пріоритетами виводу публікацій блогерів;

Кожна web-сторінка містить інформацію про користувача, що ввійшов в систему, його логін та кнопку виходу із системи, та меню.

Web-ресурс також має клієнтську частину для перегляду публікацій читачем.

2. Сторінка менеджменту користувачів

Для того, щоб користувач мав змогу зайти в систему, головний редактор має визначити його роль в системі (рис. 1). У кабінеті головного редактора у відповідному пункті меню «Ролі користувачів» може вибрати користувача та назначити відповідну роль, крім цього, в цьому пункті меню головний редактор може бачити всіх активних та неактивних (ті, що не підтвердили e-mail та/або заблокованих користувачів) (рис. 2).

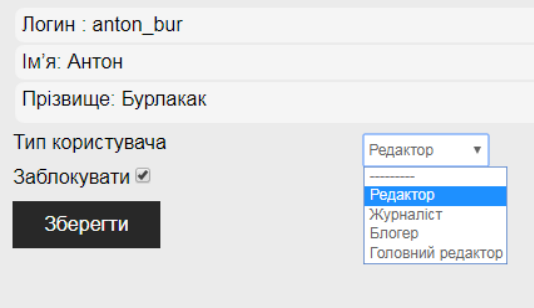


Рис 1. Список вибору ролі користувача для головного редактора

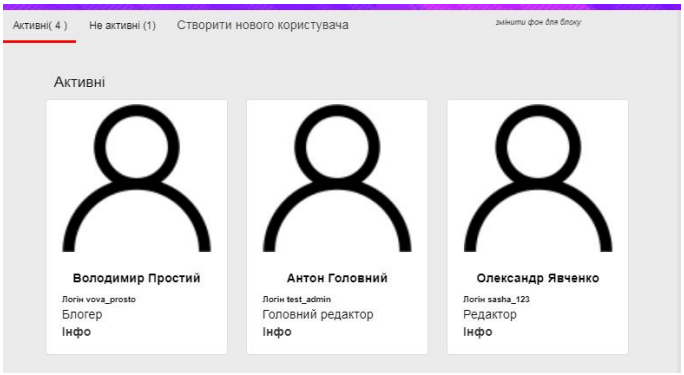


Рис. 2. Список всіх активних користувачів для головного редактора

3. Процедура авторизації користувача

Після того, як користувач має статус в системі, він може здійснити вхід та потрапити до свого робочого простору. Для цього він має заповнити просто форму входу – логін та пароль (рис. 3).

Рис. 3. Форма авторизації користувача

4. Робота з особистою інформацією користувача

На сторінці «Ваш профіль» користувач може бачити свою особисту інформацію та загальну статистику для всіх публікацій, що були опубліковані (рис. 4).

Рис. 4. Загальна статистика та особиста інформація користувача

Змінити особисту інформацію можна натиснувши на відповідну іконку «редагування особистої інформації». Можна заповнити посилання на соціальні мережі, змінити фото та вибрати тип представлення на сайті: публікуватися під ФІО або під псевдонімом) (рис. 5).

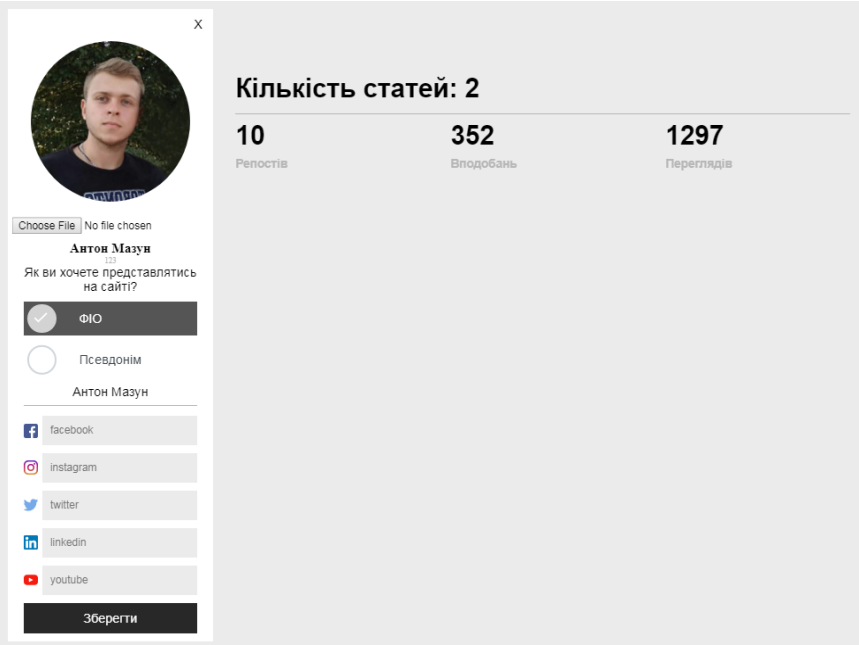


Рис. 5. Зміна особистої інформації користувача

Також на цій сторінці можна побачити статус своїх публікацій та скористуватися пошуком по трьом полям: заголовок статті, категорія та дата створення (рис. 6).

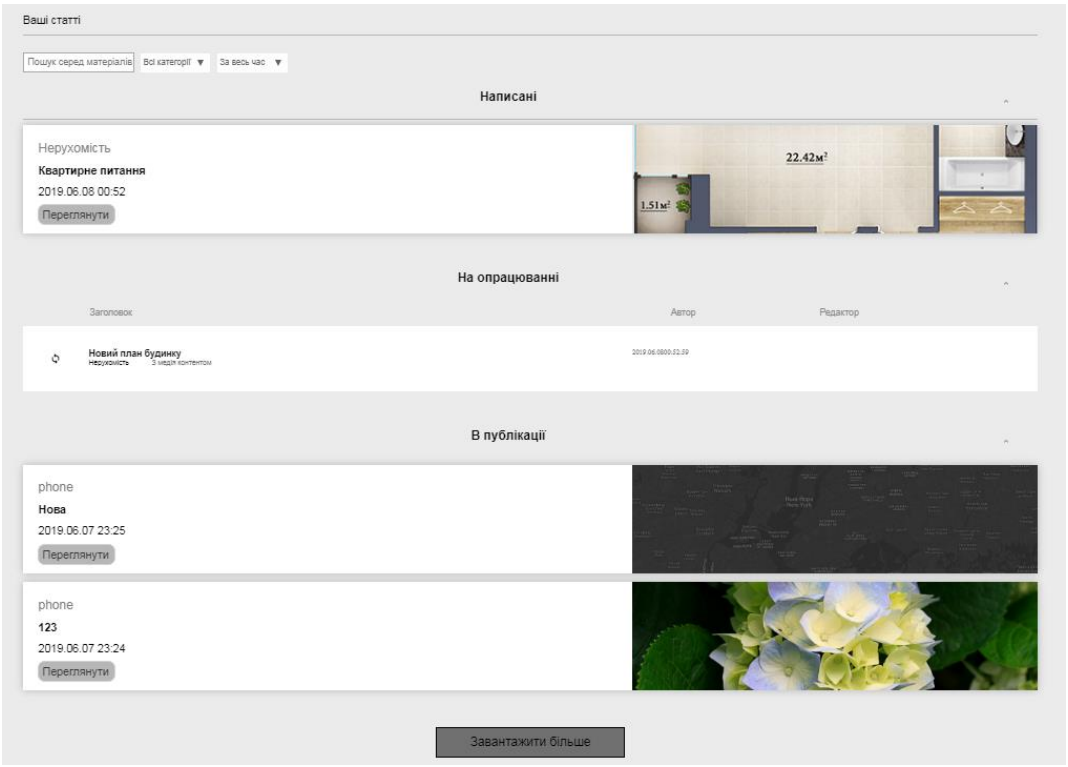


Рис. 6. Список публікацій користувача

5. Сторінка створення та редагування публікації

На сторінці «написати публікацію» користувач потрапляє на форму написання статті (рис. 7 – 8), в якій він обирає одну з запропонованих категорій, пише заголовок статті, невеликий опис (лід), мета-теги – ключові слова, обирає фото для головного зображення статті, фото для репосту в соц. мережі, коментар під фото, джерело, текст публікації, посилання на джерело (у випадку, якщо стаття не є авторською), назва джерела, статус джерела (надійне, ненадійне), може додати виноски (невеликі пояснення деяких слів у тексті), текст під статтею.

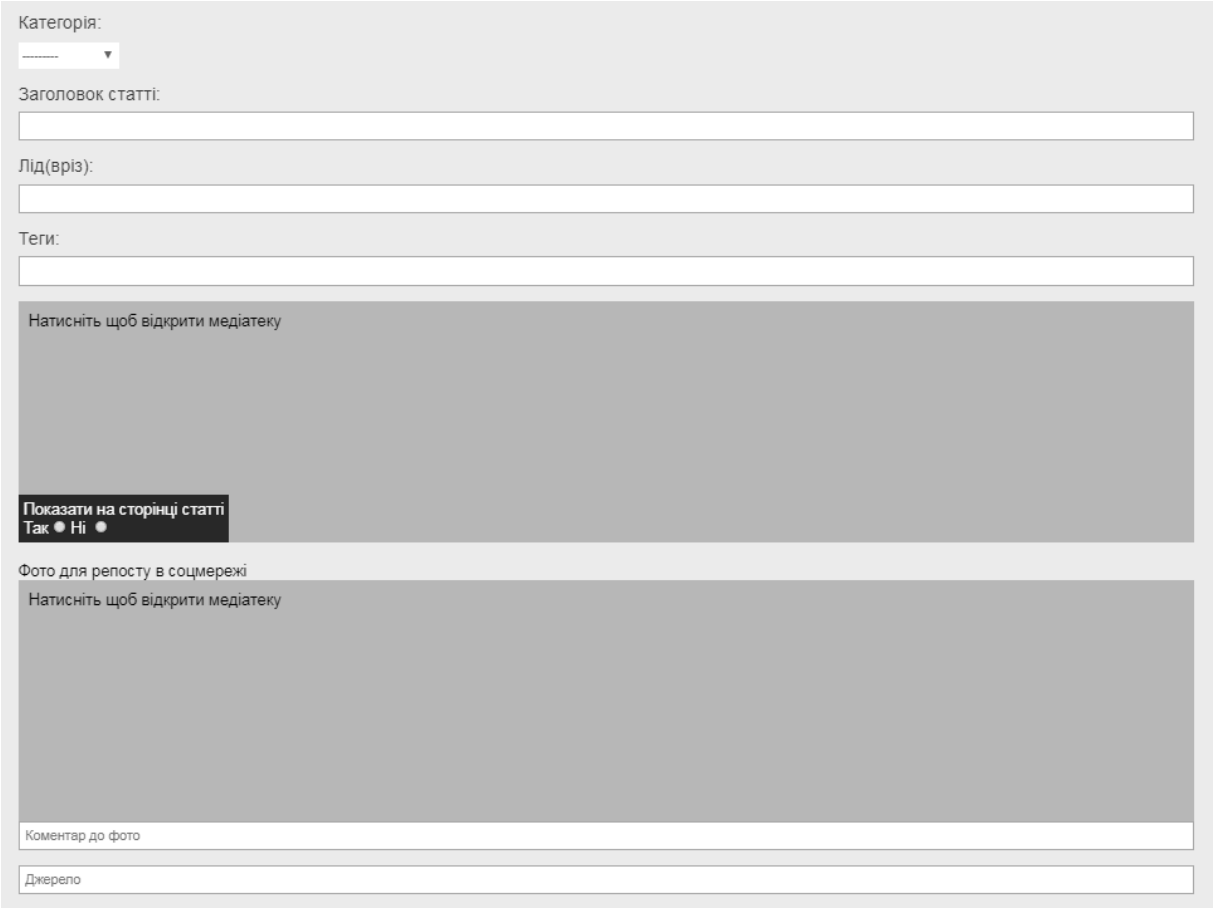


Рис. 7. Форма статті. Початок

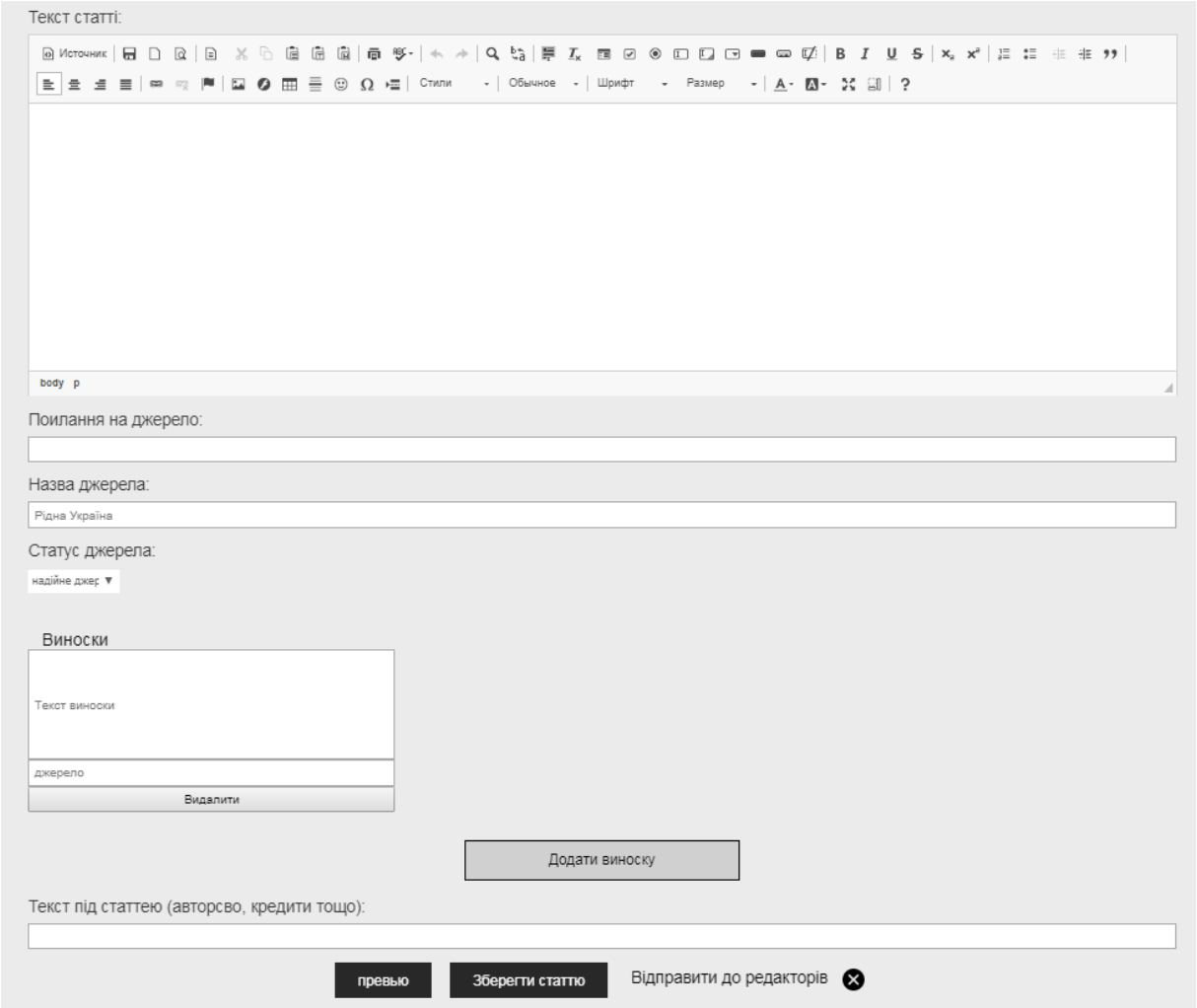


Рис. 8. Форма статті. Кінець

Також на цій сторінці розміщені елементи управління:

- прев'ю (попередній перегляд);
- зберегти публікацію;
- можливість відправити до редакторів.

Функцією попереднього перегляду передбачене збирання даних, що були введені користувачем та відкриває нове вікно браузера в якому відображається публікація так, як вона буде показана читачеві.

Функцією збереження статті передбачено перевірку чи автор захотів відправити статтю до редакторів або в публікацію чи ні. У випадку якщо автор не захотів відправляти статтю далі, то стаття зберігається зі статусом «inactive», таким чином ніхто її не бачить, окрім автора в своєму кабінеті. Якщо автор позначив, що хоче відправити статтю до редакторів стаття

зберігається з відповідним статусом – «edited» і буде доступна всім редакторам системи для взяття на опрацювання та не буде доступною для редагування самим автором. Якщо редактор відправляє на доопрацювання автору, то стаття буде збережена з відповідним статусом - «back_to_user», якщо головний редактор або автор, що наділений можливістю публікації без попередньої модерації захоче відправити статтю в публікацію, то стаття буде збережена зі статусом – «done».

Рис. 9. Попередній перегляд статті.

Для історії змін був виділений окремий блок у вигляді списку змін у клієнтській частині додатку (рис. 10), всі елементи блоку є клікабельні – відкривають діалогове вікно зі списком всіх змін статті (рис. 11).

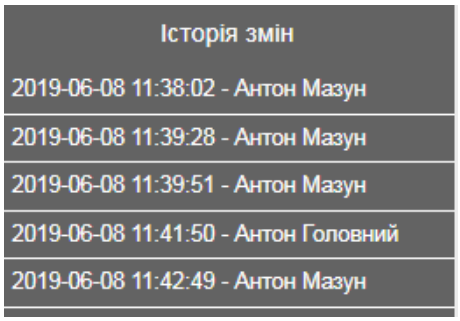


Рис. 10. Список змін статті

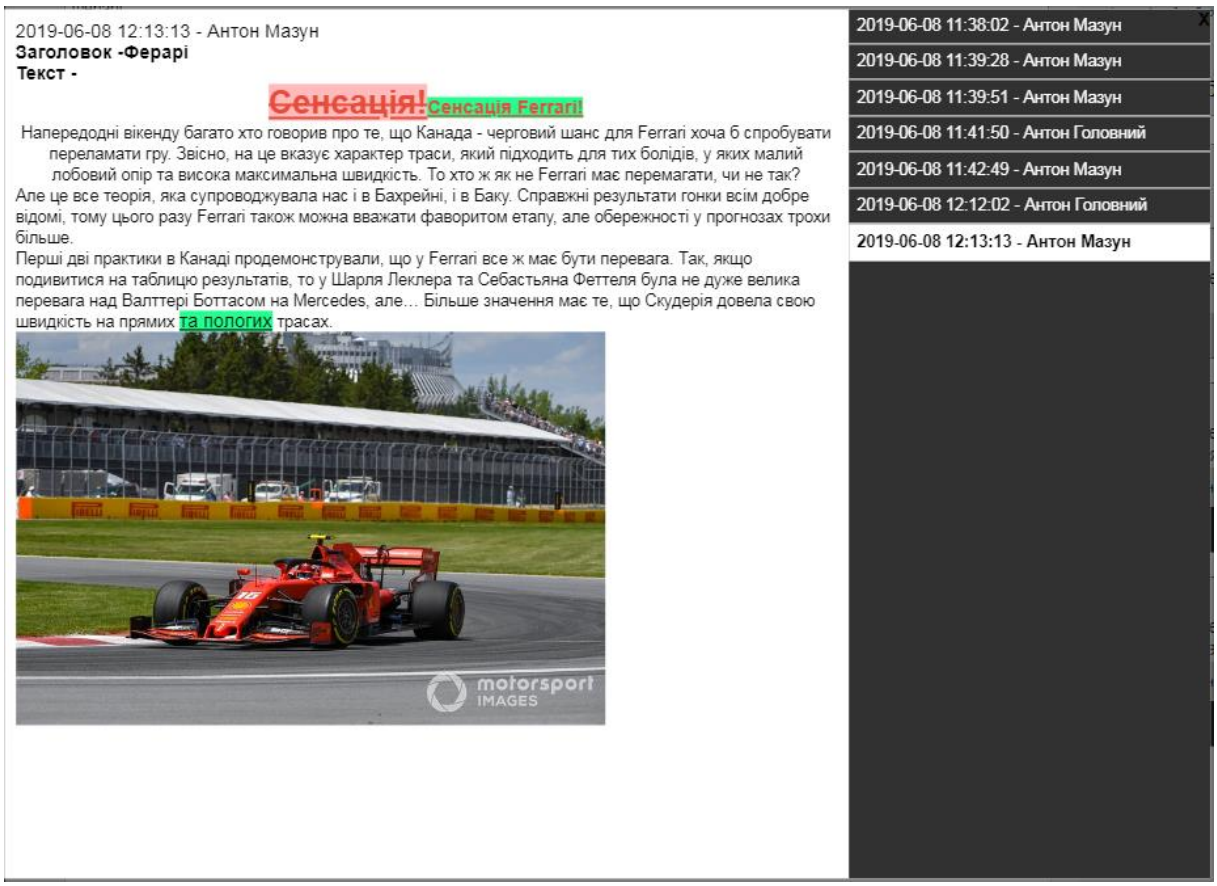


Рис. 11. Діалогове вікно зі списком змін публікації

Для відображення різниці в тексті було обрані такі інтуїтивно зрозуміли позначення – для того тексту, що був видалений показувати його на червоному тлі з закресленим текстом, для тексту, що був доданий – на зеленому фоні (рис. 11).

6. Сторінка керування пріоритетами виводу публікацій блогерів

Однією з функцій головного редактора є зміна пріоритету виводу блогів.

Для більш зрозумілого використання цього модулю на клієнтській частині було використано drag-n-drop підхід, який відповідає за переміщення блоків один відносно одного і таким чином формує пріоритет виводу публікацій тих чи інших блогерів (рис. 12).

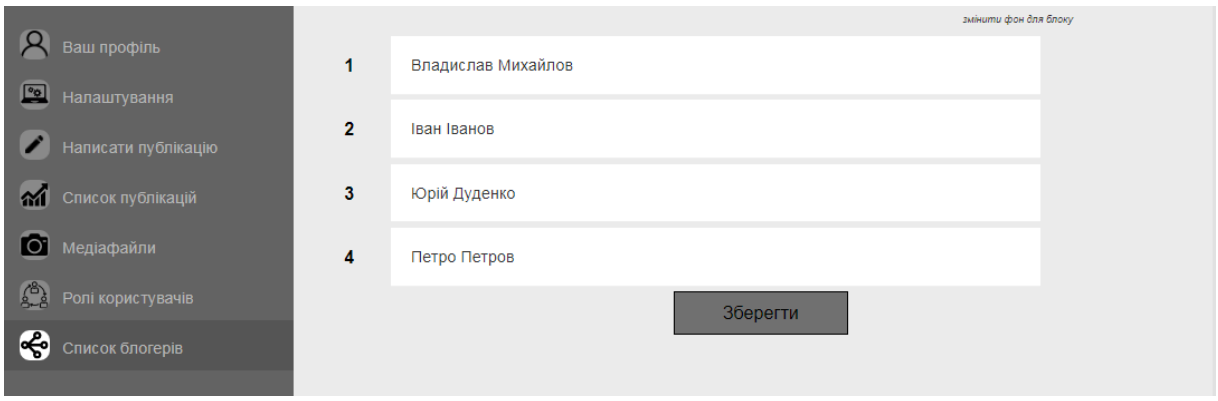


Рис. 12. Зміна порядку виводу публікацій блогерів