МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки та програмної інженерії
Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

Катерина Нестеренко
“____”_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
МАГІСТРА

Тема: “ Інформаційна система опублікування та знаходження інформації для студентів”

Виконавець: Копачевський Максим Олександрович

Керівник: к.т.н доцент Задонцев Юрій Вікторович < *підпис* >

Нормоконтролер: к.т.н доцент Гололобов Дмитро Олександрович

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки та програмної інженерії
Кафедра інженерії програмного забезпечення
Освітній ступінь магістр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Програмне забезпечення систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Катерина Нестеренко
"___" _____ 2023 р

ЗАВДАННЯ
на виконання кваліфікаційної роботи студента
Копачевського Максима Олександровича

1. Тема кваліфікаційної роботи: «Інформаційна система опублікування та знаходження інформації для студентів»
затверджена наказом ректора від 29.09.2023 р. № 1994/ст.

2. Термін виконання проекту: з 02.10.2022 р. по 31.12.2023 р.

3. Вихідні дані до роботи : програмний продукт розробити у вигляді веб-додатку для опублікування та знаходження інформації студентами.

4. Зміст пояснювальної записки:
    1. Аналіз проблеми знаходження і опублікування інформації студентами та порівняння існуючих рішень.
    2. Вимоги до застосунку для опублікування та знаходження інформації для студентів.
    3. Структура програмного застосунку для пошуку та опублікування інформації студентами.
    4. Прототип програмного застосунку для пошуку та опублікування інформації студентами.

5. Перелік обов'язкових слайдів презентації:
    1. Аналіз проблеми та аналогів додатку
    2. Вимоги до додатку
    3. Структура додатку
    4. Реалізація додатку

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1.	Розробка та затвердження графіка роботи.	02.10.2023-03.10.2023	
2.	Підготовка та написання 1 розділу «Аналіз проблеми знаходження і опублікування інформації студентами та порівняння існуючих рішень». Відсилка керівнику.	04.10.2023-20.10.2023	
3.	Підготовка та написання 2 розділу «Вимоги до застосунку для опублікування та знаходження інформації для студентів». Відсилка керівнику.	23.10.2023-08.11.2023	
4.	Підготовка та написання 3 розділу «Структура програмного застосунку для пошуку та опублікування інформації студентами». Відсилка керівнику.	09.11.2023-24.11.2023	
5.	Підготовка та написання 4 розділу «Прототип програмного застосунку для пошуку та опублікування інформації студентами». Відсилка керівнику.	27.11.2023-10.12.2023	
6.	Редагування та друк пояснювальної записки, графічного матеріалу Відсилка ПЗ для перевірки на плагіат одним файлом.	11.12.2023-15.12.2023.	
7.	Проходження нормо-контролю, перепліт пояснювальної записки. Отримання відгуку керівника. Підготовка презентації та тексту доповіді.	15.12.2023-16.12.2023	
8.	Передзахист кваліфікаційної роботи. Отримання рецензії.	16.12.2023.-17.12.2023	
9.	Здати секретарю ДЕК: ПЗ, ГМ, CD-R з електронними версіями ПЗ, ГМ, презентацію, відгук керівника,	18.12.2023-24.12.2023	

	рецензію, довідку про успішність, 2 папки, 2 конверта)		
10.	Захист дипломної роботи перед ЕК	25.12.2023-31.12.2023	

Дата видачі завдання 02.10.2023 р.
Керівник дипломної роботи: _____к.т.н доцент Юрій ЗАДОНЦЕВ

Завдання прийняв до виконання: Максим КОПАЧЕВСЬКИЙ

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Інформаційна система опублікування та знаходження інформації для студентів»: 74 сторінки, 41 рисунок, 3 таблиці, 5 використаних джерел, 0 додатків.

ІНФОРМАЦІЙНА СИСТЕМА ОПУБЛІКУВАННЯ ТА ЗНАХОДЖЕННЯ ІНФОРМАЦІЇ ДЛЯ СТУДЕНТІВ, ПРОГРАМНИЙ ЗАСТОСУНОК, АНАЛІЗ, ПРОТОТИП.

Об’єкт дослідження – застосунок для студентів що дозволяє знаходити та опубліковувати інформацію.

Мета дипломної роботи – створення застосунку для покращення якості знаходження інформації студентам та організувати платформу для обміну думками.

ABSTRACT

Explanatory note to the diploma work "Information system of publishing and finding information for students": 74 pages, 41 pictures, 3 tables, 6 sources, 0 appendices.

INFORMATION SYSTEM OF PUBLISHING AND FINDING INFORMATION FOR STUDENTS, SOFTWARE APPLICATION, ANALYSIS, PROTOTYPE.

Property development – application for students that allows them to find and publish information.

Purpose – creation of an application to improve the quality of finding information for students and organize a platform for the exchange of opinions.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

SQL – Structured query language (мова інтегрованих запитів)

UML – Unified Modeling Language (уніфікована мова моделювання)

IDE – Integrated development environment (Інтегроване середовище розробки)

ПЗ – Програмне забезпечення

ООП – Об'єктноорієнтоване програмування

MVC – Шаблон проектування Model-View-Controller

DI – Dependency injection (Впровадження залежностей)

БД – База даних

UI – User Interface(Інтерфейс користувача)

ВСТУП

В сучасному світі розробка програмного забезпечення посідає одне з найважливіших та найвпливовіших місць серед інших галузей. Це зумовлене тим що цифровий світ зростає та поширюється кожен день та набирає величезних обертів останні роки. Важливо розуміти який продукт потрібен користувачам та як саме його зробити зручним для користування.

На сьогоднішній день обмін цифровою інформацією одна з найпоширеніших тем серед людей всього світу. Це досягається за рахунок того що обмін такою інформацією є зручним та швидким способом отримання знань та навичок. Кожного дня тисячі і навіть мільйони користувачів у всьому світі читають, публікують та дізнаються нову інформацію з інтернету. Ця статистика не може не вражати.

Обмін інформацією для студентів є не менш обширною та затребуваною темою. Щодня студенти шукають корисну інформацію для навчання на просторах інтернету, намагаються знайти ресурси та поради які допоможуть в навчанні. З розвитком дистанційного навчання та відсутності живої комунікації ця тема тільки більше набрала обертів. Є потреба комунікації та обміну інформацією саме серед студентів. Але чи є такий ресурс який допоможе їм обмінюватися думками та публікувати корисну інформацію?!

Нажаль більшість застосунків націлені на широку аудиторію та зайву функціональність. В таких застосунках знайти або поділитися корисною інформацією для інших стає реальним випробуванням. Також такі додатки не орієнтовані на студентів а пропонують функціонал для звичайних людей.

Саме тому розробка застосунку для студентів який дозволить знаходити та опубліковувати інформацію є актуальною темою на сьогодні та принесе багато користі оточуючому світу.

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМИ ЗНАХОДЖЕННЯ І ОПУБЛІКУВАННЯ ІНФОРМАЦІЇ СТУДЕНТАМИ ТА ПОРІВНЯННЯ ІСНУЮЧИХ РІШЕНЬ

1.1. Існуюча система для знаходження та публікування інформації

Як еталон платформи для опублікування та знаходження інформації я взяв ресурс під назвою Habr. Це велика платформа яка користується популярністю серед великого сегменту користувачів. Вона містить велике різноманіття функціональності а саме:

- Публікування статей
- Знаходження інформації
- Підрозділи для різного типу інформації
- Функціонал коментарів
- Функціонал пошуку
- Власний обліковий запис

На головній сторінці ресурсу(Рис 1.1) розміщено інформацію про категорії та розділи інформації яка присутня на платформі. Також тут можна побачити функціонал пошуку інформації, можна дізнатися як стати автором статей, скільки нових статей було опубліковано в кожній категорії, є навігаційна панель зверху сайту та іконка входу у власний обліковий аккаунт.

При вході в обліковий запис є різні варіанти авторизації через Google, GitHub, Twitter, Facebook та інші. Після входу в особистий кабінет користувач має змогу побачити свій обліковий запис та можливості які йому надані, а саме:

- Переглянути свої статті
- Переглянути діалоги
- Переглянути коментарі
- Переглянути закладки

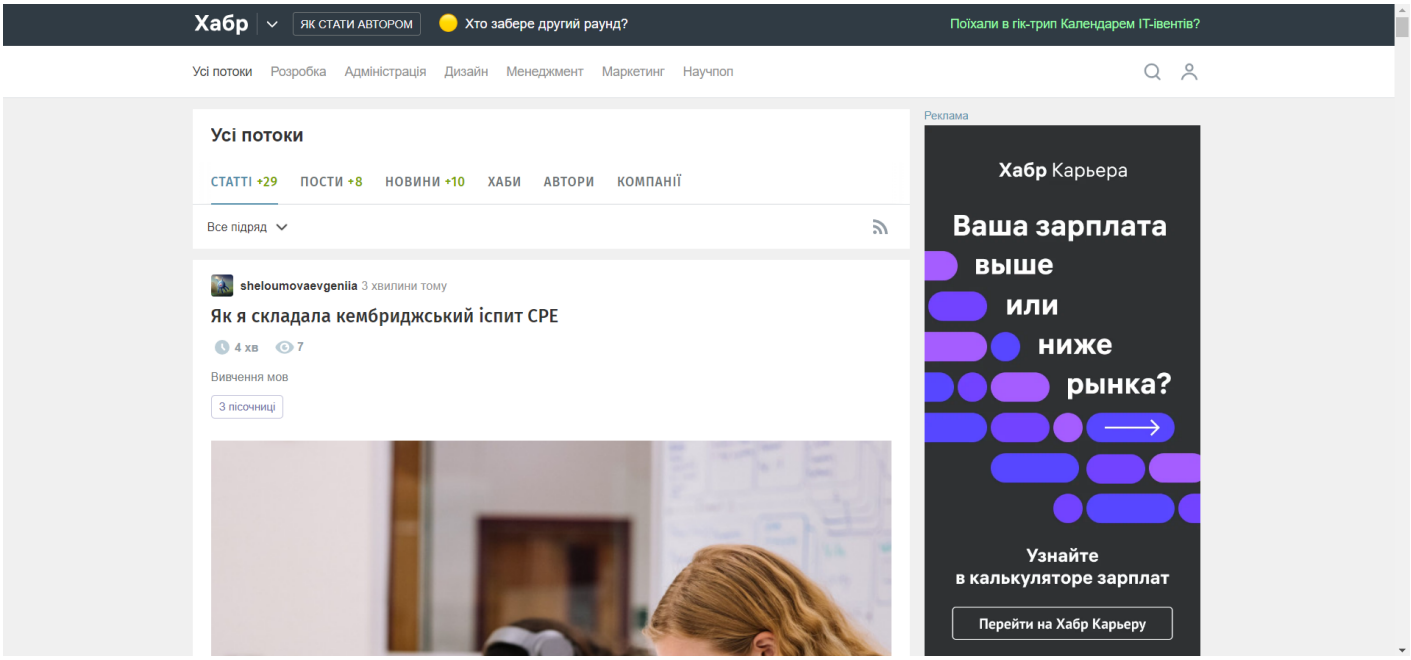

Рис.1.1 Головна сторінка Habr.

Багато функціональності яка дозволяє робити публікації та якщо переглянути як стати автором статей(Рис 1.2) то можна побачити що додаток пропонує функціонал написання публікацій і відправлення їх до так званої “Пісочниці” де ці публікації будуть розглядатися для подальшої взаємодії з реальними читачами. Якщо не впевнені в актуальності своєї інформації можна звернутися до кураторів які нададуть підтримку.

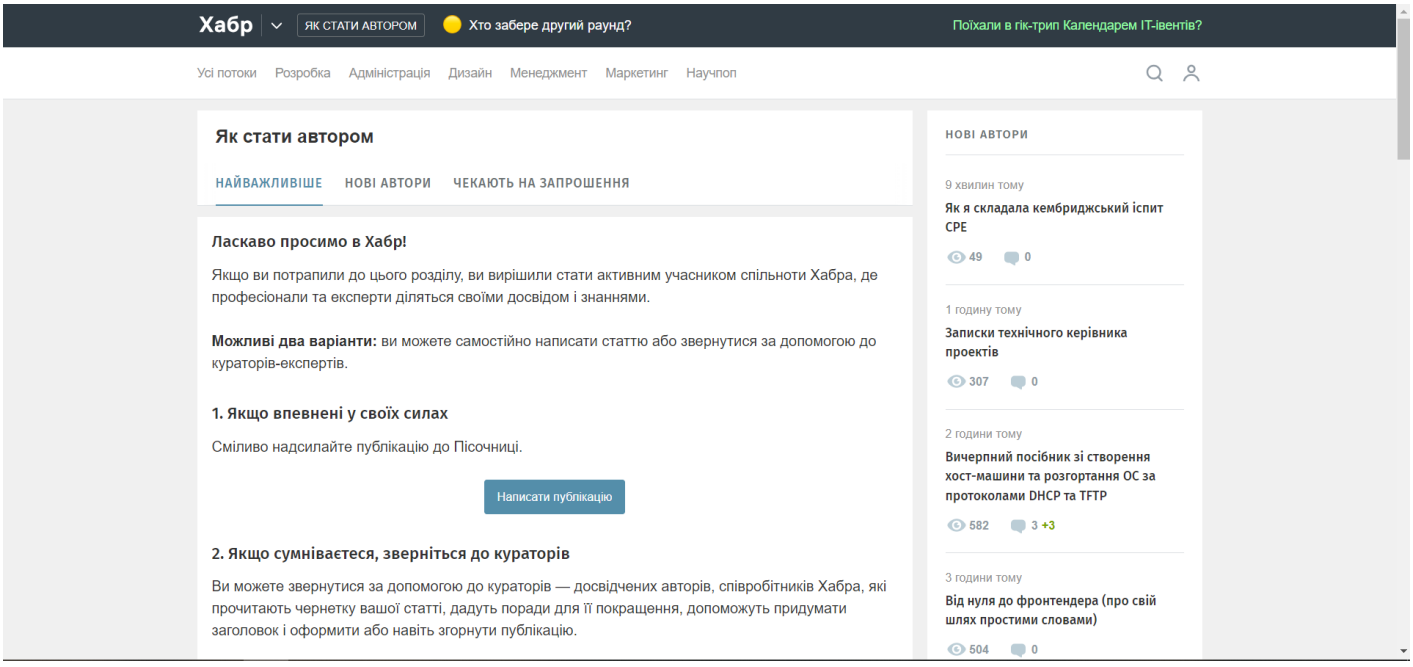

Рис.1.2 Як стати автором Habr.

Згідно з інформацією за 2022 рік опублікованою на цьому ресурсі у Habr було написано приблизно 30000 публікацій(див Таблиця 1.1) та більше 200 мільйонів переглядів що є феноменальною цифрою[4].

Таблиця 1.1

Річна статистика публікацій Habr.

Рік	**Кількість статей**
2020	26 тис.
2021	28 тис.
2022	30 тис.

Але цей додаток ніяким чином не орієнтований на студентів і містить перевантаження функцій та можливостей для даного типу користувачів. На ресурсі Habr студенти не мають змоги швидко знайти потрібну їм інформацію по темі яка їх цікавить, а також не мають змоги опублікувати інформацію якою вони хотіли би поділитися в пару кліків.

Виходячи з усього цього, необхідно створити програмний засіб для потреб саме студентів, який дозволить знаходити та публікувати інформацію на ресурсі без зайвих затрат. Звичайно всі статі повинні будуть бути оброблені адміністратором в продовж доби щоб виставляти лише якісний контент для студентів.

Це буде веб-додаток який буде доступний користувачам з усього світу. Основними функціями додатку будуть:

- Пошук інформації за фільтром
- Можливість публікування статті
- Категорії статей
- Можливість додавання фото для статті
- Облікові аккаунти користувачів

Наразі ця система буде ще більш актуальна у зв’язку з постійним дистанційним навчанням та не можливості ходити студентів до університету через війну в країні. Я

впевнений що даний додаток принесе користь багатьом студентам які прагнуть до саморозвитку та черпання знань з інтернету.

1.2. Аналіз на основі літературних джерел існуючих аналогів програмних засобів по знаходженню та публікуванню інформації для студентів

Перед розробкою застосунку дуже важливо проаналізувати всі існуючі додатки які забезпечують схожу функціональність для користувачів. Проаналізувавши такі аналоги можна виявити недоліки які варто покращити в своїй системі.

Першим аналогом можна вважати платформу Medium. Medium - це платформа для публікації блогів та статей. На головній сторінці(Рис 1.3) можна побачити навігаційну панель та можливості які пропонує система а саме написання та читання контенту з платформи.

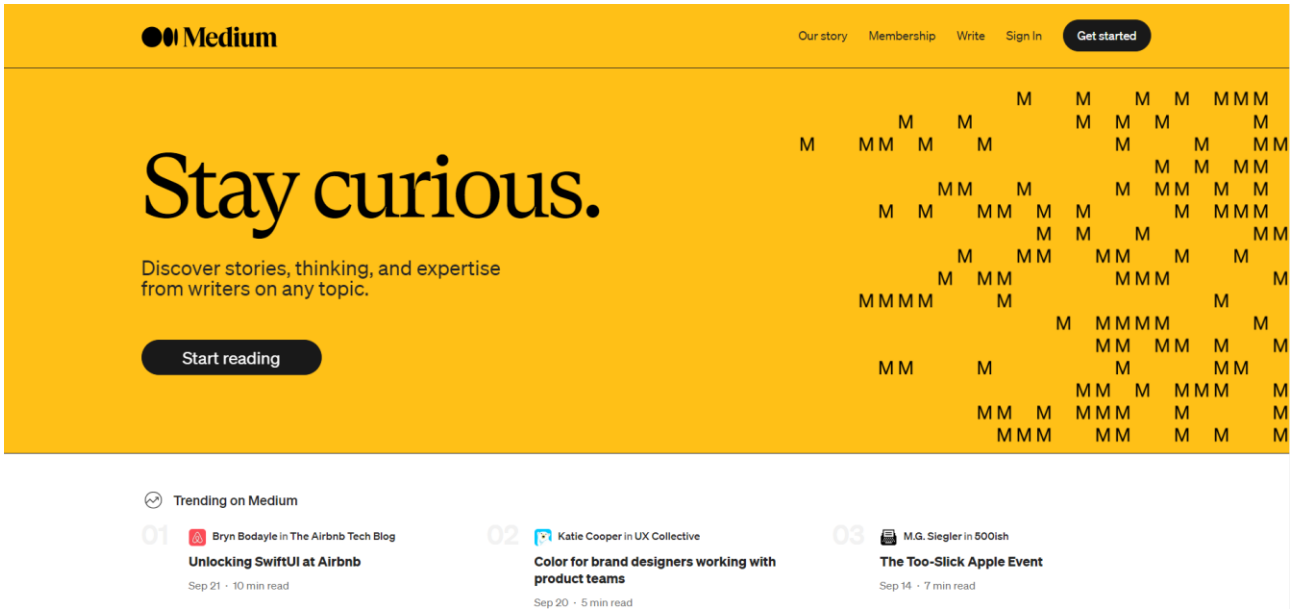

Рис.1.3 Головна сторінка Medium

Також можна ознайомитись з виглядом та контентом статей які пропонує платформа(Рис 1.4)

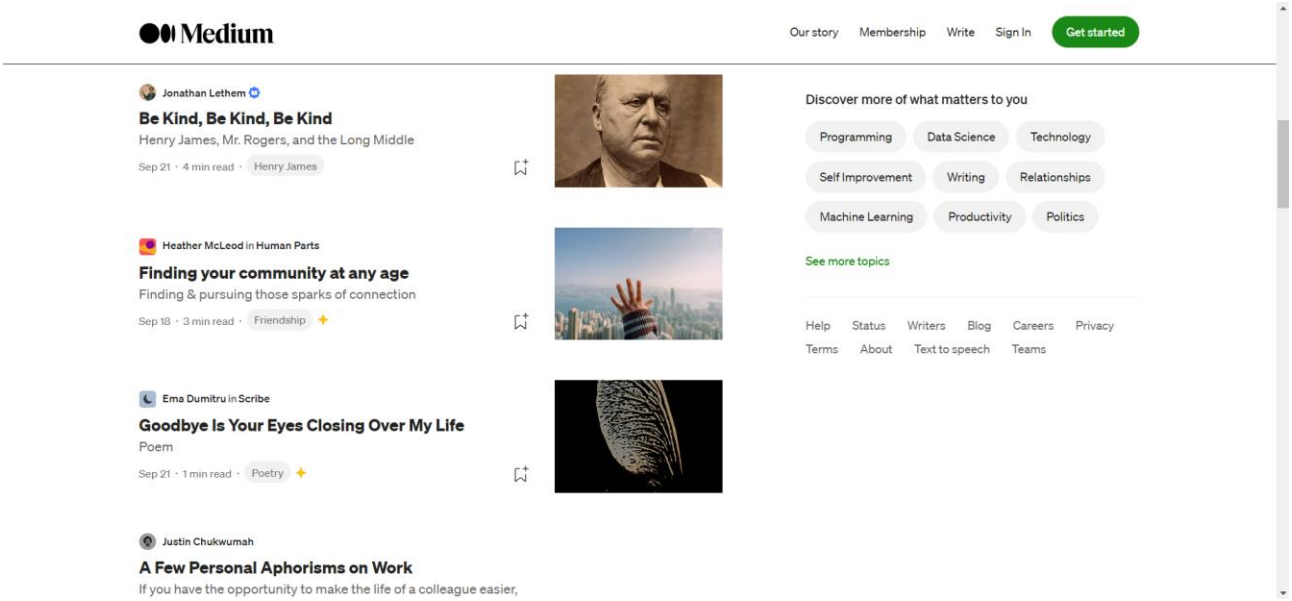

Рис.1.4 Приклад статей Medium

Для того щоб почати роботу з платформою користувачу доведеться зареєструватися за допомогою одного з декількох варіантів на вибір(Google, Facebook або пошта). Після реєстрації користувач потрапляє на сторінку де йому запропоновано оплатити послуги користування(Рис. 1.5).

Нажаль ресурс не є безкоштовним та доведеться оформлювати підписку для того щоб була змога переглядати та ділитися статтями що на мою думку є одним з найголовніших недоліків цього додатку.

І хоч ця платформа відкрита для всіх користувачів, володіє доволі зручним інтерфейсом та можливістю коментування, але вона не спеціалізована на студентському контенті. Для студентів дуже важливо мати можливість знайти інформацію яка відповідає їх потребам, нажаль на цій платформі це зробити складно.

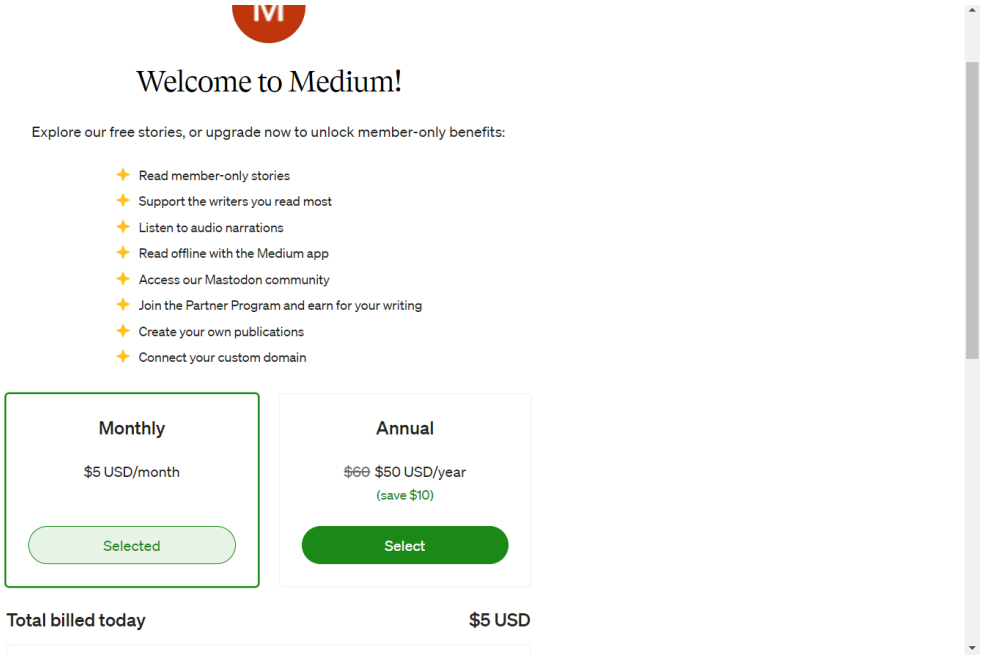

Рис.1.5 Тарифний план Medium

Наступним аналогом є EduBlogs(Рис 1.6) – це сервіс для студентів та вчителів, який дозволяє створити освітні блоги. Він спрямований на академічний контент, але може бути обмежений у функціоналі для більш розширених потреб.

Основні функції які надає додаток:

- Пошук освітніх блогів
- Створення освітніх блогів
- Будування ком’юніті користувачів

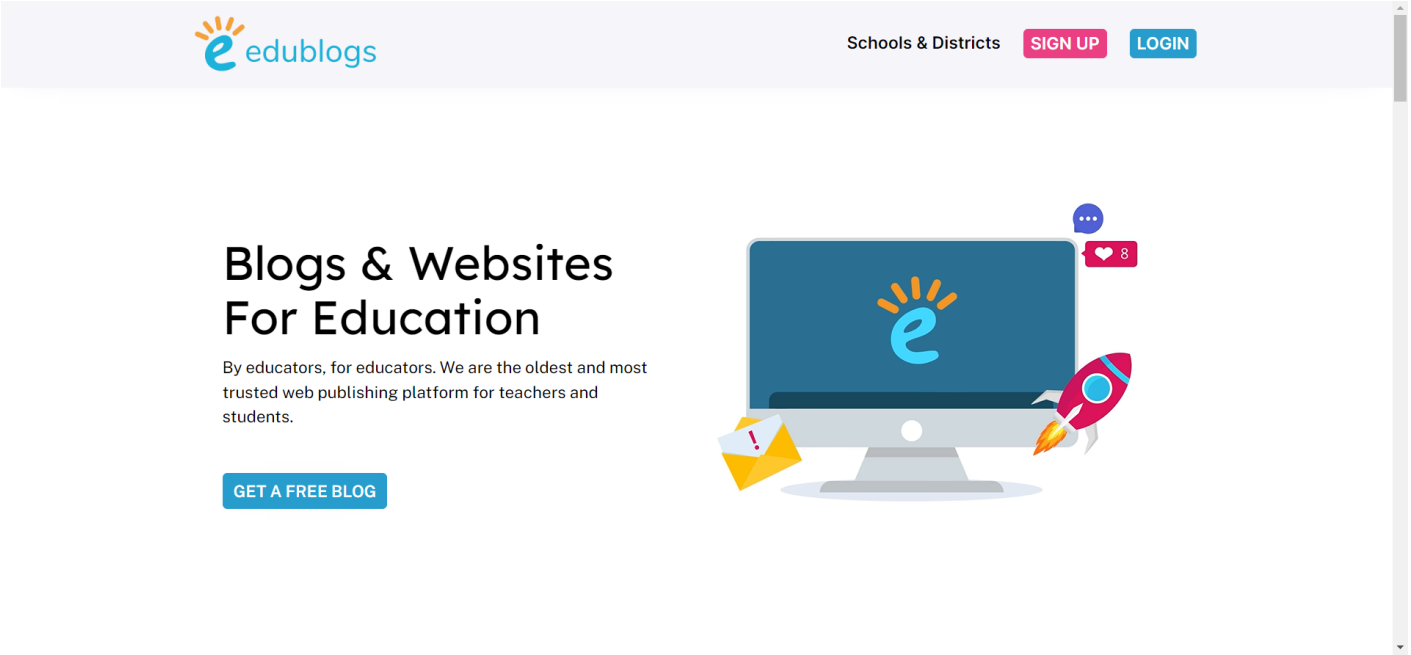

Рис.1.6 Головної сторінка додатку

Даний додаток зроблено дуже складним. Інтерфейс інтуїтивно не зрозумілий, щоб користуватися цією платформою доведеться витратити багато часу на розбирання можливостей платформи. Зазвичай це відлякує користувачів.

Наступний можливий конкурент це LinkedIn(Рис. 1.7). Насправді LinkedIn не позиціонує себе як платформу по написанню та пошуку статей користувачами. Позиція цієї платформи - це професійна соціальна мережа для користувачів де вони мають змогу вести своє професійне життя, розширювати свою професійну мережу та робити різноманітні публікації на своїх сторінках. Але якщо розглянути цю платформу глибше то можна зрозуміти що вона включно надає і можливості публікації та пошуку статей для різних типів людей.

В LinkedIn присутні функції:

- Будування мережі контактів
- Публікування статей та публікацій
- Написання коментарів
- Пошук роботи
- Особисті переписки

Але незважаючи на всі переваги платформи є один величезний мінус – додаток не орієнтований на потреби студентів і не може замінити платформу для публікування та пошуку інформації студентами через те що всі ці функції ускладнюють першочергову мету та ціль.

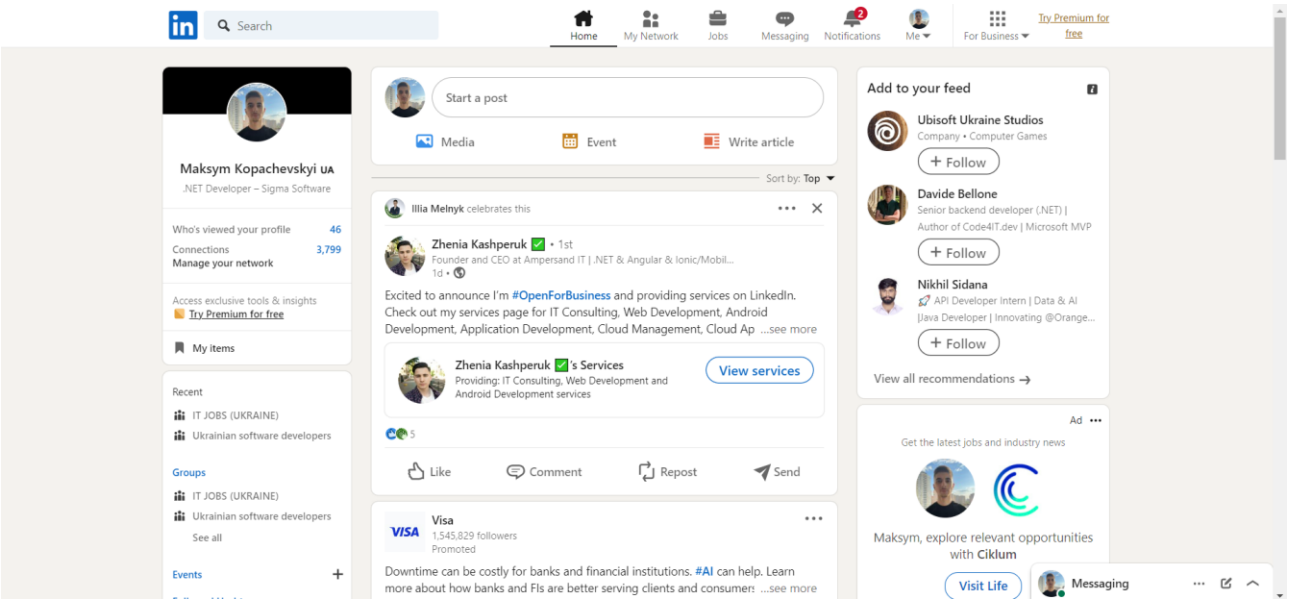

Рис.1.7 Головної сторінка LinkedIn

Виходячи з інформації зібраної по аналогам додатку можна проаналізувати загальні переваги(Таблиця 1.2) та недоліки(Таблиця 1.3) які було знайдено при розгляді цих систем.

Таблиця 1.2

Переваги існуючих систем

Назва	**Детальний опис**
Зручний інтерфейс	Багато існуючих систем мають інтуїтивний та зручний інтерфейс для публікації контенту
Спільнота користувачів	Деякі платформи вже мають активну спільноту студентів та інших

	користувачів, що сприяє обміну знаннями та досвідом
Можливість взаємодії	На деяких платформах можна легко обговорювати статті, давати коментарі та спілкуватися з іншими авторами

Таблиця 1.3

Недоліки існуючих систем

Назва	**Детальний опис**
Обмежений функціонал	Деякі існуючі системи можуть бути обмежені у функціоналі для створення та публікації інформації.
Нестабільність	Деякі платформи можуть мати проблеми з надійністю та стабільністю роботи з великою кількістю користувачів.
Брак спеціалізації	Всі розглянуті платформи не орієнтовані на потребі студентів в обміні та публікації інформації.

1.3. Пропозиції по вимогам до створюваного застосунку

Структура програмної системи: даний програмний застосунок повинен бути веб-застосунком для якого не буде важлива яка операційна система встановлена. Систему необхідно розробити використовуючи .Net Core платформу. Також даний застосунок повинен бути легко масштабованим для можливого розширення в майбутньому. Основною парадигмою для розробки веб-додатку є об’єктоорієнтовна парадигма програмування. Використовуючи платформу .Net Core основною мовою

програмування повинен бути C# який є строго типізованою мовою та використовую розробку в стилі ООП.

Через те що я працюю як .Net Developer при написанні застосунку пошуку та публікування інформації студентами я обрав саме мову програмування C# та стек технологій .Net Core.

Чому саме C#:

- C# є строго типізованою мовою для розробки серверної сторони додатку;
- Це одна з найпопулярніших мов програмування у світі;
- Він простий у використанні та надає величезний спектр можливостей;
- Має велику підтримку комьюніті;
- C# — це об'єктно-орієнтована мова, яка дозволяє повторно використовувати код, знижуючи витрати на розробку та надає чітку структуру програмам;
- Платформа .Net надає багато можливостей написання безпечного коду;

Зазвичай при написанні коду на C# використовують одну з найпопулярніших IDE – Visual Studio. Але на мою думку в цієї IDE є ряд мінусів такі як низький перформенс на великих додатках, погані інструменти рефакторингу коду та інше. Саме тому я обираю для розробки IDE Rider яка розроблена компанією JetBrains і під капотом містить інструмент рефакторингу Resharper. Також ця IDE показує значно кращі показники перформенсу на великих проектах та має інструменти аналізу показників перморменсу.

Висновок до розділу 1

Мною було досліджено актуальність застосунку для студентів що дозволяє знаходити та опубліковувати інформацію та аналоги представлені на ринку.

В результаті досліджень можна зробити висновок що розглянуті аналоги застосунку є в більшості випадків зручними та непогано спроектованими але не задовольняють потреби студентів та не містять необхідних функцій та інформації яка

необхідна для пошуку та публікування інформації. Також проаналізувавши тенденції ринку можна зробити висновок що дана тема є дуже актуальною та стане ще більш потрібною в найближчому майбутньому.

РОЗДІЛ 2
ВИМОГИ ДО ЗАСТОСУНКУ ДЛЯ ОПУБЛІКУВАННЯ ТА ЗНАХОДЖЕННЯ ІНФОРМАЦІЇ ДЛЯ СТУДЕНТІВ

2.1. Функціональні вимоги до програмного застосунку

Вимоги до програмного забезпечення це своєрідний опис всіх функціональних цілей системи та можливостей за допомогою яких вона буде розроблена. Очікування від користувачів від користування системою визначають саме вимоги до неї. Вони можуть бути різних типів, таких як невідомі чи відомі, приховані або очевидні, неочікуваними або ж очікуваними. Всі вони визначаюсь як система повинна поводитися при певних діях від користувача.

Основними типами вимог являються функціональні та нефункціональні вимоги.

Основна відмінність одного типу від іншого полягає в тому що функціональні вимоги пояснюють що повинно робити саме програмне забезпечення, в той час як нефункціональні пояснюють якою повинна бути система.

Також частиною нефункціональних вимог є покращення та вдосконалення які допомагають зробити систему кращою, відповідаючи кращим канонам розробки програмного забезпечення.

Функціональна вимога - це опис послуги, яку повинно надавати програмне забезпечення. Вона описує програмну систему або її компонент. Функція - це щось інше, як вхідні дані для програмної системи, її поведінка та вихідні дані. Це може бути розрахунок, маніпуляція з даними, бізнес-процес, взаємодія з користувачем або будь-яка інша специфічна функціональність, яка визначає, яку функцію повинна виконувати система. Функціональні вимоги в програмній інженерії також називають функціональною специфікацією[7].

В інженерії програмного забезпечення та системній інженерії функціональні вимоги можуть варіюватися від абстрактного викладу потреби відправника на високому рівні до детальних математичних специфікацій функціональних вимог. Функціональні вимоги до програмного забезпечення допомагають вам зафіксувати передбачувану поведінку системи.

Функціональні вимоги до системи повинні включати наступні речі:

- Деталізація операцій, що проводяться на кожному екрані
- Логіка обробки даних повинна бути введена в систему
- Повинні бути описи системних звітів або інших вихідних даних
- Повна інформація про робочі процеси, що виконуються системою
- Необхідно чітко визначити, кому буде дозволено створювати/модифікувати/видаляти дані в системі
- У функціональному документі має бути описано, як система буде виконувати відповідні регуляторні потреби.

Нижче наведені переваги створення типового документа з функціональними вимогами

- Допомагає перевірити, чи додаток забезпечує всі функціональні можливості, які були згадані у функціональних вимогах до цього додатку
- Документ функціональних вимог допомагає визначити функціональність системи або однієї з її підсистем.
- Функціональні вимоги разом з аналізом вимог допомагають виявити відсутні вимоги. Вони допомагають чітко визначити очікуваний сервіс і поведінку системи.
- Помилки, виявлені на етапі збору функціональних вимог, найдешевше виправити.
- Підтримуйте цілі, завдання чи діяльність користувачів

Ось найпоширеніші типи функціональних вимог:

- Обробка транзакцій
- Бізнес-правила
- Вимоги до сертифікації
- Вимоги до звітності
- Адміністративні функції
- Рівні авторизації
- Відстеження аудиту
- Зовнішні інтерфейси
- Управління історичними даними
- Правові та регуляторні вимоги

Нижче наведено приклади популярних функціональних вимог:

- Програмне забезпечення автоматично перевіряє клієнтів на відповідність системі управління контактами ABC
- Система продажів повинна дозволяти користувачам реєструвати продажі клієнтів
- Колір фону для всіх вікон програми має бути синім і мати шістнадцяткове значення кольору RGB 0x0000FF.
- Право на перегляд даних про доходи мають лише працівники управлінського рівня.
- Програмний комплекс повинен бути інтегрований з банківським API
- Програмний комплекс повинен відповідати вимогам доступності

Важливою найкращою практикою для розробки документа з функціональними вимогами є наступне:

- Не об'єднуйте дві вимоги в одну. Зберігайте вимоги деталізованими.

- Ви повинні зробити кожну вимогу максимально повною і точною.
- Документ повинен містити всі технічні вимоги.
- Зіставте всі вимоги з цілями та принципами, які сприяють успішній розробці програмного забезпечення
- З'ясовуйте вимоги, використовуючи інтерв'ю, семінари та невимушене спілкування.
- Якщо існує будь-яке відоме, перевірене обмеження, яке суттєво впливає на вимогу, то це критичний стан, який повинен бути задокументований.
- Необхідно, щоб ви задокументували всі припущення в документі.

Нижче наведені деякі типові помилки, яких припускаються при створенні документа з функціональними вимогами:

- Внесення невиправданої додаткової інформації, яка може заплутати розробників
- Недостатня деталізація документа з вимогами.
- Ви додаєте правила або приклади, твердження про масштабування або цілі - все, що завгодно, окрім самої вимоги.
- Ви пропустили важливу інформацію, яка є абсолютною необхідністю для повного, точного і остаточного формулювання вимоги.
- Деякі фахівці починають захищати вимоги, які вони задокументували, коли вимога змінюється, замість того, щоб знайти правильну істину.
- Вимоги, які не співвіднесені з цілями чи принципами.

Кінцевими користувачами даного програмного забезпечення виступають студенти тому система повинна бути добре спроектована щоб відповідати зручностям в використанні та навантаженням на неї багатої кількості користувачів. Потреби користувачів повинні бути гарно проаналізовані та сформовані згідно з потреб та цілей програмного застосунку.

Ключові функції, які має виконувати застосунок для опублікування та знаходження інформації студентами – це:

- реєстрація та створення облікового запису
- вхід в обліковий запис використовуючи пароль та логін
- можливість перегляду статей
- можливість опублікування статей
- сортування статей за категоріями
- перегляд дати опублікування статті
- перегляд автора статті
- пошук статей за фільтром
- вихід з особистого кабінету

Щоб користувач мав можливість публікувати інформацію треба мати можливість створення облікового запису та механізму авторизації який буде спільним для всіх юзерів. При авторизації користувач повинен ввести пароль та логін(рис 2.1) після чого йому будуть надані відповідні права. Логіном слугуватиме електронна адреса. Якщо користувач не авторизувався він може бачити контент але не може його додавати. Ця система дозволить позбавитися від анонімних статей які можуть нести не правдиву інформацію. Всі статі будуть містити ім’я автора публікації тому кожний охочий зможе зв’язатися з автором напряму.

Для того щоб валідувати контент застосунку необхідно встановити роль адміністратора, який буде обробляти та підтверджувати актуальність публікацій. Якщо адміністратор вважає що контент публікації не є прийнятним або оформлений не правильно він може відхилити запит на публікацію пояснивши свої аргументи автору. Якщо контент задовільний адміністратор підтверджує це, стаття публікується в додатку і її можуть побачити всі інші юзери. Для реалізації цих цілей необхідна наявність адмін панелі яку буде бачити лише користувач з правами адміністратора. Основні функції які повинна мати панель адміністратора:

- Перегляд нових статей які надійшли на валідування

- Можливість відхилити публікацію
- Можливість додати коментар до відхилення
- Можливість підтвердження публікацій
- Можливість видалення неактуальних публікацій
- Можливість навігації та пошуку даних

Вся інформація буде відображена у вигляді таблиці та кнопок для навігації. На адмін панелі повинні міститися наступні кнопки: Підтвердити/Відхилити/Видалити.

Вимоги користувача яким повинен відповідати застосунок по пошуку та публікуванню інформації студентами::

- Публікування інформації
- Перегляд інформації
- Пошук інформації

2.2. Не функціональні вимоги до програмного застосунку

Для того щоб зрозуміти якою система повинна бути важливо сформувати всі основні не функціональні вимоги і проаналізувати які характеристики повинна мати система, яким апаратним можливостям повинна слідувати система, які програмні та операційні вимоги представлені до застосунку.

Нефункціональні вимоги - це набір специфікацій, які описують робочі можливості та обмеження системи і спрямовані на покращення її функціональності. По суті, це вимоги, які визначають, наскільки добре система буде працювати, включаючи такі речі, як швидкість, безпека, надійність, цілісність даних і т.д[8].

Функціональні та нефункціональні вимоги

Як функціональні, так і нефункціональні вимоги описують конкретні характеристики, які повинен мати продукт, щоб задовольнити потреби зацікавлених сторін і самого бізнесу. Але, як ви можете зрозуміти з назви, вони зосереджені на різних речах.

Функціональні вимоги визначають, що повинен робити програмний продукт: його можливості та функції. Прикладом функціональної вимоги для месенджера може бути щось на кшталт: "Користувач повинен мати можливість редагувати повідомлення після їх відправлення, щоб виправити помилки".

Нефункціональні вимоги визначають атрибути якості системи, звідси їхня друга назва - атрибути якості. Продовжуючи наш приклад з платформою для обміну повідомленнями, нефункціональною вимогою може бути швидкість, з якою система повинна виконувати редагування, щоб задовольнити очікування користувачів: "Повідомлення повинно оновлюватися для всіх користувачів у чаті протягом 0,1 секунди, за умови, що всі користувачі знаходяться онлайн і мають LTE-зв'язок або краще".

Які основні типи нефункціональних вимог?

Найпоширеніші з них - продуктивність, масштабованість, портативність, сумісність, надійність, доступність, ремонтопридатність, безпека, локалізація та юзабіліті. Але існує досить багато типів нефункціональних вимог, які також можуть братися як приклад.

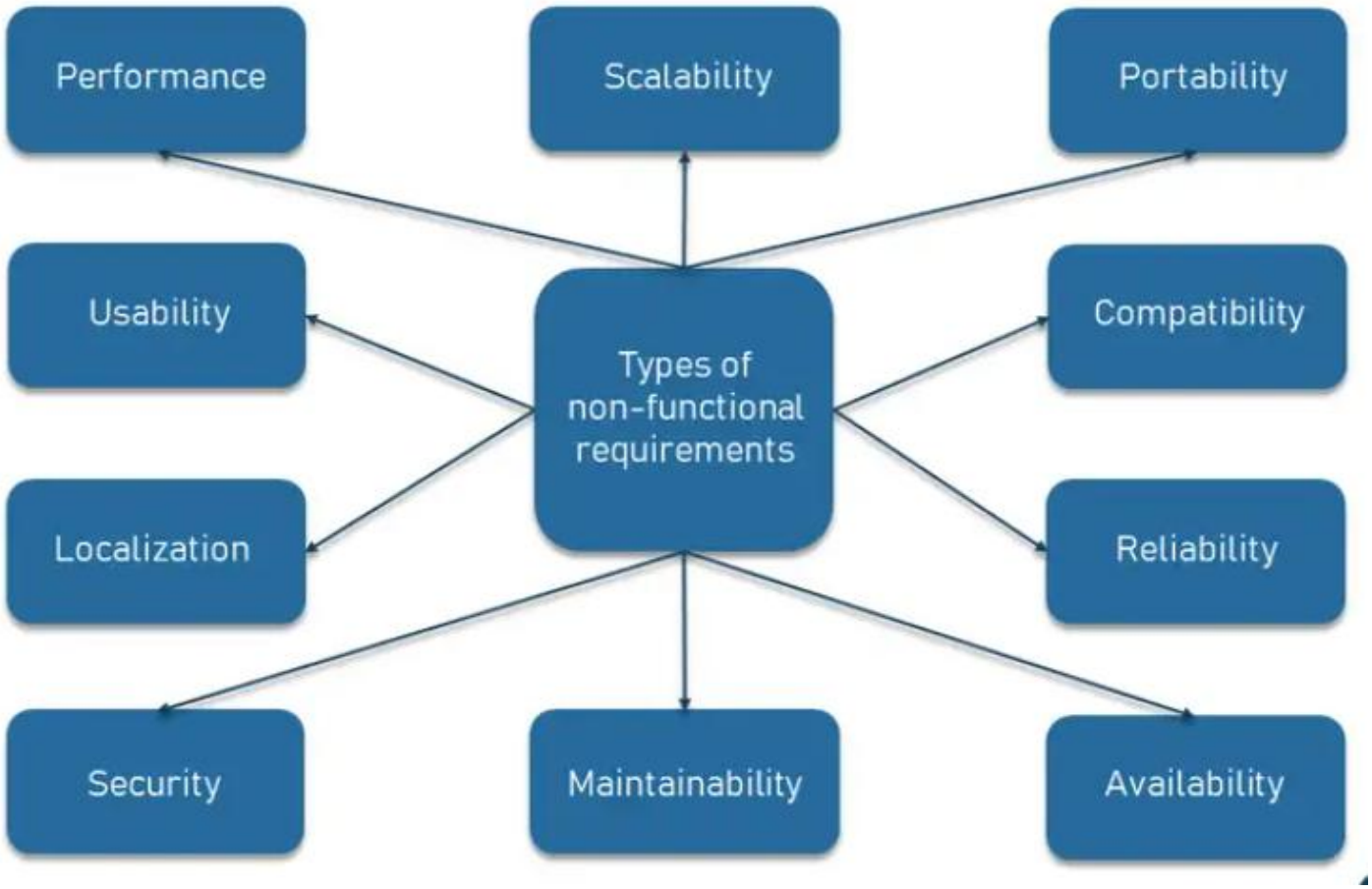

Рис.2.1 Типи не функціональних вимог

Продуктивність і масштабованість. Як швидко система повертає результати? Наскільки зміниться ця продуктивність при збільшенні робочого навантаження?

Переносимість та сумісність. На якому обладнанні, операційних системах і браузерах, а також їхніх версіях працює програмне забезпечення? Чи конфліктує воно з іншими програмами та процесами в цих середовищах?

Надійність, ремонтопридатність, доступність. Як часто система зазнає критичних збоїв? Скільки часу потрібно для усунення проблеми, коли вона виникає? І як час доступності для користувачів співвідноситься з часом простою?

Безпека. Наскільки добре система та її дані захищені від атак?

Локалізація. Чи сумісна система з місцевою специфікою?

Юзабіліті. Наскільки легко клієнту користуватися системою?

Продуктивність і масштабованість - це дві основні нефункціональні вимоги, без яких не може обійтися жодна система. Оскільки вони йдуть пліч-о-пліч, ми об'єднали їх в один розділ.

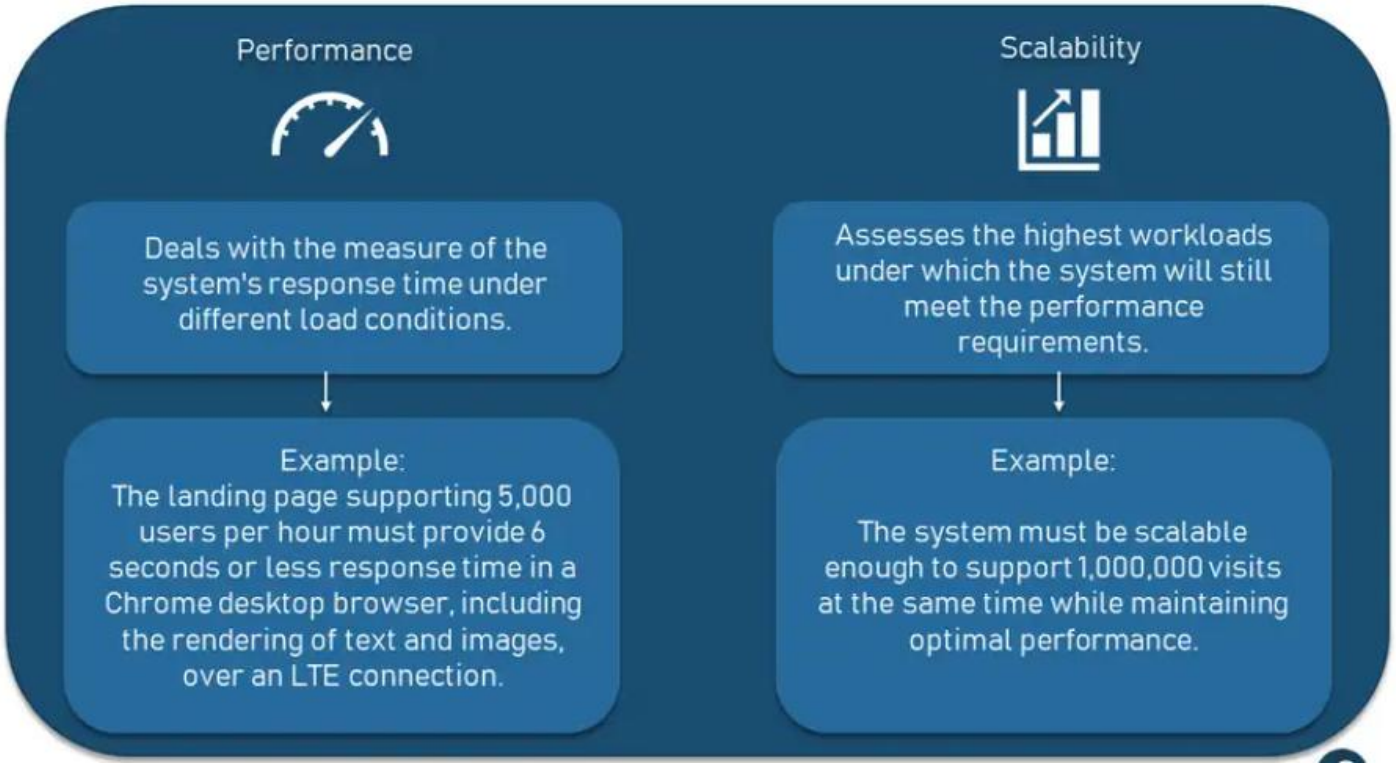

Рис 2.2 Продуктивність і масштабованість

Продуктивність

Продуктивність визначає, наскільки швидко програмна система або її окрема частина реагує на дії користувачів за певного робочого навантаження. У більшості випадків ця метрика пояснює, скільки часу користувач повинен чекати, перш ніж відбудеться цільова операція (рендеринг сторінки, обробка транзакції тощо), враховуючи загальну кількість користувачів на даний момент. Але це не завжди так. Вимоги до продуктивності можуть описувати фонові процеси, невидимі для користувачів, наприклад, резервне копіювання. Але давайте зосередимося на продуктивності, орієнтованій на користувача.

Приклад вимог до продуктивності:

Цільова сторінка, що підтримує 5 000 користувачів на годину, повинна забезпечувати час відгуку 6 секунд або менше в десктопному браузері Chrome, включаючи рендеринг тексту і зображень, а також через LTE-з'єднання.

Масштабованість

Масштабованість оцінює найвищі робочі навантаження, за яких система все ще відповідатиме вимогам до продуктивності. Існує два способи масштабування системи в міру зростання робочих навантажень: горизонтальне та вертикальне масштабування.

Горизонтальне масштабування забезпечується додаванням нових машин до пулу серверів.

Вертикальне масштабування досягається додаванням більшої кількості процесорів та оперативної пам'яті до існуючих машин.

Приклад вимог до масштабованості:

Система повинна бути достатньо масштабованою, щоб підтримувати 1 000 000 відвідувань одночасно, зберігаючи при цьому оптимальну продуктивність.

Ще двома ключовими гравцями у світі нефункціональних вимог є такі атрибути, як портативність та сумісність.

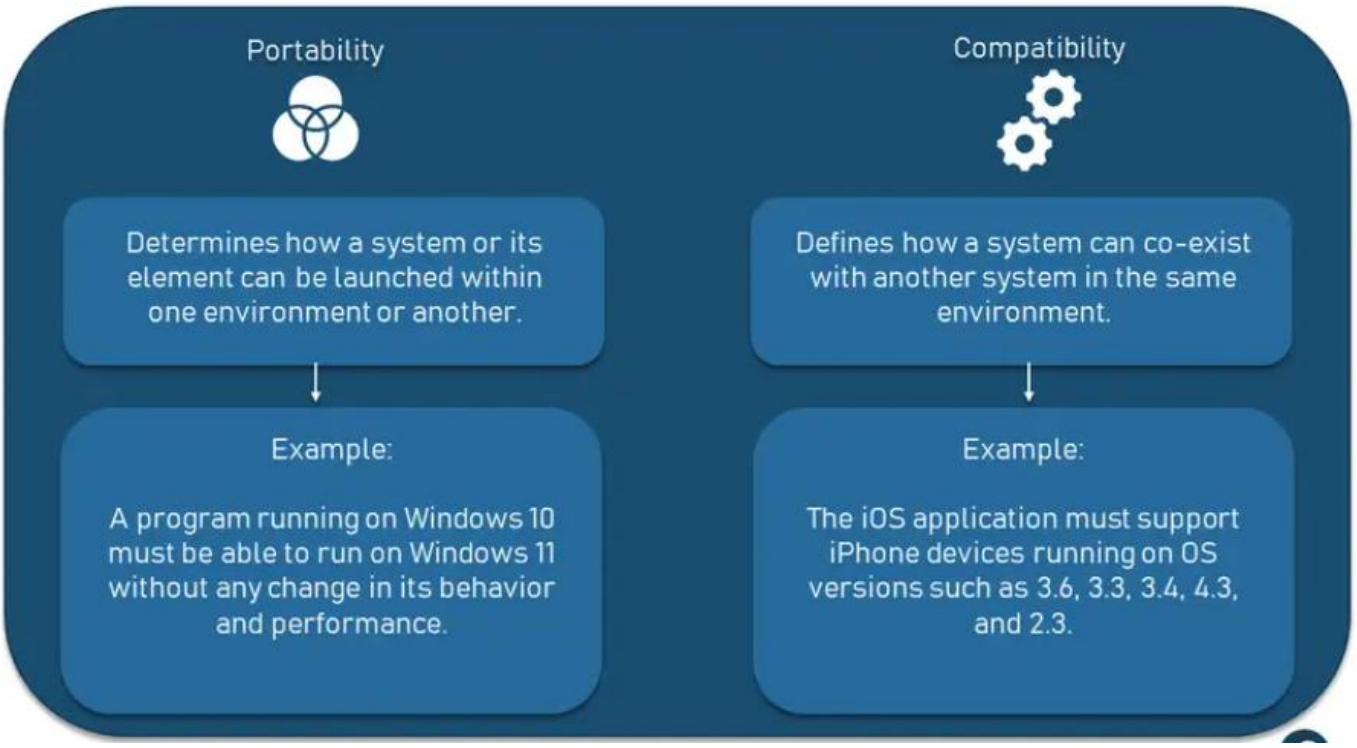

Рис. 2.3 Портативність та сумісність

Переносимість

Переносимість визначає, як систему або її елемент можна запустити в тому чи іншому середовищі. Зазвичай вона включає в себе специфікації апаратного, програмного забезпечення або інших платформ використання. Простіше кажучи, вона визначає, наскільки добре дії, виконані на одній платформі, виконуються на іншій. Крім того, він визначає, наскільки добре елементи системи можуть бути доступні та взаємодіяти з двох різних середовищ.

Приклад вимог до переносимості:

Програма, що працює на Windows 10, повинна мати можливість працювати на Windows 11 без будь-яких змін у своїй поведінці та продуктивності.

Сумісність

Сумісність, як додатковий аспект переносимості, визначає, як система може співіснувати з іншою системою в одному середовищі. Наприклад, програмне забезпечення, встановлене в операційній системі, має бути сумісним з її брандмауером або антивірусним захистом.

Приклад вимог до сумісності:

Додаток для iOS повинен підтримувати пристрої iPhone, що працюють на версіях операційної системи:

- 3.6
- 3.3
- 3.4
- 4.3
- 2.3

Переносиміcть і сумісність встановлюються з точки зору операційних систем, апаратних пристроїв, браузерів, програмних систем та їхніх версій. На сьогоднішній день кросплатформеність, кросбраузерність та адаптивність до мобільних пристроїв є загальним стандартом для веб-додатків.

Нефункціональні вимоги до портативності зазвичай ґрунтуються на попередньому дослідженні ринку, польових дослідженнях або аналітичних звітах про типи програмного забезпечення та пристроїв, якими користується цільова аудиторія. Якщо ви працюєте в корпоративному середовищі і доступ до програмного забезпечення буде здійснюватися через задокументований список пристроїв і операційних систем, визначити сумісність і портативність досить легко.

Надійність, ремонтопридатність, доступність

Хоча ці три типи вимог зазвичай документуються окремо, ми об'єднали їх в одному розділі, оскільки вони підходять до однієї і тієї ж проблеми з різних сторін. Ще одна

річ, яку слід мати на увазі з цими вимогами, - це те, що їх надзвичайно важко виразити в термінах обчислення. І, чесно кажучи, багато системних провайдерів взагалі не документують їх.

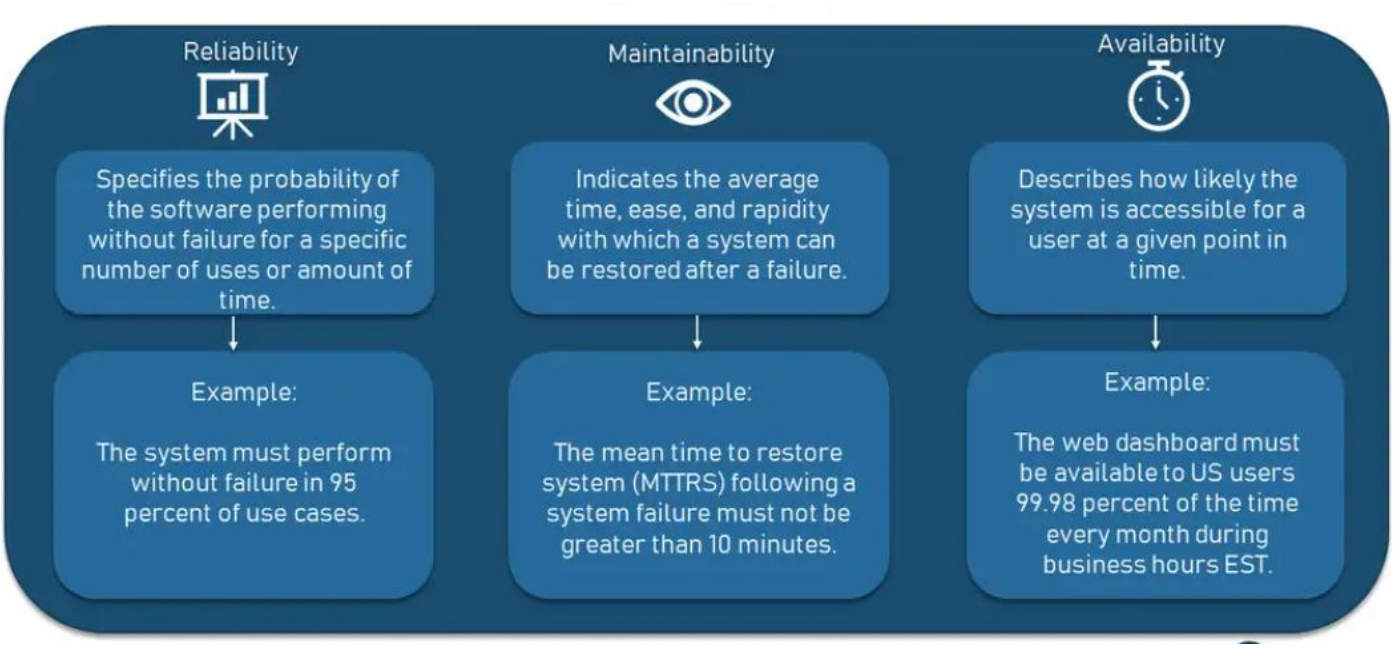

Рис 2.4. Надійність та доступність

Надійність

Надійність визначає, наскільки ймовірно, що система або її елемент буде працювати без збоїв протягом певного періоду часу за заданих умов. Традиційно ця ймовірність виражається у відсотках. Наприклад, якщо система має 85-відсоткову надійність протягом місяця, це означає, що протягом цього місяця, за нормальних умов використання, існує 85-відсоткова ймовірність того, що система не зазнає критичних збоїв.

Як ви вже здогадалися, досить складно визначити критичну відмову, час і нормальні умови використання. Інший, дещо простіший підхід до цієї метрики полягає в підрахунку кількості критичних помилок, виявлених у виробництві за певний період часу, або в обчисленні середнього часу напрацювання на відмову.

Ремонтопридатність

Ремонтопридатність визначає час, необхідний для того, щоб рішення або його компонент було виправлено, змінено для підвищення продуктивності або інших якостей, або адаптовано до мінливого середовища. Як і надійність, вона може бути виражена як ймовірність ремонту протягом певного часу. Наприклад, якщо ремонтопридатність протягом 24 годин становить 75 відсотків, це означає, що існує 75-відсоткова ймовірність того, що компонент може бути відремонтований протягом 24 годин. Ремонтопридатність часто вимірюється за допомогою такого показника, як MTTRS - середній час відновлення системи.

Доступність

Доступність описує, наскільки ймовірно, що система буде доступна користувачеві в певний момент часу. Вона може бути виражена як очікуваний відсоток успішних запитів, але ви також можете визначити її як відсоток часу, протягом якого система доступна для роботи протягом певного періоду часу. Наприклад, система може бути доступною 98% часу протягом місяця. Доступність є, мабуть, найбільш важливою вимогою для бізнесу, але щоб визначити її, ви також повинні мати оцінки надійності та ремонтопридатності.

Безпека

Безпека - це нефункціональна вимога, яка гарантує, що всі дані всередині системи або її частини будуть захищені від атак шкідливих програм або несанкціонованого доступу. Але тут є підступ. Левову частку нефункціональних вимог безпеки можна перевести в конкретні функціональні аналоги. Якщо ви хочете захистити адмін-панель від несанкціонованого доступу, ви визначите потік входів і різні ролі користувачів як поведінку системи або дії користувачів.

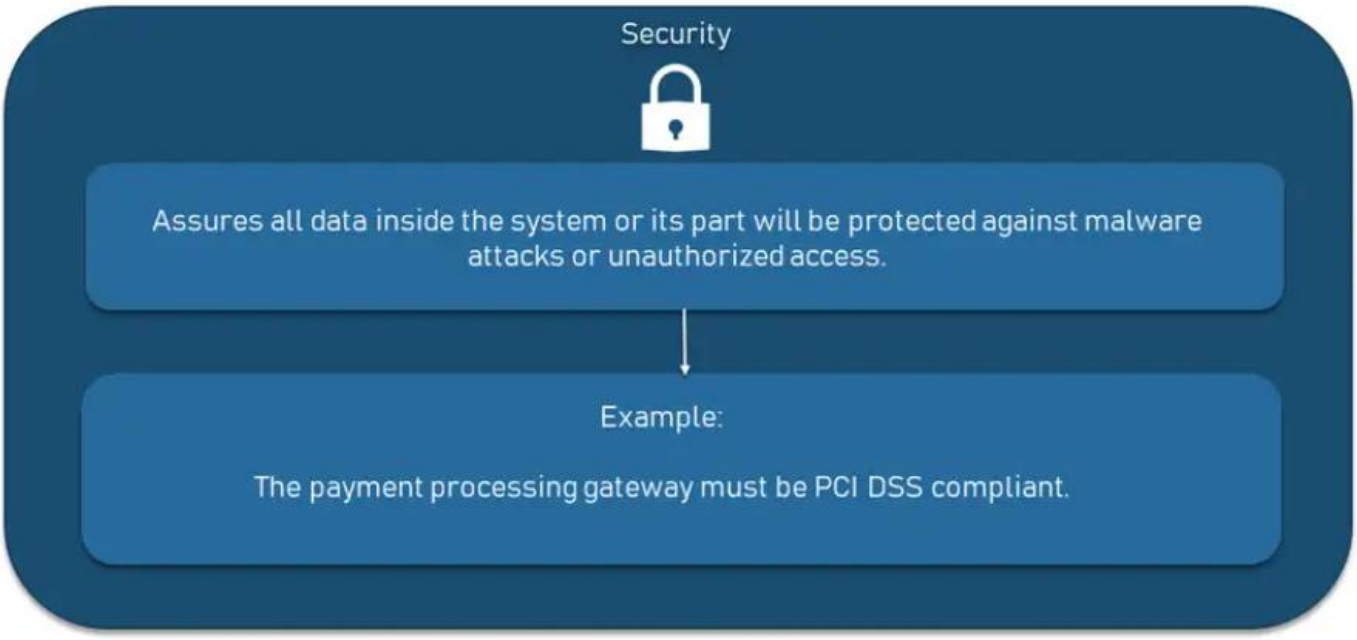

Рис.2.5 Безпека

Отже, частина нефункціональних вимог визначає конкретні типи загроз, які більш детально розглядаються у функціональних вимогах. Але це не завжди так. Якщо ваша безпека залежить від певних стандартів і методів шифрування, ці стандарти не описують безпосередньо поведінку системи, а скоріше допомагають інженерам з інструкціями з реалізації.

Локалізація

Атрибут локалізації визначає, наскільки добре система або її елемент відповідає контексту місцевого ринку. Контекст включає місцеві мови, закони, валюту, культуру, правопис та інші аспекти. Чим більше продукт відповідає цьому контексту, тим більший успіх він матиме серед певної цільової аудиторії.

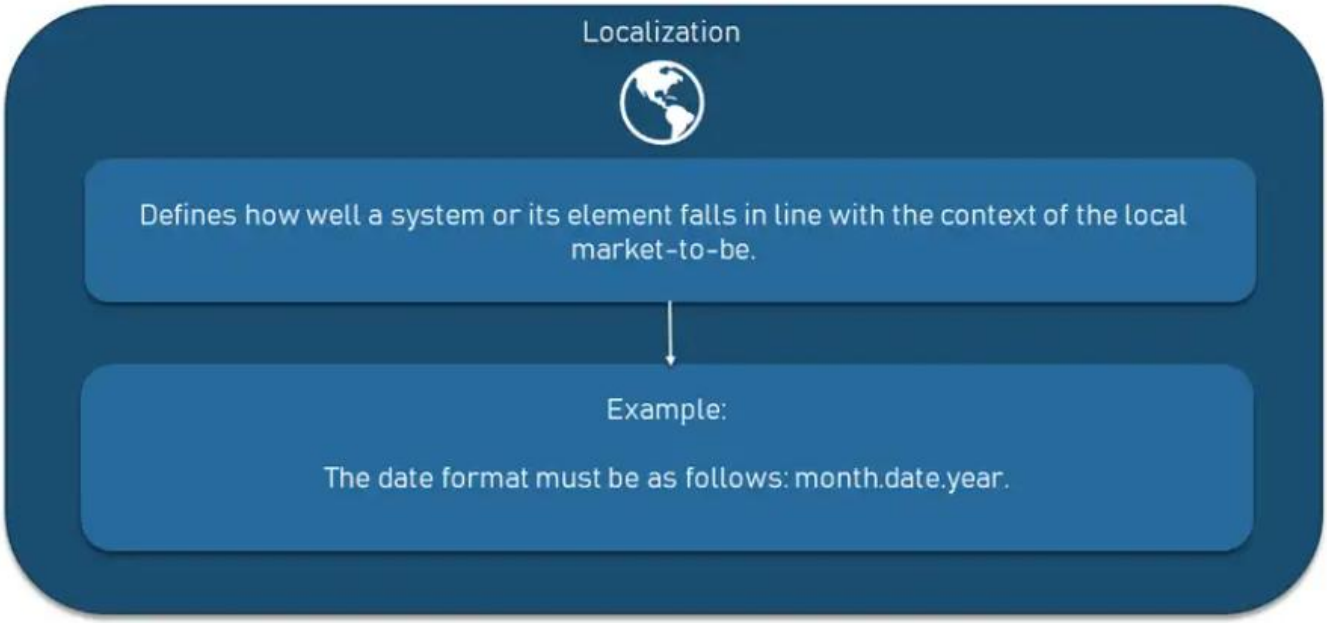

Рис.2.6 Локалізація

Юзабіліті

Юзабіліті - це ще одна класична нефункціональна вимога, яка відповідає на просте запитання: Наскільки складно користуватися продуктом? Визначити ці вимоги не так просто, як здається.

Рис.2.7 Юзабіліті

Існує багато видів критеріїв юзабіліті. Один з найпопулярніших - від Nielsen Norman Group, який пропонує оцінювати юзабіліті за п'ятьма параметрами:

- Здатність до навчання. Як швидко користувачі можуть виконати основні дії після того, як побачили інтерфейс?
- Ефективність. Як швидко користувачі можуть досягти своїх цілей?
- Запам'ятовуваність. Чи можуть користувачі повернутися до інтерфейсу через деякий час і відразу почати ефективно працювати з ним?
- Помилки. Як часто користувачі роблять помилки?
- Задоволеність. Чи приємно користуватися дизайном?

Апаратні вимоги яким повинен відповідати застосунок по пошуку та публікуванню інформації студентами:

- Операційна система яка підтримується: Linux/MacOS/Windows
- Версія браузеру: будь яка версія браузерів Google Chrome, Microsoft Edge , Mozilla Firefox, Opera чи Safari.
- Швидкість інтернет-підключення: не менше за 500 Кбіт/с.
- Географічне розташування юзерів: не має значення.

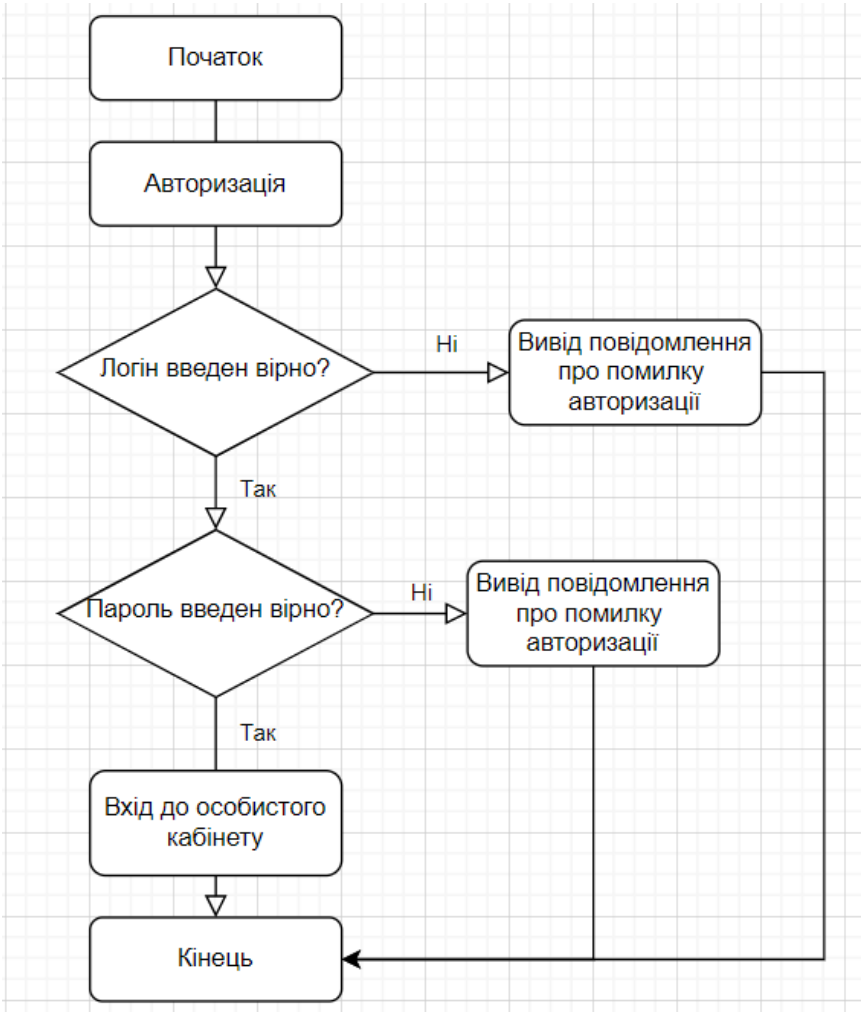

Рис.2.8 Блок схема авторизації

Для того щоб додаток був написаний з правильного архітектурного погляду та був масштабованим в майбутньому потрібно мати гарно сформовані програмні вимоги до системи.

Програмні вимоги яким повинен відповідати застосунок по пошуку та публікуванню інформації студентами:

- Веб-додаток написаний на мові C#
- Використання ASP.NET Core фреймворку
- Розробка в середовищі Rider
- В якості бази даних використати MSSQL
- Взаємодія з БД повинна відбуватися за допомогою підходу code-first та фреймворку Entity Framework Core

- Для швидкої взаємодії з користувачем використати розмітку Razor від Microsoft
- Для адаптивності інтерфейсу використати Bootstrap який надає пакет можливостей для розробки гарних та адаптивних дизайнів

Задля зручності використання додатку користувачами важливо розробити вимоги інтерфейсу які будуть гарантувати те що юзеру буде комфортно працювати з системою.

Вимоги до інтерфейсу яким повинен відповідати застосунок по пошуку та публікуванню інформації студентами:

- Основні графічні компоненти кнопки, таблиці, текст, малюнки
- Зручність користування додатком
- Всі поля обґрунтовані та логічно пояснені
- Кольори комбінуються між собою та сприяють комфортному користування програмою
- Взаємодія відбувається за допомогою мишки та курсору
- Всі функції додатку інтуїтивно зрозумілі

Операційні вимоги яким повинен відповідати застосунок по пошуку та публікуванню інформації студентами:

- Масштабованість – в майбутньому при збільшенні функцій і кількості користувачів додаток потрібно буде масштабувати, розширення функціональних можливостей повинно не стати проблемою, тому що в архітектуру додатку закладено всю необхідну інфраструктуру.
- Продуктивність – додаток розраховано на взаємодію з студентами яких в майбутньому може бути тисячі і навіть мільйони. Тому важливо розраховувати на велику кількість користувачів і закладати таку архітектуру яка впорається з навантаженням.

- Безпека та конфіденційність – всі дані якими оперує система повинні бути конфіденційними та захищеними. Всі особисті дані повинні міститися в зашифрованому вигляді використовуючи двухстороннє шифрування.
- Надійність – додаток повинен бути надійним та забезпечувати відказостійкість за допомогою кластерів серверів.
- Зручність – додаток повинен бути інтуїтивно зрозумілим та зручним у використання для того щоб користувач без досвіду міг спокійно з ним працювати.
- Цілісність даних – додаток має гарантувати цілісність даних для юзерів.
- Збереження даних – дані обов'язково повинні бути збережені при втраті відповідей від серверів, для цього використовувати бекапи які регулярно будуть робитися з бази даних та зберігатися на сторонніх серверах. В разі втрати даних їх можна буде відновити за допомогою бекапів.

Було сформовано всі функціональні та нефункціональні вимоги до застосунку опублікування та знаходження інформації студентами. Якщо додаток буде відповідати всім вище наведеним вимогам він буде успішно задовільняти всіх користувачів не залежно від їх кількості.

Висновок до розділу 2

Отже, було досліджено основні вимоги до застосунку опублікування та знаходження інформації студентами.

Сформовано всі типи вимог до системи включаючи нефункціональні, функціональні, апаратні, операційні, вимоги до інтерфейсу, вимоги користувача та інше.

Якщо підсумувати, то всі вимоги буде гарно продумані та сформовані виходячи з потреб користувачів а саме студентів. Все це дозволить розробити та спроектувати дійсно гарний продукт який допоможе тисячам студентів отримувати та ділитися інформацією в відкритому доступі.

РОЗДІЛ 3
СТРУКТУРА ПРОГРАМНОГО ЗАСТОСУНКУ ДЛЯ ПОШУКУ ТА ОПУБЛІКУВАННЯ ІНФОРМАЦІЇ СТУДЕНТАМИ

3.1. Структура програмного застосунку

Для даного типу додатку логічним рішенням буде використати класичну трирівневу архітектуру тому що додаток має не великі розміри але в майбутньому можливе масштабування системи та розширення функціональних можливостей. Такий тип архітектури забезпечує гнучкість розширення та логічне розподілення функціоналу на рівні.

Логіка в даному типі архітектури розподілена на 3 рівні:

- Рівень доступу до даних
- Рівень логіки
- Рівень інтерфейсу додатку

Рівень доступу до даних працює з даними та відповідає за взаємодію з ними. Вся робота з базою даних відбувається на цьому рівні і ізольована від інших що забезпечує цілісність даних та логічне розподілення ролей. Тут зазвичай знаходяться контекст БД та моделі які являють собою таблички в базі даних.

Рівень логіки містить в собі логіку додатку, всі складні вичислення та всі логічні операції. Зазвичай тут розташовані різні провайдери, колектори, репозиторії та інші сутності які відносяться до роботи з логікою застосунку.

Рівень інтерфейсу це найвищий рівень яких представляє собою всю візуальну частину та взаємодію з нею. Користувач бачить та користується тільки цим рівнем.

Всі рівні між собою комунікувати не можуть, кожен рівень може звернутися тільки до рівня першого зв’язку. Це означає що рівень інтерфейсу ніколи не може напряму звернутися до рівня доступу до даних і навпаки.

Завдяки такому розподіленню на рівні ця архітектура забезпечує гнучкість та структурованість коду та в майбутньому легке масштабування функціональних можливостей.

Для демонстрування цієї архітектури гарно підходить архітектурний шаблон Model-View-Controller (MVC) (рис. 3.1).

Цей шаблон є доволі популярним для розробки невеликих додатків, забезпечує гарну взаємодію та демонстрацію роботи системи з середини. Він розділяє додаток на три логічні рівні:

Рівень моделі(Model) – на цьому рівні відбувається взаємодія з базою даних, зберігаються моделі, формується контекст БД та будуються взаємодії між таблицями які потім створюються в базі даних.

Рівень інтерфейсу(View) – відповідає за взаємодію з користувачем та містить всю структуру інтерфейсу, клієнтську частину застосунку, стилі, розмітку та обробники подій.

Рівень контролеру(Controller) – є логічним рівнем застосунку який містить всю бізнес логіку, бізнес процеси та містить контролери які за допомогою API взаємодіють з рівнем інтерфейсу.

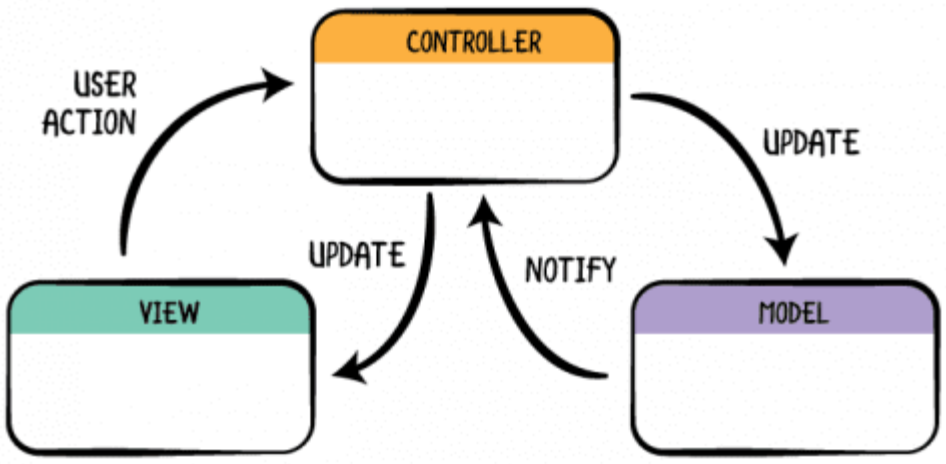

Рис.3.1 Взаємодія компонентів шаблону MVC

Цей архітектурних шаблон було використано при розробці застосунку по пошуку та публікування інформації студентами через його гнучкість та адаптивність.

На першому рівні(Model) зберігаються моделі які використовуються в додатку, їх опис, структура та контекст бази даних для взаємодії з нею(рис. 3.2)

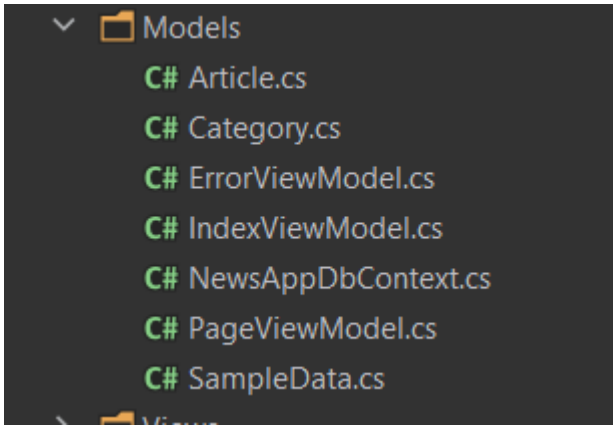

Рис.3.2 Рівень моделі(Model)

Наступні модельки реалізовані на рівні Model:

ErrorViewModel – ця модель потрібна для відображення помилок. Тут зберігається ідентифікатор запиту, за допомогою якого можна отримати помилку та повернутий код помилки. І користувачі, і розробники можуть використовувати цю інформацію, щоб отримати детальну інформацію про причину помилки.

PageViewModel — це модель, яка відповідає за сторінку. Він містить сторінку та кількість елементів, які відображатимуться на цій сторінці.

SampleData — це модель, яка використовується для заповнення даних у програмах. Коли додаток розгортається, на сервері згортається порожня база даних, яка не містить тестових даних і необхідних даних про населення. Модель дозволяє заповнювати порожню базу даних після її створення даними, які міститиме модель.

Article – модель зберігає всю необхідну інформацію про опубліковану статтю, включаючи текст, заголовок, дату публікації, ім’я автора, короткий опис та має зв’язок з категоріями для фільтрування і відображення на правильній сторінці.

Category – містить інформацію про категорії публікацій, слугує для фільтрування та відображення статей по категоріям, категорії легко розширюються додаванням нового запису до таблички.

IndexViewModel – це додаткова модель яка в собі зберігає фільтри пошуку, пагінацію та фільтри по публікаціям.

NewsAppDbContext – контекст БД, який потрібен для взаємодії з таблиця в базі даних, містить всі таблички та відповідає за їх створення. При проектуванні застосунку було використано підхід code-first, який формує таблички в базі даних на основі моделей. Тож в даному контексті бази даних знаходиться вся взаємодія з БД (див. Лістинг 3.1.)

Лістинг 3.1

Лістинг контексту бази даних

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ASP.NET_Core_NewsApp.Models
{
    public class NewsAppDbContext : DbContext
    {
        public DbSet<Article> Articles { get; set; }
        public DbSet<Category> Categories { get; set; }

        public NewsAppDbContext(DbContextOptions<NewsAppDbContext>
options)
            : base(options)
        {
            Database.EnsureCreated();
        }
```

```
        }
    }}
```

На наступному рівні інтерфейсу(View) зберігається вся взаємодія з інтерфейсом користувача включаючи розмітку, сторінки та інше. (рис.3.3)

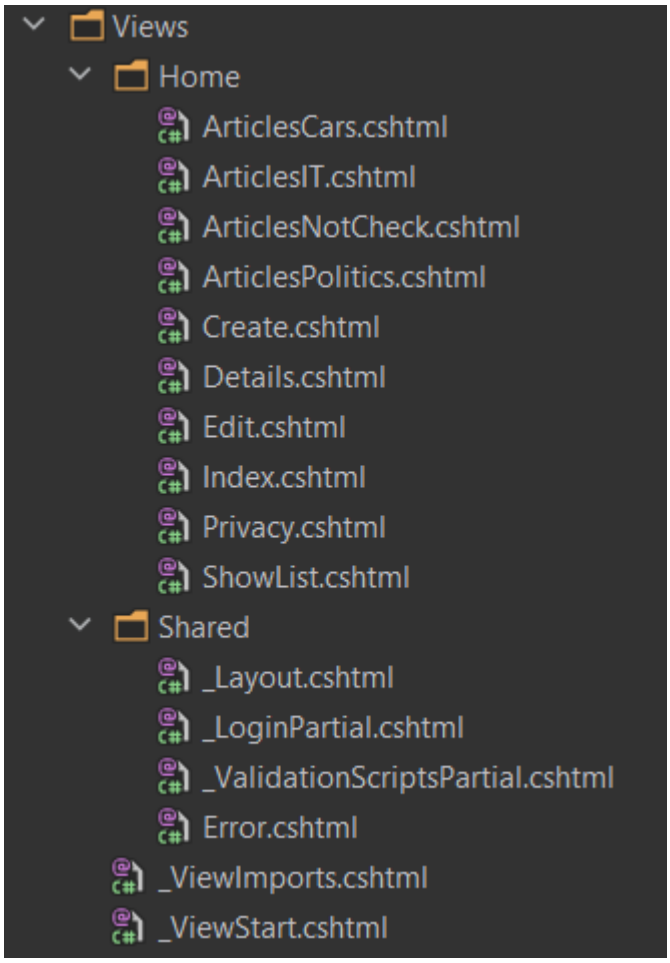

Рис.3.3 Рівень інтерфейсу(View)

Тут можна побачити такі представлення як:

Articles – це представлення які відповідають за відображення статей по різним категоріям. Містять розмітку за взаємодію з інтерфейсом саме публікацій.

Create – це представлення яке відображає сторінку додавання нової публікації та заповнення всієї необхідної інформації про неї включаючи назву, опис, категорію, зображення та інше.

Details – містить сторінку детального перегляду публікації, де відображається повний текст, зображення, ім’я автора, заголовок.

Edit – відповідає за сторінку редагування інформації про публікацію, дає можливість зміни даних та актуалізацію після публікації.

Index – головна сторінка додатку, показує всі публікації без фільтрів та пагінацію по сторінкам публікацій.

Privacy – допоміжна сторінка яка містить загальні інформацію про додаток або його авторів.

Лістинг 3.2

Лістинг одного з представлень

```
@using ASP.NET_Core_NewsApp.Models
@model IndexViewModel

@{
    ViewData["Title"] = "Index";
}

<style>
    @@font-face {
        font-family: 'FontAwesome';
        src: url('https://maxcdn.bootstrapcdn.com/font-
awesome/4.7.0/fonts/fontawesome-webfont.woff2') format('woff2'),
url('https://maxcdn.bootstrapcdn.com/font-
awesome/4.7.0/fonts/fontawesome-webfont.woff') format('woff'),
url('https://maxcdn.bootstrapcdn.com/font-
awesome/4.4.0/fonts/fontawesome-webfont.ttf') format('truetype');
        font-weight: normal;
        font-style: normal
    }

    .glyphicon {
```

```
            display: inline-block;
            font: normal normal normal 14px/1 FontAwesome;
            font-size: inherit;
            text-rendering: auto;
            -webkit-font-smoothing: antialiased;
            -moz-osx-font-smoothing: grayscale
        }

        .glyphicon-chevron-right:before {
            content: "\f054";
        }

        .glyphicon-chevron-left:before {
            content: "\f053";
        }
    </style>

    <div class="row">
        <div class="col-md-12" style="float:right;">
            <form asp-action="Index">
                <div asp-validation-summary="ModelOnly" class="text-
danger"></div>
                <div class="form-group">
                    <input asp-for="Search" class="form-control col-md-
12" placeholder="Поиск статей по сайту..." />
                    <span asp-validation-for="Search" class="text-
danger"></span>
                </div>
                <div class="form-group">
                    <input type="submit" value="Поиск" class="btn btn-
dark col-md-12" />
                </div>
            </form>
```

```
        </div>
    </div>

    <center><h3><i>Все статьи</i></h3></center>
    <br />
    <div class="row">
        @foreach (var item in Model.Articles)
        {
            <div class="col-lg-4 col-md-6 mb-4">
                <div class="card h-100" style="width: 22rem;">
                    <img src="@item.Img" class="card-img-top" alt="...">
                    <div class="card-body">
                        <h5 class="card-title">@item.Title</h5>
                        <p class="card-text">@item.ShortDesc</p>
                        @Html.ActionLink("Подробнее", "Details", new {
id = item.Id }, new { @class = "btn btn-dark" })
                    </div>
                    <div class="card-footer">
                        <small class="text-muted">Дата добавления
статьи:<br />@item.DateTime</small>
                    </div>
                </div>
            </div>
        }
    </div>

    @if (Model.PageViewModel.HasPreviousPage)
    {
        <a asp-action="Index"
           asp-route-page="@(Model.PageViewModel.PageNumber - 1)"
           class="btn btn-outline-dark">
            <i class="glyphicon glyphicon-chevron-left"></i>
            Назад
```

```
        </a>
    }
    @if (Model.PageViewModel.HasNextPage)
    {
        <a asp-action="Index"
           asp-route-page="@(Model.PageViewModel.PageNumber + 1)"
           class="btn btn-outline-dark">
            Вперед
            <i class="glyphicon glyphicon-chevron-right"></i>
        </a>
    }
```

Останній рівень це рівень контролеру(Controller), який містить загальну логіку додатку.

На останньому рівні контролеру(Controller) зберігається вся бізнес логіка додатку яка включає в себе взаємодію з публікаціями та категоріями публікацій. Сюди входять контролери(рис 3.4) та репозиторії(рис 3.5) які відповідають за доступ до даних.

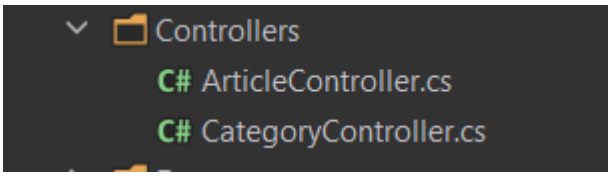

Рис.3.4 Контролери додатку

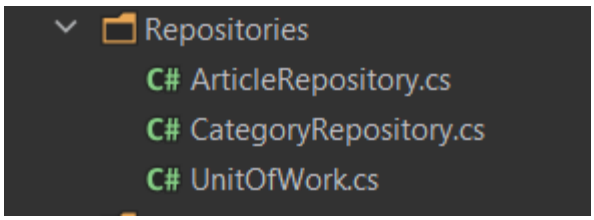

Рис.3.5 Репозиторії додатку

Контролери взаємодіють з представленнями та з рівнем доступу до даних. На кожен контролер є своя сторінка представлення яка взаємодіє з ними. В свою чергу контролер взаємодіє з модельками для зберігання змін або діставання даних з БД. Контролер слугує проміжним шаром між рівнем інтерфейсу та рівнем доступу до даних.

Репозиторії взаємодіють з контекстом бази даних напряму, дістаючи дані або змінюючи їх. Контролери не повинні містити логіки, вони повинні викликати репозиторії для взаємодії з даними. Вони можуть містити тільки виклики інших сутностей і не покриваються тестами. Це необхідно для відокремлення логіки взаємодії з БД. При цьому код стає структурованим, підтримуваним та відповідає гарним практикам написання коду.

Також гарною практикою є використання шаблону проектування під назвою UnitOfWork. Цей шаблон дає можливість побудувати слабо зв'язний код який в свою чергу буде краще підтримуватся та розширюватися в майбутньому.

Принцип UnitOfWork полягає у створенні класу який інкапсулює в собі всі виклики репозиторіїв для єдиної структури та взаємодії з ними. Контролери не мають використовувати напряму репозиторії, замість цього варто використовувати UnitOfWork для виклику методів репозиторіїв.

Ще одною гарною практикою є використання DI для того щоб не прив'язувати код до реалізації а прив'язувати до абстракції. За допомогою цього код стає більш зручним та гнучким. Насправді використання конкретної реалізації нам необхідно в дуже обмежених цілях таких як написання юніт тестів. Завжди краще використати Dependency Injection для використання сутності. Для цих цілей можна скористатися як стандартними можливостями .Net Core платформи так і сторонніми бібліотеками(наприклад Ninject).

Перевагу UnitOfWork та DI краще всього показати на прикладі. Давайте уявімо такий сценарій: під час розробки програми клієнт каже нам використовувати базу даних NoSQL MongoDB, оскільки її розробка займає менше часу, ніж впровадження з'єднань у реляційній базі даних. Ми погоджуємося та розробляємо додаток з використанням цієї бази даних, але в якийсь момент обсяг даних починає масово зростати, і підтримувати таку базу даних стає дуже важко, сервер не витримує. Отже приймається рішення щодо переходу на нову реляційну базу даних, наприклад MSSQL.

1) Що нам робити в цьому випадку, якщо ми не використовуємо DI та шаблон проектування UnitOfWork?!

У цьому випадку необхідно буде втрутитися і переписати всі місця, де ми використовуємо репозиторій, що займе багато часу і призведе до появи багатьох помилок.

2) Якщо ми використовуємо DI та шаблон UnitOfWork, що нам робити в цьому випадку?!

Напишіть новий UnitOfWork, який реалізує інтерфейс, а потім просто використовуйте DI, щоб повторно прив'язати новий клас до інтерфейсу.

Переваги видно не озброєним оком, в першому випадку ми мали багато проблем і втратили багато часу та бюджету клієнта. При цьому в другому випадку ми працюємо з невеликим, що не повинно викликати проблем.

Лістинг 3.3

Лістинг репозиторію

```
using System.Linq;
using ASP.NET_Core_NewsApp.Interfaces;
using ASP.NET_Core_NewsApp.Models;

namespace ASP.NET_Core_NewsApp.Repositories
{
    public class ArticleRepository : IRepository<Article>
    {
        NewsAppDbContext db;

        public ArticleRepository(NewsAppDbContext context)
        {
            db = context;
        }
        public IQueryable<Article> GetAll()
```

```
        {
            return db.Articles;
        }

        public Article Get(int? id)
        {
            return db.Articles.Find(id);
        }

        public void Create(Article item)
        {
            db.Articles.Add(item);
        }

        public void Update(Article item)
        {
            db.Articles.Update(item);
        }

        public void Delete(int id)
        {
            var item = db.Articles.Find(id);
            if (item != null)
            {
                db.Articles.Remove(item);
            }
        }
    }
}
```

3.2. Діаграми класів та логіки програмного застосунку

Найчастіше діаграма UML використовується для аналізу існуючого програмного забезпечення, моделювання нового програмного забезпечення, планування розробки програмного забезпечення та встановлення пріоритетів. Простіше кажучи, якщо вам потрібен спосіб візуалізації та планування процесу розробки програмного забезпечення, діаграма UML неймовірно допоможе.

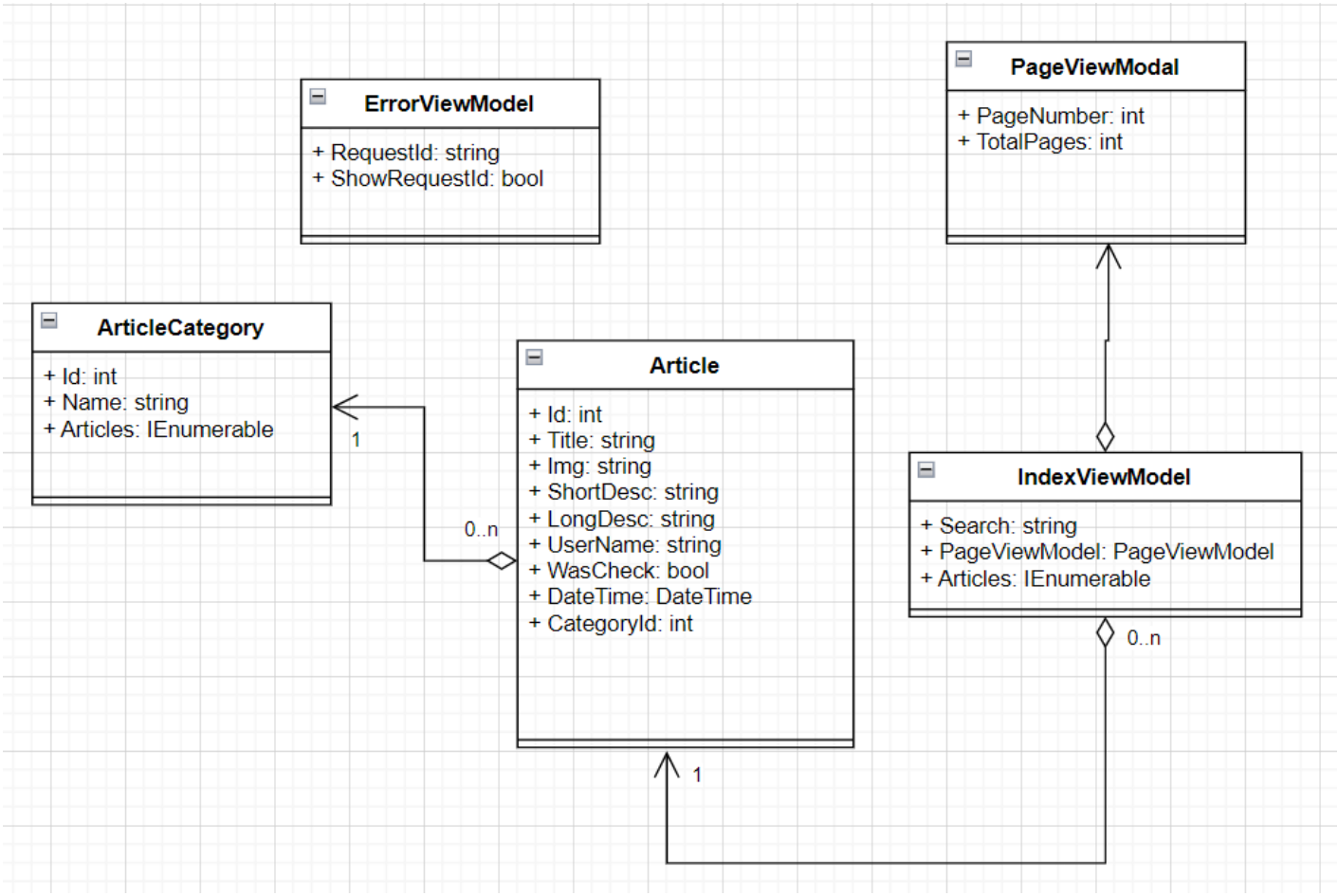

Рис.3.6 Діаграма класів для моделей

На наведеній діаграмі класів (рис.3.6) продемонстровано моделі які містить застосунок, їх взаємодію одна з одною, поля та їх типи. Головною моделлю являється Acticle, вона зберігає дані про публікації такі як: ідентифікатор, заголовок, шлях до зображення, короткий опис, повний текст, ім’я автора та дату публікації. Так як це основна модель вся логіка будується навколо неї. Допоміжна модель Category

відповідає за категорії публікацій та пов’язана з головною моделлю зв’язком один до багатьох. Інші моделі необхідні для пагінації, пошуку та фільтрації публікацій.

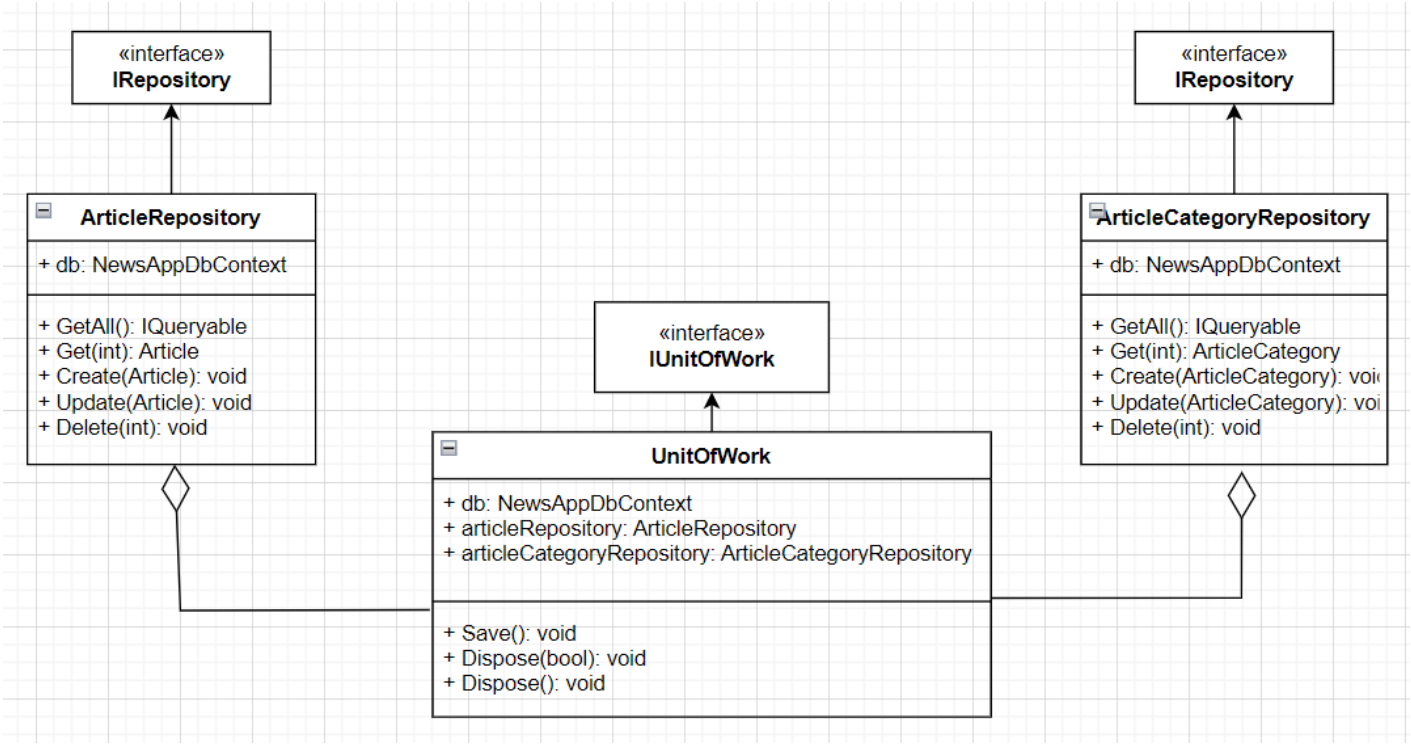

Рис.3.7 Діаграма класів репозиторіїв

На наведеній вище діаграмі класів для репозиторіїв (рис.3.7) продемонстровано взаємодія між репозиторіями та класом UnitOfWork. Репозиторії інкупсулюють основні CRUD операції з сутностями такі як:

- Отримання даних
- Створення даних
- Оновлення даних
- Видалення даних

UnitOfWork зв'язує всі репозиторії в єдине ціле та презентує інтерфейс взаємодії з репозиторіями що дозволяє працювати через єдиний клас.

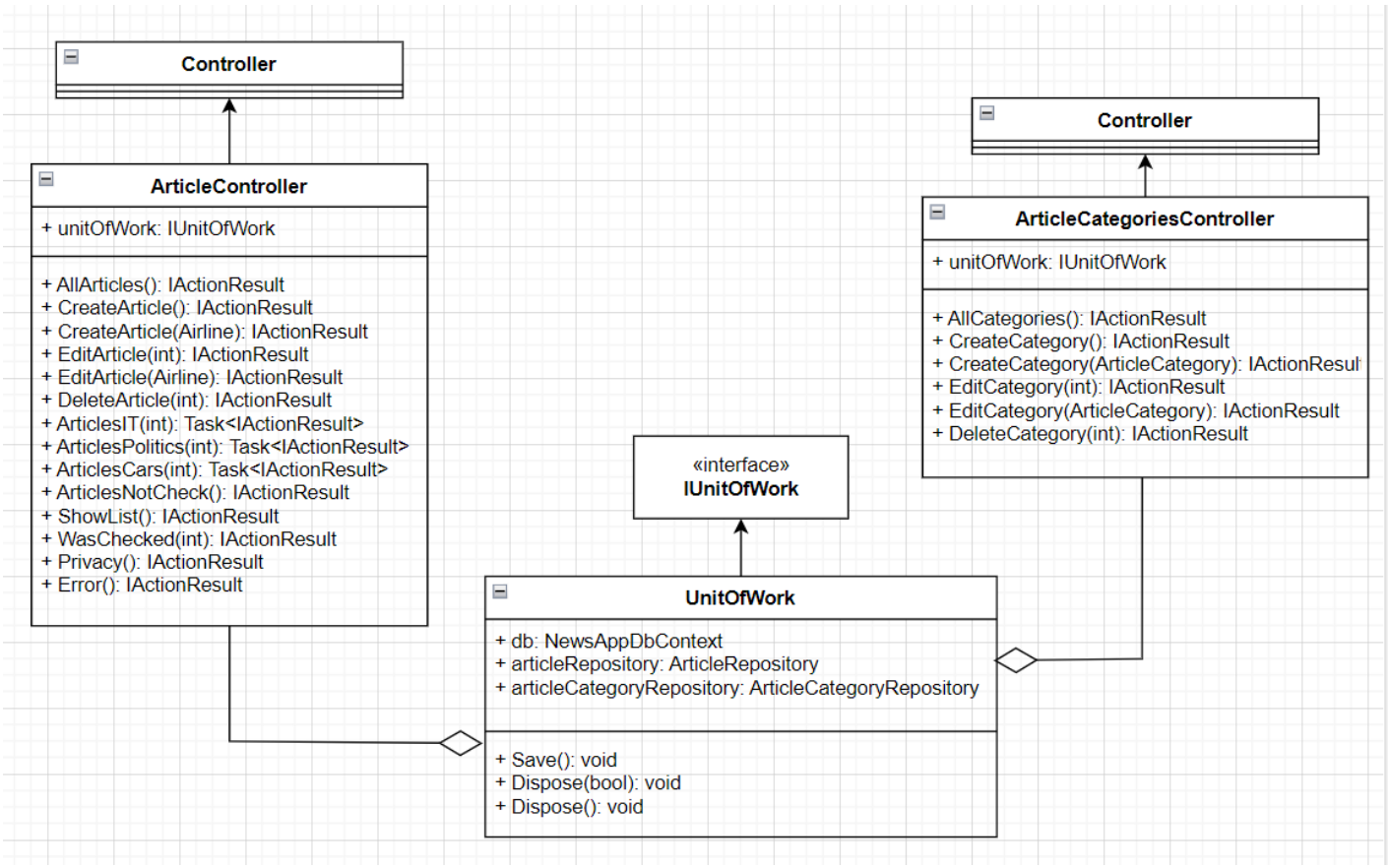

Рис.3.8 Діаграма класів для контролерів

В наступній діаграмі класів(рис.3.8) наведено взаємодію контролерів між собою та які сутності вони використовують.

Всі контролери повинні бути наслідувані від базового класу .NET Core платформи під назвою Controller. Для того щоб взаємодіяти з даними та викликати репозиторії контролери використовують клас UnitOfWork через який викликають потрібні їм сутності.

Через UnitOfWork контролери звертаючись до репозиторіїв, оброблюють вхідні та вихідні дані та направляють їх до клієнтської частини застосунку. Завдяки такій взаємодії дані відображаються на UI.

Для демонстрації архітектури застосунку, взаємодії компонентів та розуміння загальної картини наведено діаграму розгортання(рис.3.9). Для додатку по пошуку та публікування інформації студентами ця діаграма матиме вигляд:

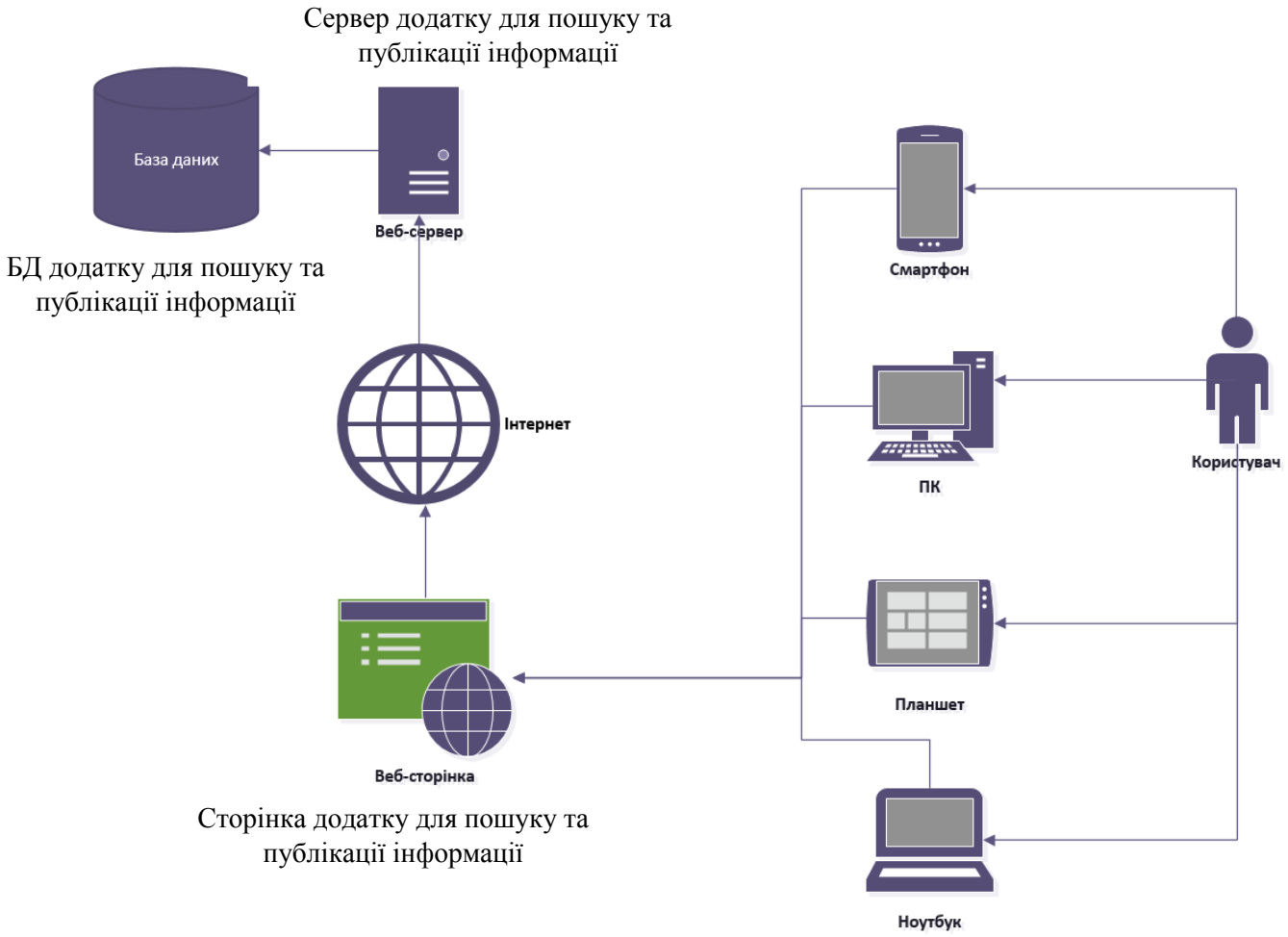

Рис.3.9 Діаграма розгортання

На даній діаграмі видно що користувач обирає будь який зручний пристрій з доступом в інтернет для взаємодії. Це може бути смартфон, ПК, планшет чи ноутбук. Взаємодія відбувається через веб-сторінку яку можна відкрити в будь-якому браузері. Після цього веб-сторінка відправляє запит до серверу використовуючи Інтернет. Сервер в свою чергу взаємодіє з базою даних для діставання, додавання або

редагування необхідних даних. Таким чином користувач бачить результат взаємодії на веб сторінці в браузері.

Для розуміння загального сценарію взаємодії між компонентами наведено діаграму послідовності(рис.3.10).

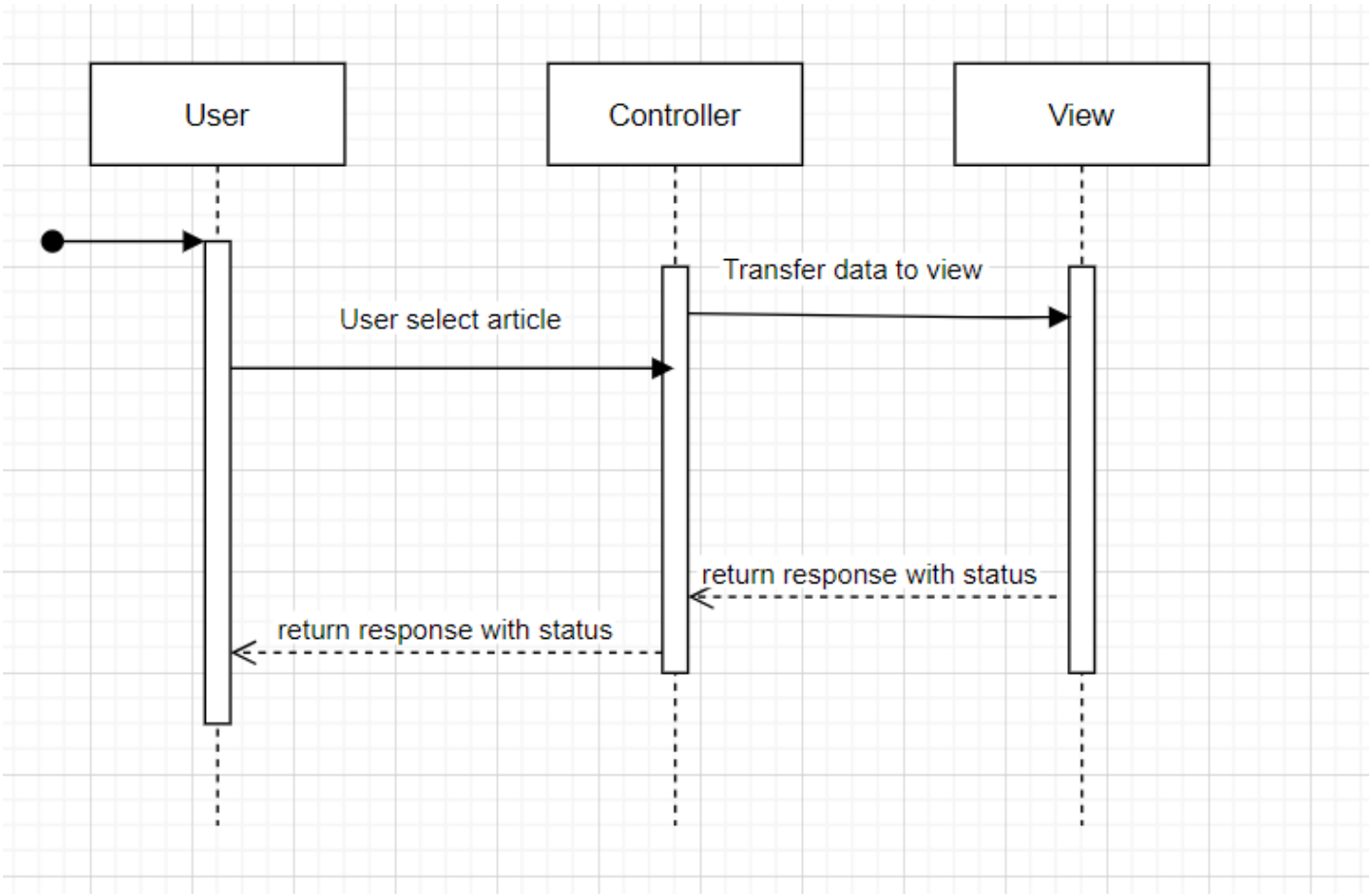

Рис.3.10 Діаграма послідовності

На цій діаграмі можна зрозуміти основний флоу взаємодії користувача з системою. В той час як користувач якимось чином взаємодіє зі застосунком відправляється запит на контролер, контролер обробляє дані за допомогою репозиторіїв та повертає дані на рівень інтерфейсу. Це може бути будь яка дія, наприклад обрати статтю та переглянути її детальніше. Таким чином користувач отримує або очікуваний результат або помилку в залежності від повернутого статус коду.

Для розуміння основного сценарію який очікується від системи при публікації нової публікації було наведено блок схем діаграму(рис. 3.11) яка демонструє поведінку системи в такому контексті.

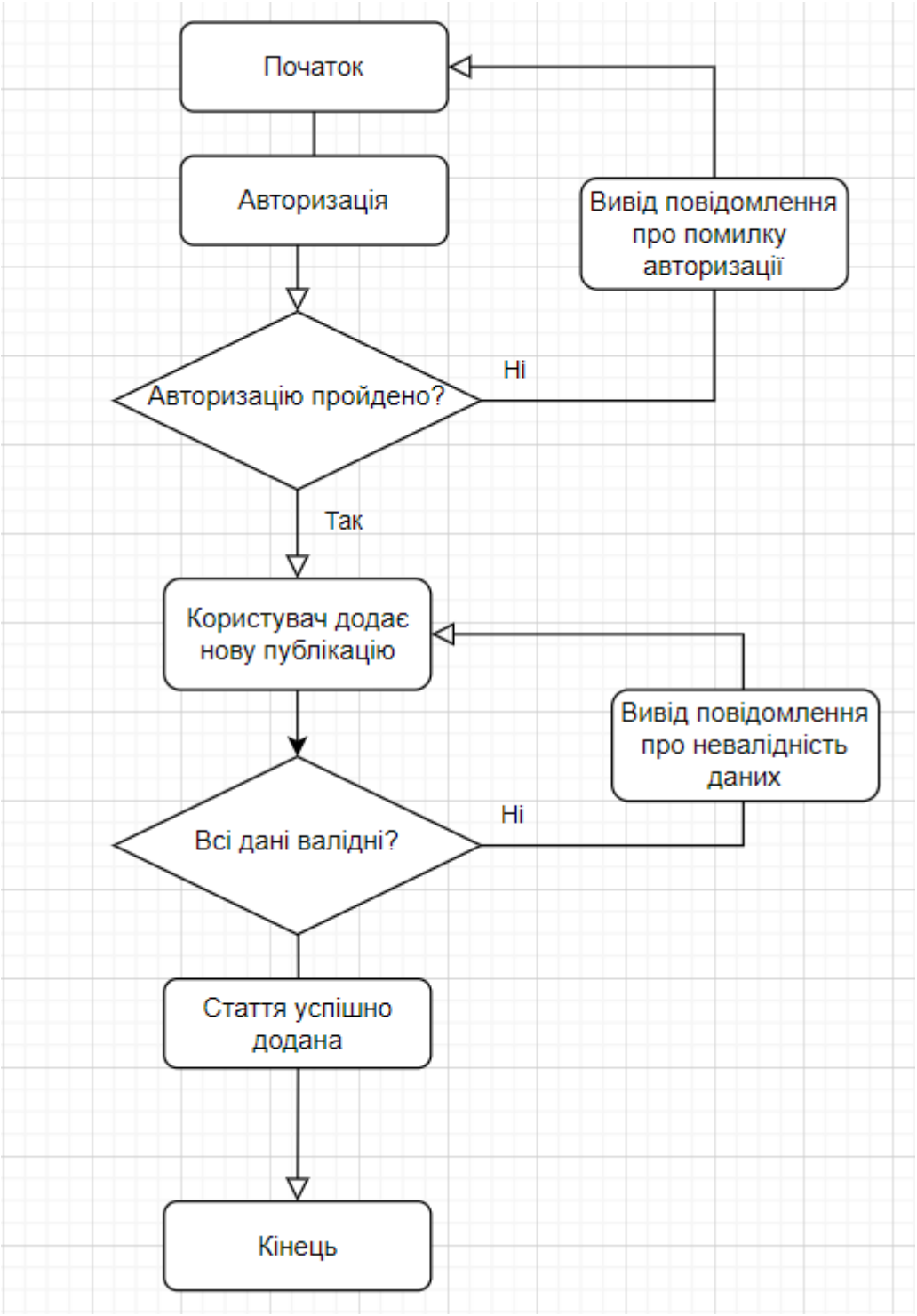

Рис.3.11 Блок схема дій при додаванні нової публікації

Для того щоб опублікувати статтю користувачу необхідно авторизуватися, без авторизації публікування не дозволено. Якщо авторизацію пройдено успішно користувач може додати нову публікацію. При цьому валідується правильність введення всіх даний. Якщо дані не вірні користувачу показується повідомлення про це. Якщо все добре стаття успішно відправляється на перевірку і публікацію.

Висновок до розділу 3

В даному розділі було продемонстровано всі модулі застосунку, як вони взаємодіють між собою та за що відповідають. Описано архітектуру та основні практики які були використані при розробці додатку. Представлено всі основні типи UML діаграм для демонстрації взаємодії компонентів, наведення основних флоу та пояснення основних сценаріїв. Кожен компонент системи детально розписаний та містить приклади лістингів коду.

РОЗДІЛ 4
ПРОТОТИП ПРОГРАМНОГО ЗАСТОСУНКУ ДЛЯ ПОШУКУ ТА ОПУБЛІКУВАННЯ ІНФОРМАЦІЇ СТУДЕНТАМИ

4.1. Модуль адміністратора

Якщо застосунок в собі містить дані які необхідно валідувати то в край необхідно розробити панель адміністратора яка дозволить редагувати, змінювати або додавати необхідні дані обмеженої кількості користувачів у яких є права адміністратора.

Застосунок для пошуку та опублікування інформації студентами розрахований на публікування різних типів статей всім зареєстрованим користувачам. Весь контент статей повинен валідуватися адміністратором системи який буде вирішувати чи варто публікувати дану інформацію чи ні. Для цих цілей розроблено модуль адміністратора який видно лише юзерам з відповідними правами, інші користувачі застосунку не можуть побачити цей модуль.

Панель адміністратора містить весь необхідний функціонал для детального перегляду, редагування, підтвердження або видалення публікації.

Для того щоб відкрився доступ до панелі адміністратора (рис.4.2) користувач повинен виконати вхід до облікового запису. Якщо роль користувача є “Адміністратор”, то для нього відкриються додаткові можливості такі як редагування та підтвердження статей, якщо у користувача немає “Адміністратор” прав то він не побачить ніяких привілеїв на сторінці.

Вхід до облікового запису має загальний вигляд(рис. 4.1). Користувачу необхідно ввести логін та пароль. Логіном слугує адреса електронної пошти. Якщо користувач не

пам’ятає пароль він може його поновити використовуючи електронну адресу яка вказана в логіні або ж зареєструвати новий обліковий запис.

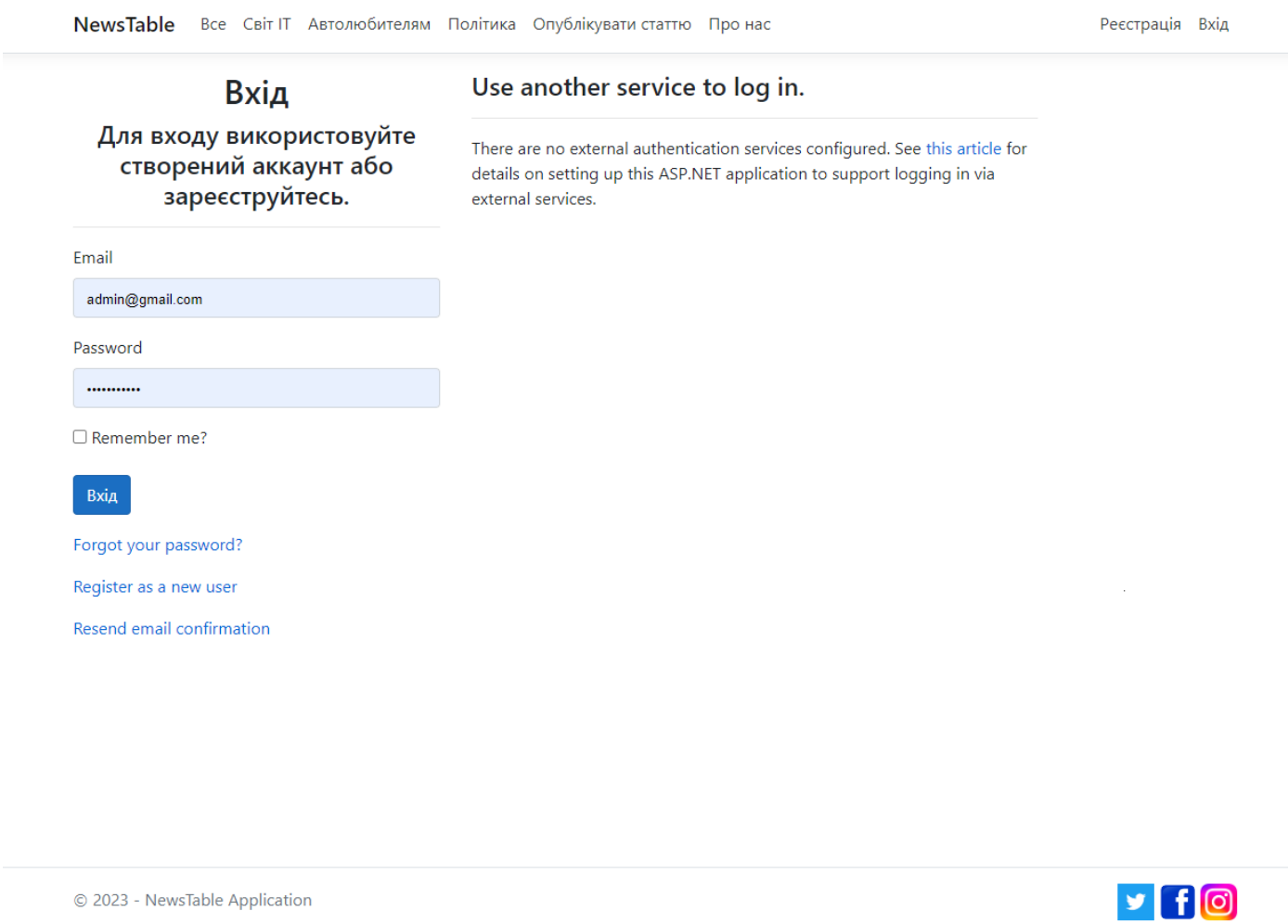

Рис.4.1 Вхід до облікового запису користувача

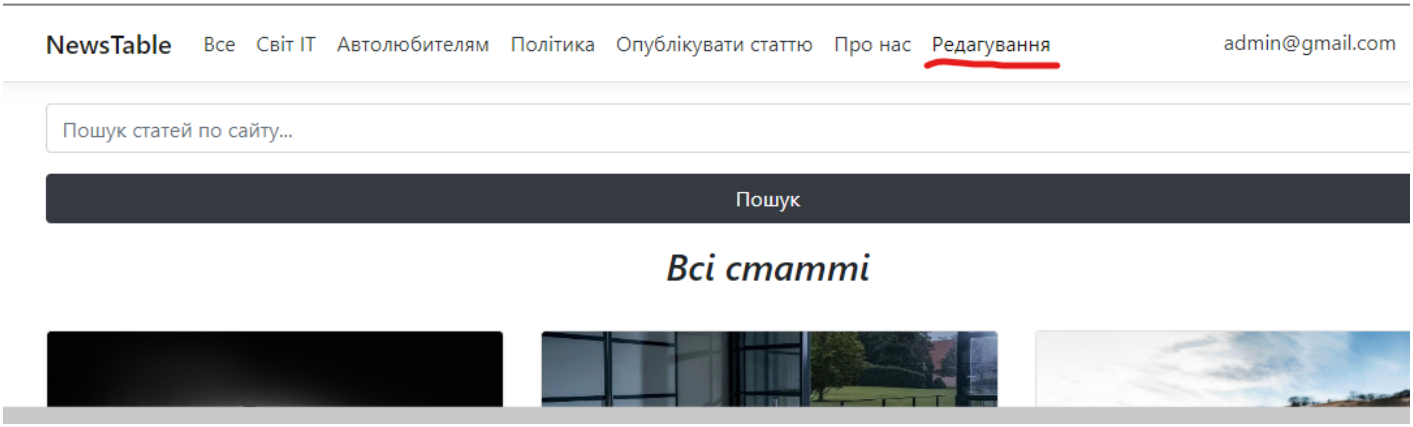

Рис.4.2 Панель адміністратора

На панелі адміністратора наявні всі основні функції якими володіє адміністратор для роботи з публікаціями. Зверху розташовані навігаційні кнопки для додавання або підтвердження статей (рис.4.3)

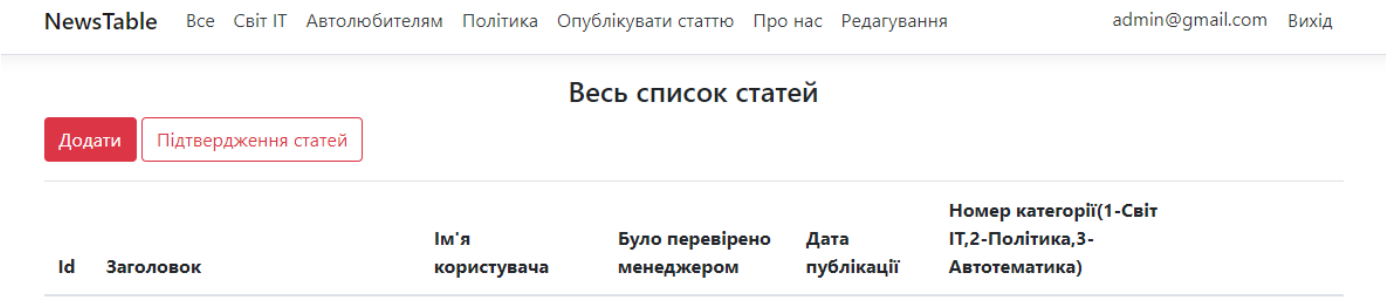

Рис.4.3 Кнопки для навігації

При натисненні на “Підтвердження статей” адміністратору відкривається сторінка де знаходяться всі статті які очікують підтвердження. Також з’являються додаткові кнопки “Підтвердити” та “Видалити”(Рис. 4.4).

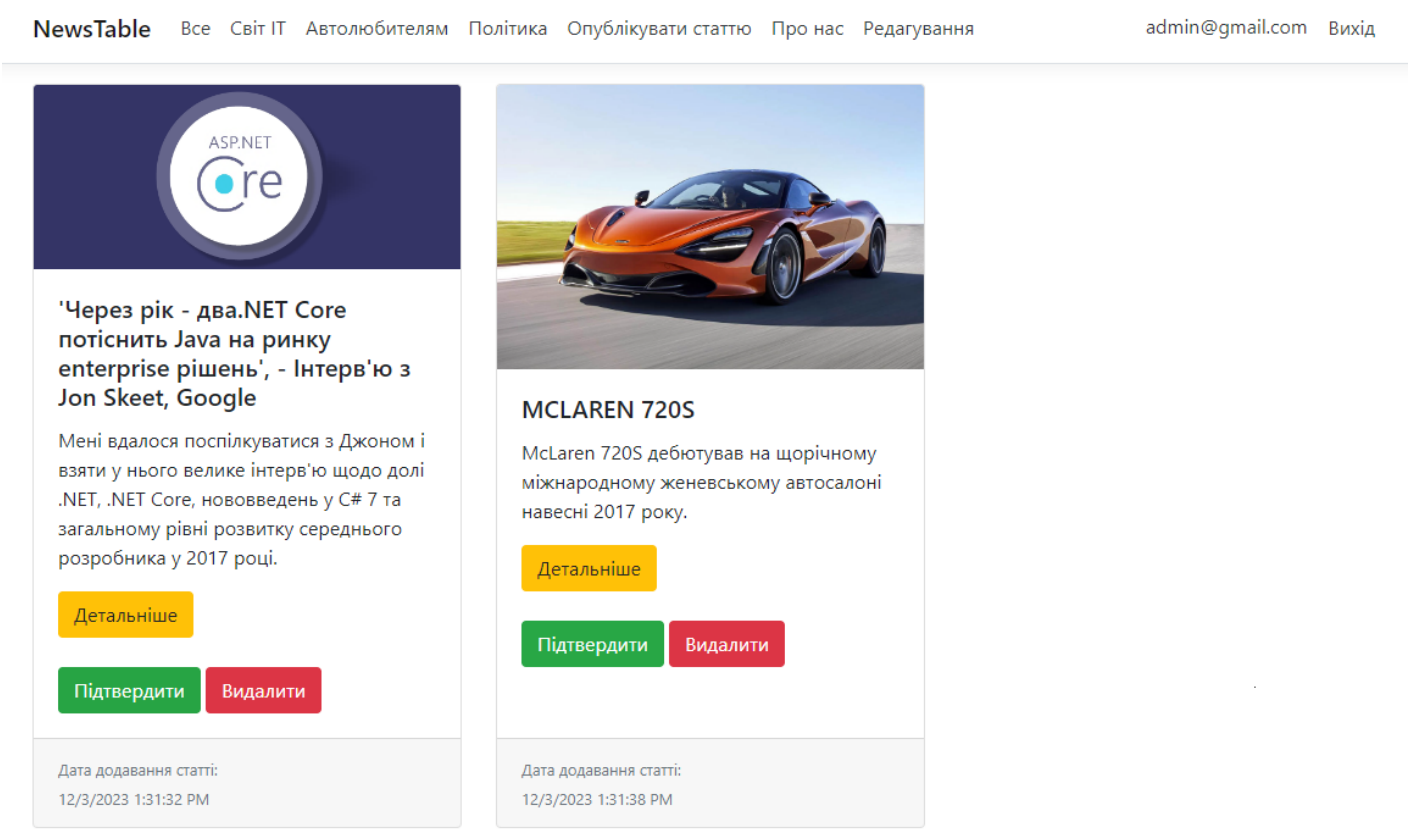

Рис.4.4 Підтвердження статей

Всі публікації відображаються у вигляді списку(рис. 4.5) та містять коротку інформацію про зміст, дату публікації, автора, категорії та чи була стаття перевірена адміністратором. Також є додаткові кнопки які дозволяють редагувати, переглядати детальну інформацію або видалити публікацію.

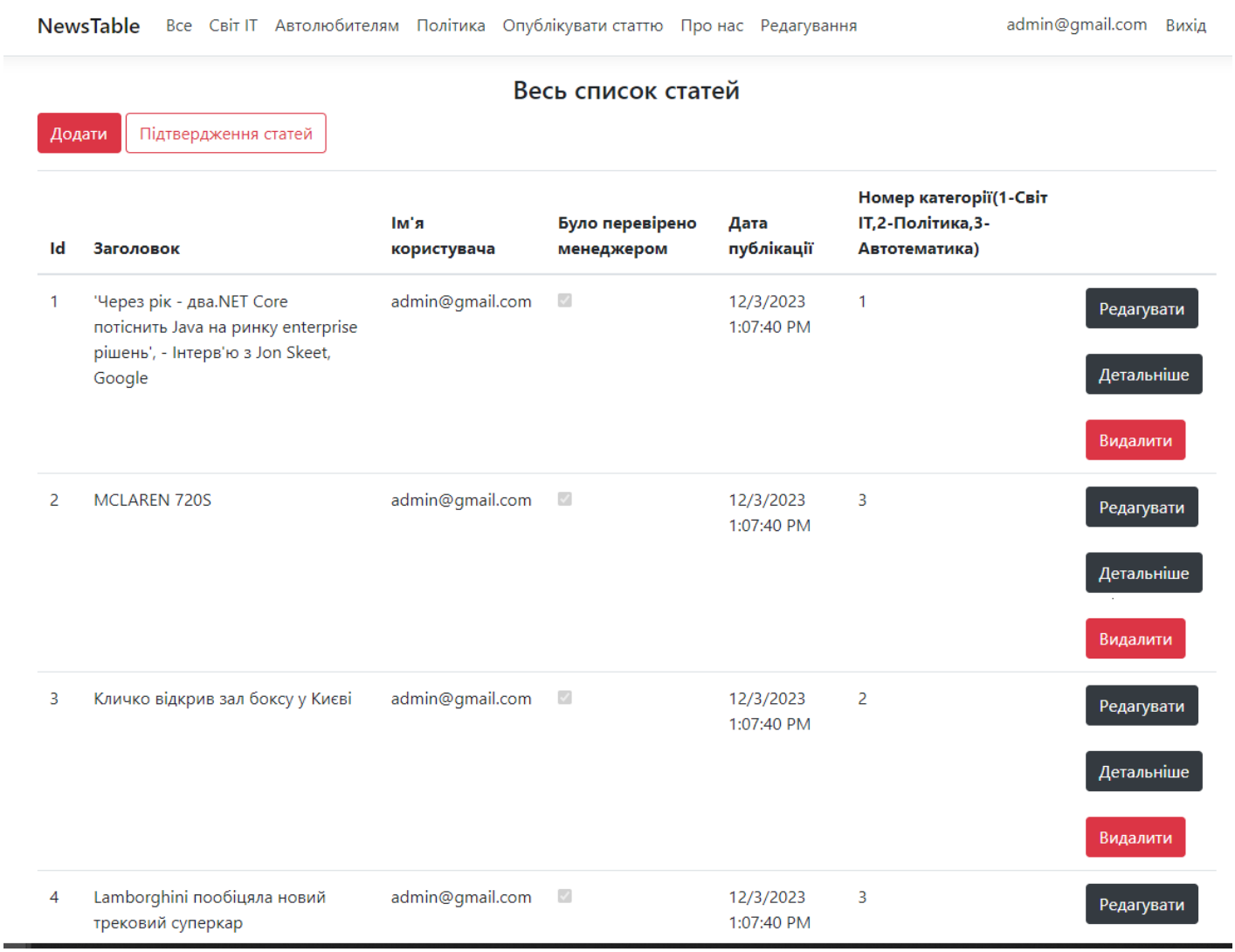

Рис.4.5 Список публікацій

При натисканні на кнопку “Детальніше” відкривається детальна інформація яка в результаті буде доступна користувачам при перегляді(рис. 4.6.).

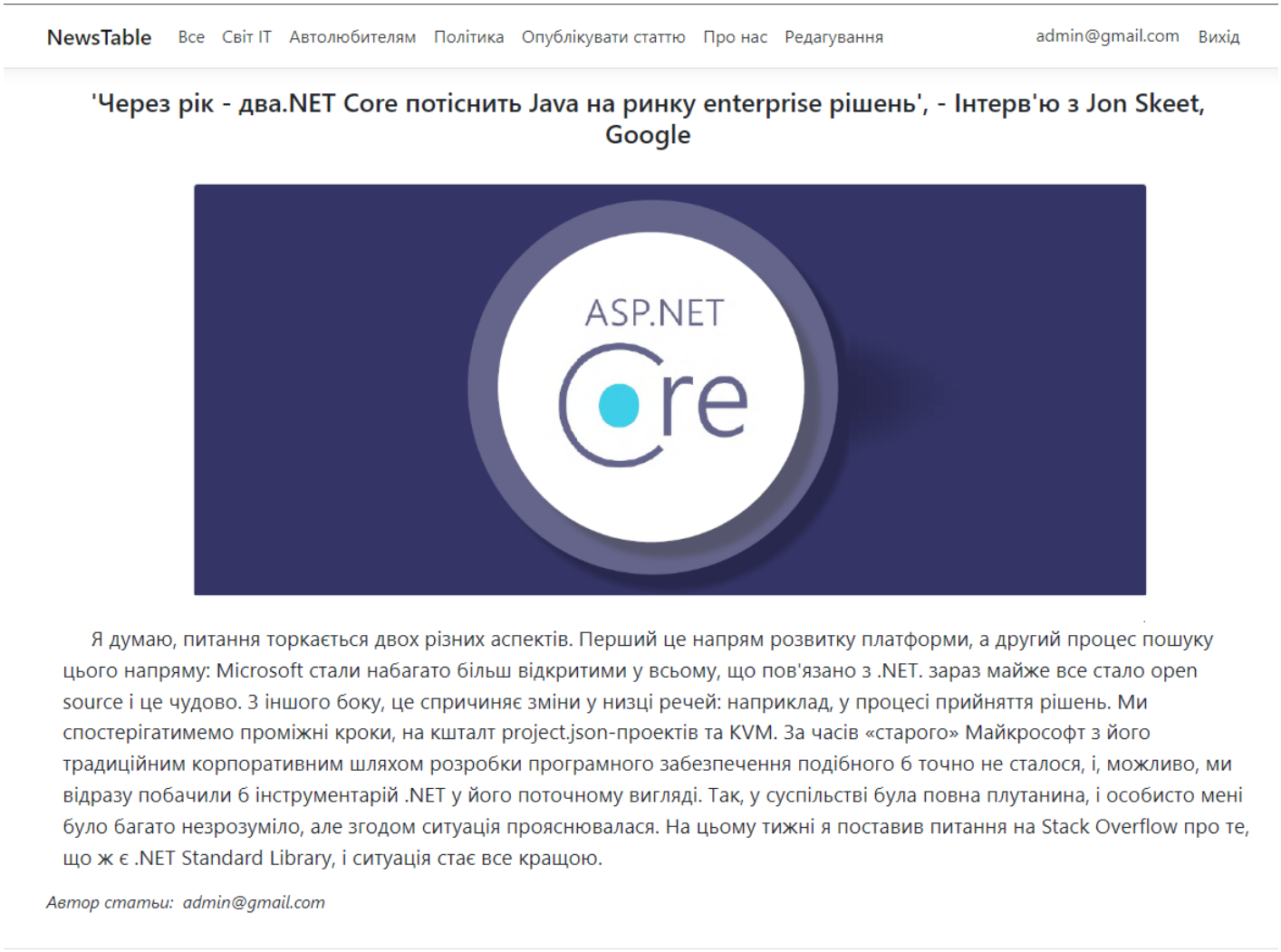

Рис.4.6 Детальна інформація про публікацію

Якщо адміністратор хоче змінити щось в публікації він натискає кнопку “Редагувати” та відкривається сторінка яка дозволяє це робити(рис. 4.7). Тут можна змінити заголовок, опис, основний текст статті, категорію, зображення, ім’я автора та інше. Це необхідно для актуалізації статті та підтримки актуального контенту для користувачів на сайті.

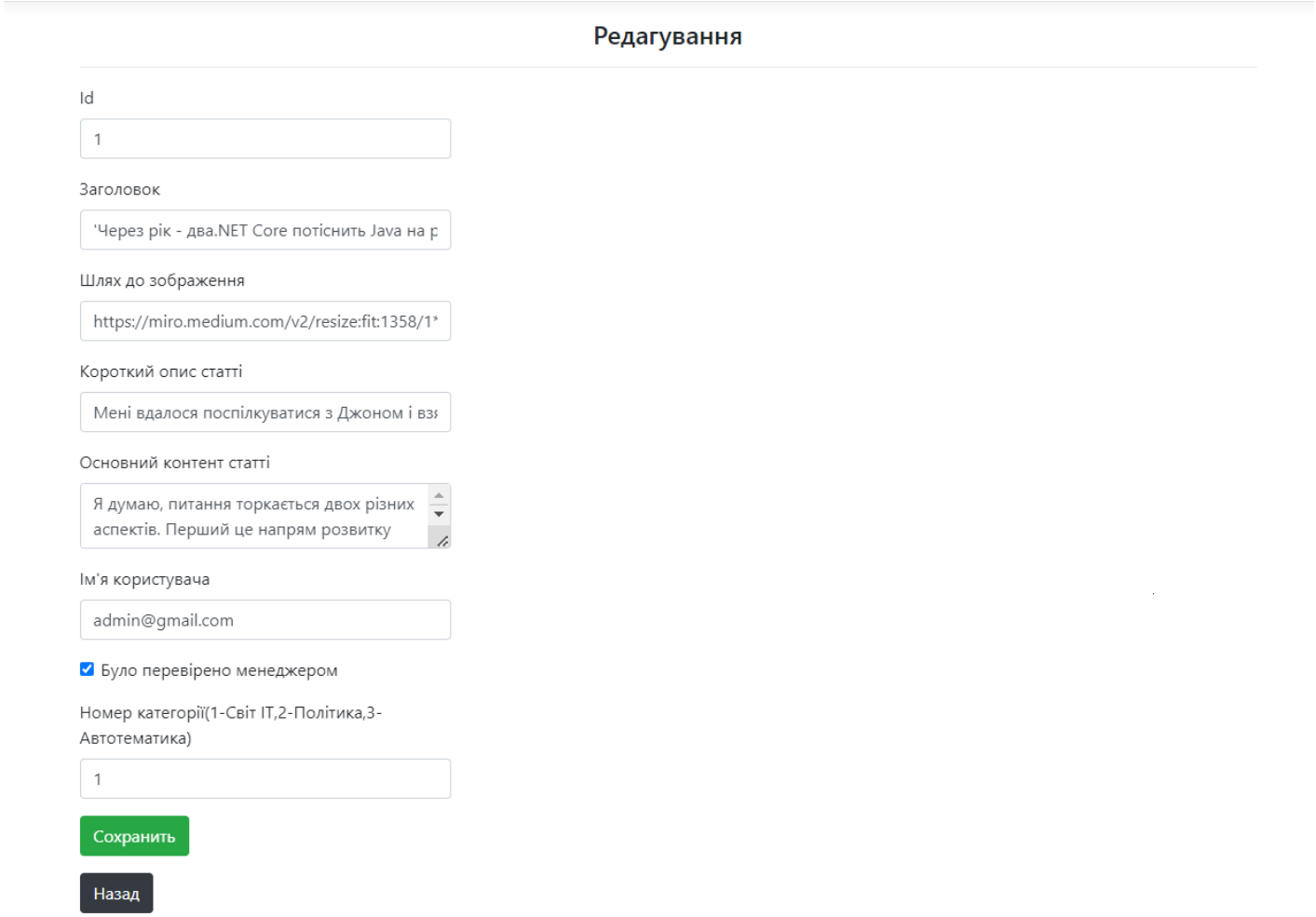

Рис.4.7 Форма редагування

Підсумовуючи наведений функціонал, модуль адміністратора являє собою необхідний елемент системи для редагування та управління всіма публікаціями, підтримку актуалізації контенту та відкидання публікацій які не відповідають правилам.

Для користування адмін інструментами користувач має мати роль адміністратора, якщо ця роль не надана користувач не побачить та не буде мати доступу до сторінки, що дозволяє уникнути небажаних управлінь даними від звичайних користувачів.

4.2. Модуль перегляду та публікації інформації

Модуль перегляду та публікації інформації містить основні функції системи задля яких вона розроблялась. Цей модуль містить такі функції як перегляд публікацій, публікування статей та пошук по категоріям. Головна сторінка додатку показує всі публікації без фільтрів, пошук статей(рис. 4.8), а також містить систему пагінації та посилання на додаткові ресурси соціальних мереж(рис. 4.9).

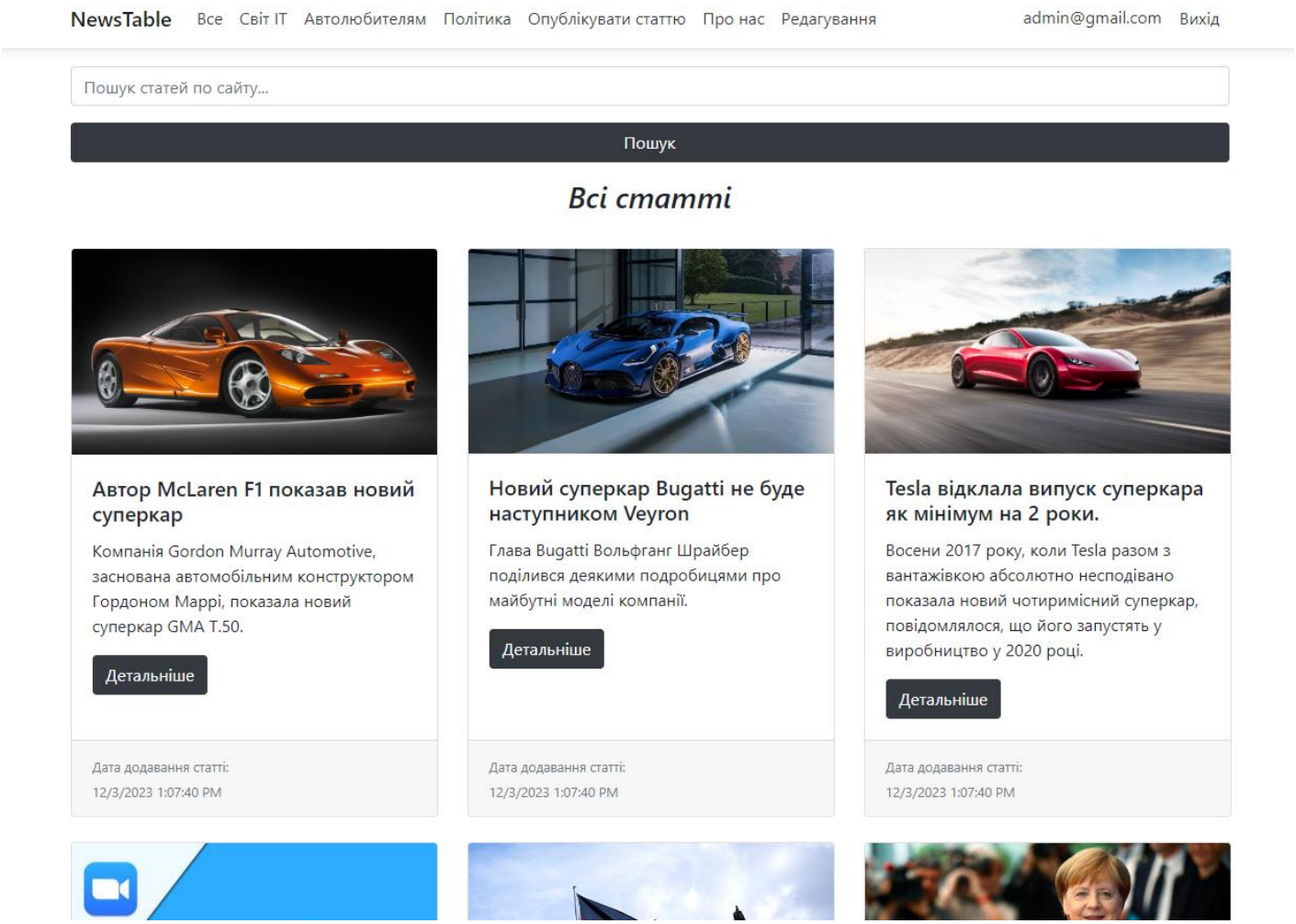

Рис.4.8 Головна сторінка додатку

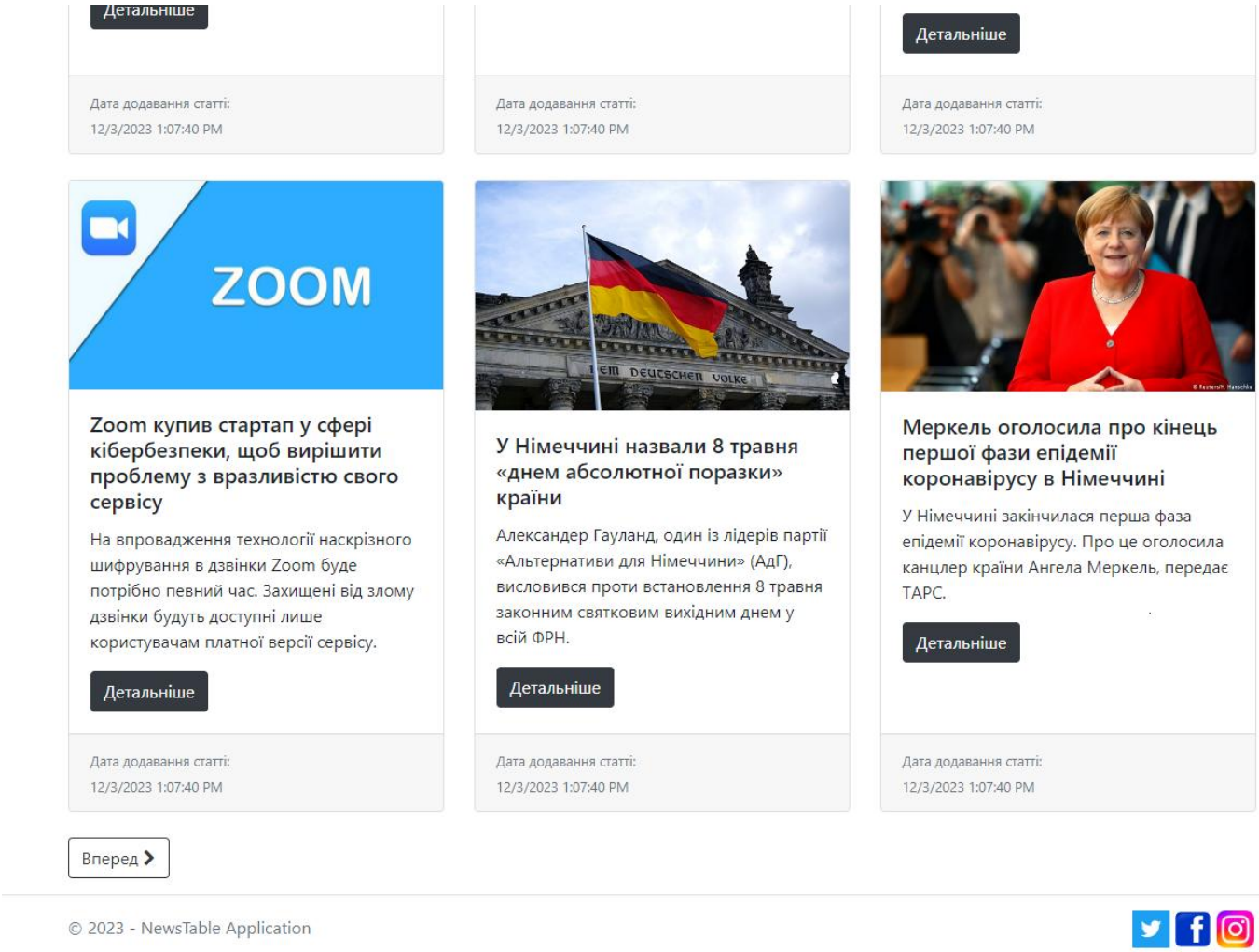

Рис.4.9 Пагінація та нижня частина сторінки

Система пагінації дозволяє містити тільки певну кількість публікацій на сторінці, якщо користувач хоче побачити більше статей він може це зробити за допомогою кнопок “Вперед” та “Назад”. Знизу розташовано посилання на соціальні мережі застосунку такі як Twitter, Facebook та Instagram.

На верхній навігаційній панелі(рис. 4.10.) розташовано декілька основних кнопок які допомагають орієнтуватися в функціональності додатку, серед них навігація по категоріям публікацій, розділ який містить інформацію про авторів або ж будь-яку корисну інформацію та функціонал опублікування статей.

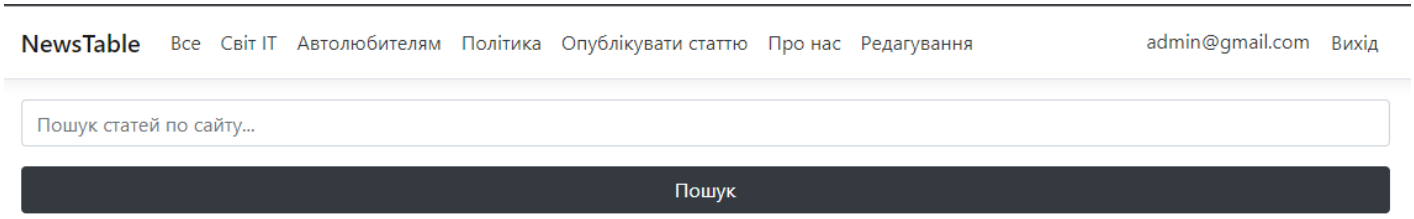

Рис.4.10 Навігаційна панель застосунку

За необхідністю категорії легко розширюються, в даному прототипі наведені лише приклади можливих категорій, в реальному проекті їх може бути більше та іншого типу. Категорії розділяють публікації по контенту для зручності та допомагають користувачам легше шукати необхідний контент на сайті. При переході на будь-яку категорію статті відфільтровуються(рис.4.11).

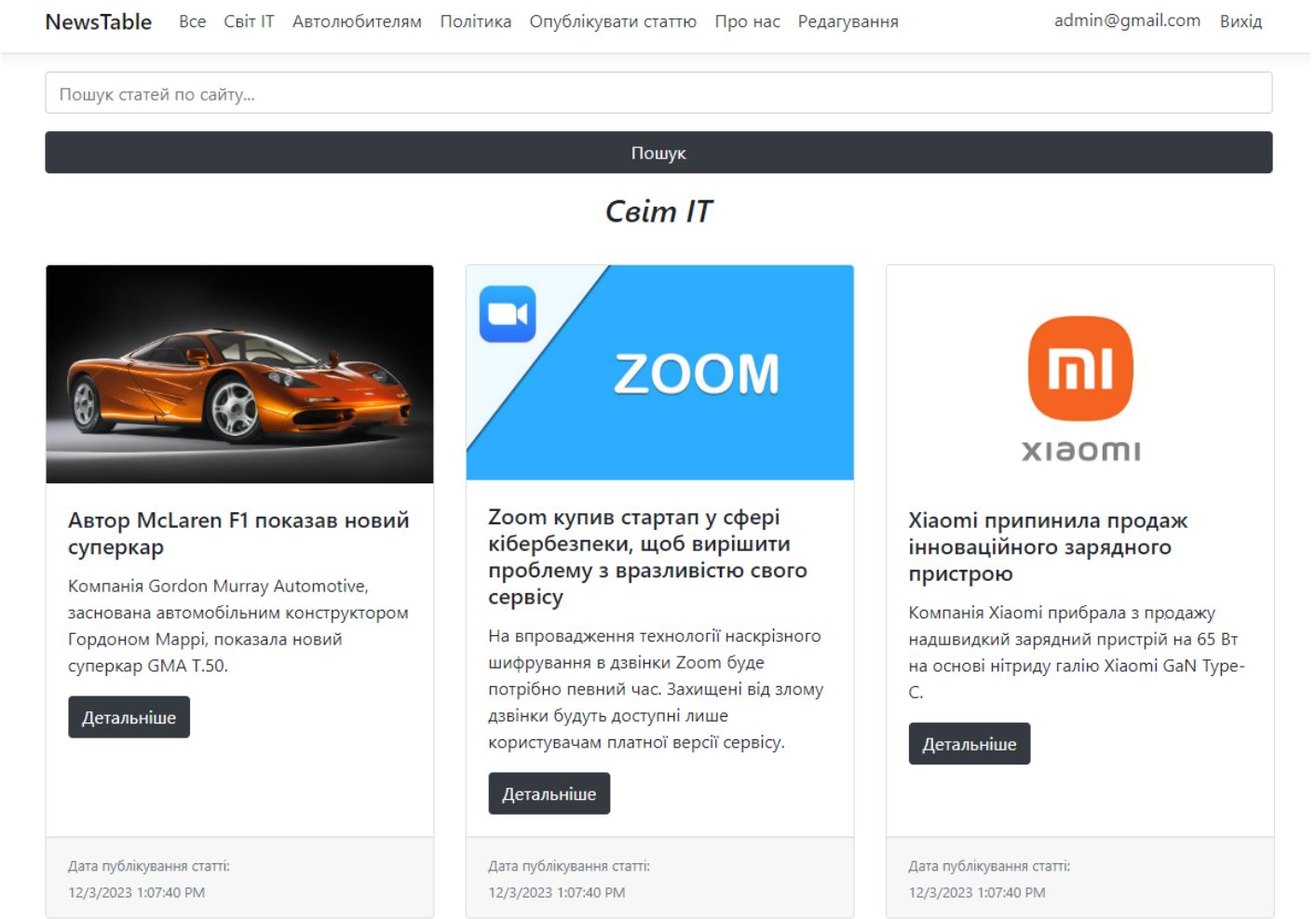

Рис.4.11 Фільтрація по категоріям

Ще одною частиною функціоналу для навігації є фільтр по назві (рис 4.12).

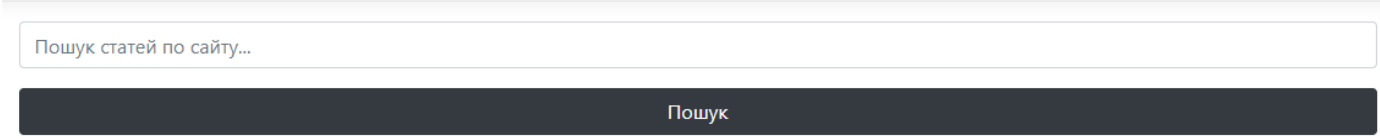

Рис.4.12 Пошук публікацій

Фільтр знайде всі статті які містять назву вказану в цьому полі.

Для перегляду повної інформації по публікації користувачу потрібно натиснути на кнопку “Детальніше”. При цьому відкриється детальна інформація по публікації (рис.4.13) де знаходиться повний опис, зображення та ім'я автора.

Рис. 4.13 Детальна інформація про публікацію

Для публікації інформації користувачу необхідно натиснути кнопку “Опублікувати статтю”. При цьому відкриється сторінка додавання нової публікації(рис. 4.14.). Всі поля необхідні для заповнення, якщо користувач не ввів необхідну інформацію спрацьовує валідація і повідомляє про необхідність заповнення даних(рис. 4.15.). Якщо всі дані коректні публікація відправляється на валідування адміністратором і згодом публікується на сторінці.

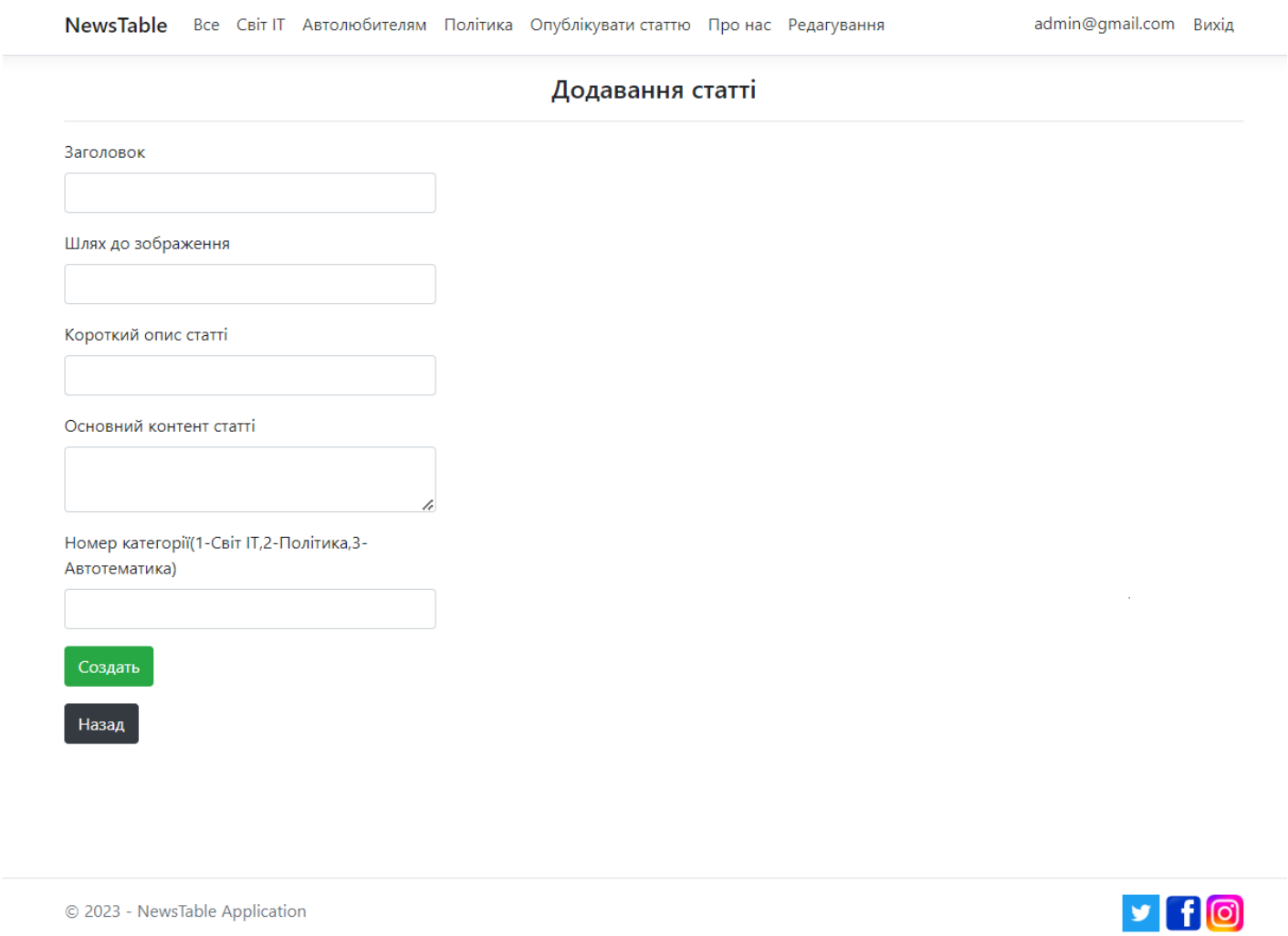

Рис.4.14 Публікація інформації

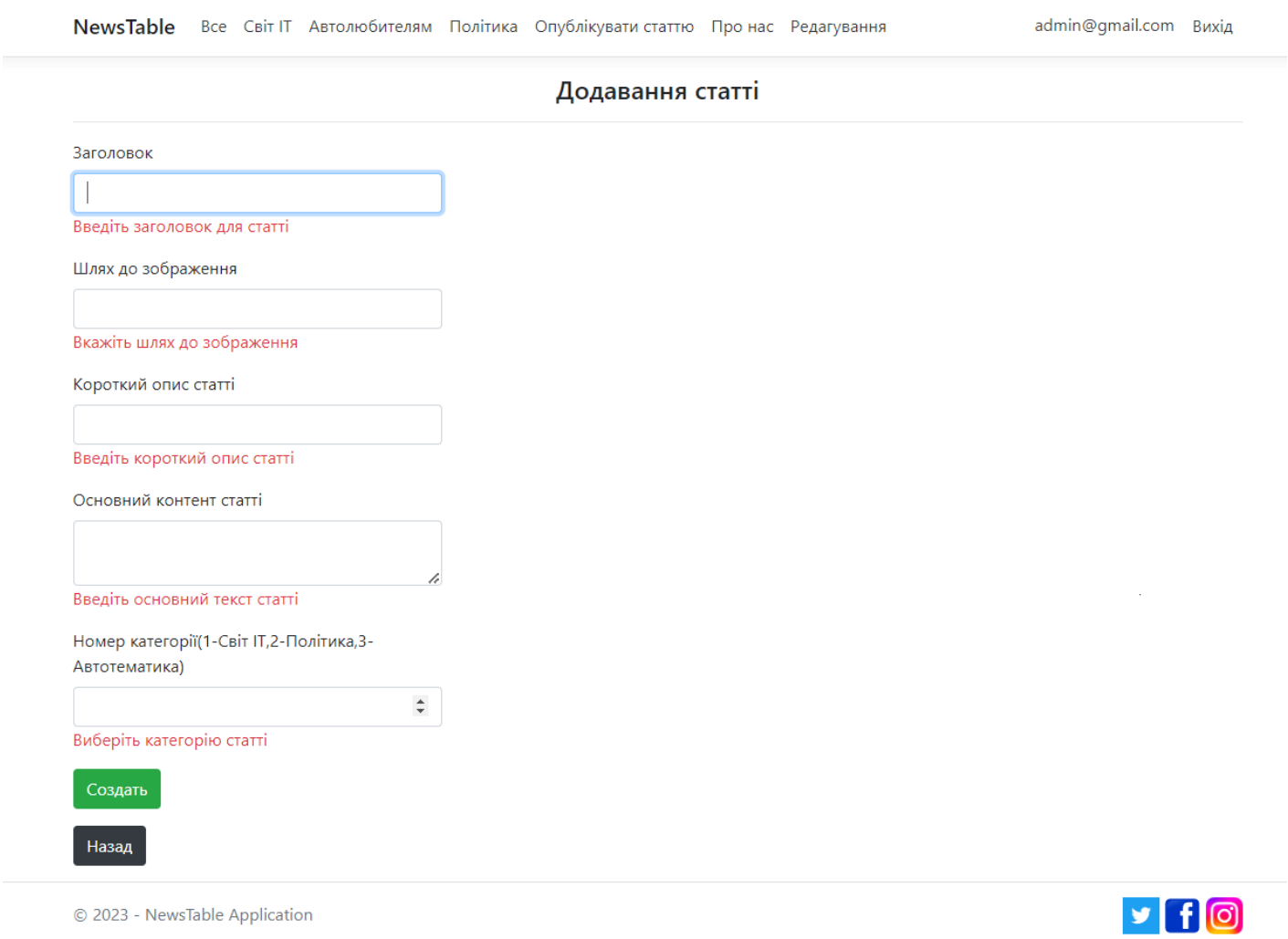

Рис.4.15 Валідування даних

4.3. Оцінка зручності клієнтської частини

Інтерфейс застосунку розроблено згідно з проаналізованих потреб користувачів системи. Для проектування використовувались стандартні інструменти створення графічних елементів які добре знайомі користувачам з інших подібних систем. Кольори застосунку не перевантажені, збережено стриманий стиль який комбінує чорно-білі кольори.

Весь інтерфейс є зручним та інтуїтивно зрозумілим для того щоб будь який недосвідчений користувач міг з легкістю використовувати систему. Текст оптимальних розмірів та всі елементи розташовані в полі зору користувача та задокументовані.

4.4. Напрямки оптимізації та можливості розширення застосунку

Застосунок спроектовано використовуючи платформу .NET Core яка включає різні механізми оптимізації та гарантування безпеки даних. Для роботи з базою даних використано Entity Framework Core що забезпечує оптимальне використання та розширення бази даних. Сама база даних MSSQL є реляційною базою даних що оптимізує запити та підтримує велику структуру БД.

В майбутньому для оптимізації запитів є можливість використання індексів бази даних. Це необхідно робити лише після аналізу запитів до БД та полей які найчастіше використовуються при діставанні та фільтрації даних. Якщо буде виявлено такі поля, наприклад Name по якому здійснюється пошук публікацій, то необхідно додати індекс на цю колонку в БД. Важливо робити ці дії лише після аналізу перформенсу тому що навішування індексів може і негативно вплинути на швидкодію застосунку. Також якщо при аналізі перформенсу будуть виявлені деякі гепи необхідно проаналізувати код за допомогою профайл інструментів, виявити місця де код не оптимізовано та переписати його згідно з потреб.

При розширені кількості користувачів гарним покращенням буде додати додаткову NoSQL базу даних для зберігання сесій та кеш файлів. Для цих потреб може підійти Redis.

При бажанні масштабування застосунку великих труднощів це викликати не повинно, бо додаток спроектовано згідно з основними принципами розробки програмного забезпечення та використанням деяких архітектурних паттернів таких як репозиторії та UnitOfWork.

Якщо підсумувати то застосунок спроектовано так щоб його можна було легко розширити, оптимізувати або переписати під свої цілі.

Висновок до розділу 4

В даному розділі розглянуто основні функціональні модулі та можливості системи такі як модуль адміністратора і модуль пошуку та опублікування інформації. Всі функції системи були детально описані та продемонстровані.

Також проаналізовано розроблений інтерфейс з точки зору зручності для користувача, наведено рекомендації, можливості розширення та оптимізації системи в майбутньому.

ВИСНОВКИ

В результаті виконаної роботи було спроектовано та розроблено веб-застосунок для пошуку та публікування інформації студентами. Побудовано та описано використану архітектуру та основні архітектурні шаблони що було використано при розробці системи.

Інтерфейс застосунку відповідає потребам клієнтів які були сформовані та проаналізовані до початку роботи над системою. Всі елементи є зручними у використанні та інтуїтивно зрозумілі для користувача.

Застосунок вирішує проблему комунікації студенів, за допомогою нього всі бажаючі користувачі зможуть з легкість шукати та публікувати інформацію по різним типам та категоріям. Система прозора та відкрита для будь-яких типів користувачів. Реалізовані основні функція такі як публікування, пошук, валідування та навігація по інформації.

Архітектура була продемонстрована за допомогою найважливіших UML діаграм, було показано всі основні зв'язки сутностей, юзер флоу, діаграму розгортання та інше. Технології та інструменти які використано описано та наведено пояснення чому саме ці інструменти використано в застосунку.

Сформовано потреби по оптимізації та масштабуванню додатку при майбутньому зростанні об’єму функціональності та користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Платформа публікування статей Medium. Режим доступу: https://medium.com/
2. Платформа публікування статей EduBlog. Режим доступу: https://edublogs.org/
3. Професійна соціальна мережа LinkedIn. Режим доступу: https://www.linkedin.com/
4. Підсумки 2022 року на Хабрі. Режим доступу: https://habr.com/ru/companies/habr/articles/705476/
5. C# як мова програмування. Режим доступу: https://www.developer.com/guides/what-is-c/, https://www.w3schools.com/cs/cs_intro.php
6. Роль UML діаграм. Режим доступу: https://miro.com/blog/uml-diagram/#:~:text=Most%20commonly%2C%20a%20UML%20diagram,UML%20diagram%20is%20incredibly%20helpful.
7. Функціональні вимоги. Режим доступу: https://www.guru99.com/functional-requirement-specification-example.html
8. Не функціональні вимоги. Режим доступу: https://www.guru99.com/functional-requirement-specification-example.html