

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:
«Автоспортивний інформаційно-статистичний портал»

Виконав студент групи ПЗ-22-2
БЛИЗНЮК Тарас Русланович

Керівник роботи:
САВІЦЬКИЙ Роман Святославович

Рецензент:
ЛОБАНЧИКОВА Надія Миколаївна

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

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **БЛИЗНЮК Тарас Русланович**

Керівник роботи: **САВІЦЬКИЙ Роман Святославович**

Тема роботи: **«Автоспортивний інформаційно-статистичний портал»**

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92/с**

Вихідні дані для роботи: об'єктом дослідження є засоби та методи для розробки вебплатформи автоспортивного інформаційно-статистичного порталу; предметом дослідження є методи та засоби розробки сучасного автоспортивного інформаційно-статистичного порталу.

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Савіцький Р.С	28.02.2026	28.02.2026
2	Савіцький Р.С	28.02.2026	28.02.2026
3	Савіцький Р.С	28.02.2026	28.02.2026

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра виконана на тему «Автоспортивний інформаційно-статистичний портал». Робота складається зі вступу, трьох частин, висновків, а також списку використаних джерел. Обсяг записки становить 72 сторінки, а також 52 рисунки, 23 таблиці та 32 джерела.

Головною метою є створення сучасного автоспортивного інформаційно-статистичного порталу, який дасть користувачам можливість читати та обговорювати різноманітну інформацію, пов'язану з перегонами, водіями тощо, шукати необхідні об'єкти, а також переглядати та аналізувати великі обсяги статистичних даних.

У процесі роботи було сформовано вимоги до розробки порталу та проаналізовано відомі аналоги рішень для розробки автоспортивного інформаційно-статистичного порталу. У результаті роботи було обрано архітектуру додатка, яка задовольняє дані вимоги. Також було визначено можливі варіанти використання системи та створено об'єктно-орієнтовану структуру. Виконано розробку БД та алгоритмів системи, які підпадають під можливі варіанти її використання.

Ключові слова: ПОРТАЛ, ПЕРЕГОНИ, АВТОСПОРТ, СТАТТІ, ВОДІЇ, ДИСЦИПЛІНИ, БАЗА ДАНИХ, ФУНКЦІОНАЛ

					ІПЗ.КР.Б-121-26-ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата	Автоспортивний інформаційно- статистичний портал		
Розроб.		Близнюк Т.Р.					
Керівник		Савіцький Р.С.					
Рецензент		Лобанчикова Н.М.					
Зав. каф.		Вакалюк Т.А.					
					Літ.	Арк.	Аркушів
						3	82
					Житомирська політехніка, група ІПЗ-22-2		

ABSTRACT

This bachelor's degree dissertation is entitled 'A Motorsport Information and Statistics Portal'. The dissertation comprises an introduction, three chapters, conclusions, and a list of references. It consists of 72 pages, as well as 52 figures, 23 tables and 32 references.

The main aim is to create a modern motorsport information and statistics portal that will enable users to read and discuss a wide range of information relating to races, drivers and so on, search for specific items, and view and analyse large volumes of statistical data.

During the course of the work, requirements for the development of the portal were defined and known analogous solutions for the development of a motorsport information and statistics portal were analysed. As a result of this work, an application architecture was selected that meets these requirements. Possible scenarios for using the system were also identified, and an object-oriented structure was created. The database and system algorithms were developed to accommodate the possible scenarios for its use.

Keywords: PORTAL, RACING, MOTORSPORT, ARTICLES, DRIVERS, DISCIPLINES, DATABASE, FUNCTIONALITY

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.	I			4

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ.....	9
1.1. Постановка задачі.....	9
1.2. Аналіз аналогів програмного продукту	11
1.3. Вибір архітектури веборієнтованої системи.....	15
1.4. Обґрунтування вибору інструментальних засобів.....	18
Висновки до першого розділу.....	21
РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ.....	23
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	23
2.2. Розробка бази даних системи.....	26
2.3. Проєктування та реалізація алгоритмів роботи системи.....	34
2.4. Реалізація функціоналу системи.....	38
Висновки до другого розділу.....	41
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ПОРТАЛУ.....	43
3.1. Встановлення варіантів використання системи.....	43
3.2. Структура інтерфейсу системи.....	46
3.3. Тестування системи.....	63
Висновки до третього розділу.....	67
ВИСНОВКИ.....	68
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	70
ДОДАТКИ.....	73
Додаток А.....	74

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

API – прикладний програмний інтерфейс

CI/CD – набір процесів для автоматизації розгортання ПЗ

CRUD – основні операції роботи з даними

JWT – вебтокен JSON

ORM – інструмент для роботи з БД

SPA – односторінковий додаток

UI – користувацький інтерфейс

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				6

ВСТУП

Актуальність роботи. Стрімке зростання популярності автоспорту та активний перехід сучасних медіа у цифровий простір змінило підхід до висвітлення спорту, особливо автоспорту. Значний вплив на залучення нової аудиторії до перегонів мають різноманітні документальні фільми і серіали та книги про автоспорт, які дозволяють мільйонам глядачів та читачів по всьому світу глибше зануритись у внутрішню кухню цього виду спорту, історії команд, пілотів та їх протистоянь. У результаті цього сформувалась велика спільнота фанатів, яка зацікавлена не лише у перегляді самих перегонів, а й у доступі до статистики, аналітики, новин та історичних даних.

Сучасний автоспорт характеризується великим обсягом інформації щодо результатів перегонів, чемпіонатів, статистики пілотів і команд. Водночас ці дані часто розпорошені між різними ресурсами, що ускладнює їх сприйняття та аналіз. Це зумовлює потребу у створенні єдиного інформаційно-статистичного порталу, який об'єднає дані про автоспортивні події, надасть зручні інструменти для перегляду статистики та сприятиме популяризації перегонів серед ширшої аудиторії.

Розвиток вебтехнологій дозволяє створювати інтерактивні платформи, які забезпечують швидкий доступ до актуальної інформації з будь-яких пристроїв. Завдяки цьому фанати можуть у режимі реального часу слідкувати за результатами змагань, аналізувати статистику пілотів і команд, а також взаємодіяти між собою в межах спільноти. Такий підхід відповідає сучасним тенденціям створення та споживання контенту.

Мета роботи. Створити автоспортивний інформаційно-статистичний портал, що забезпечує функціональність для збору, обробки та візуалізації даних про перегони й пілотів і надає зручний інтерфейс користувачам. Особливу увагу приділено формуванню аналітичних розділів даних, а також модерації контенту та взаємодії користувачів.

Завданням роботи є аналіз існуючих аналогів спортивних порталів задля визначення їх переваг та недоліків, визначення функціональних та

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				7

нефункціональних вимог до розробки системи, проєктування архітектури ПЗ для клієнтської і серверної частин системи та побудова БД, а також тестування функціоналу порталу.

Об’єктом дослідження є система автоспортивного інформаційно-статистичного порталу, яка забезпечує доступ до актуальних даних і сприяє об’єднанню фанатської спільноти.

Предметом дослідження є засоби та методи для створення автоспортивного інформаційно-статистичного порталу.

Практичне значення роботи полягає у розробці системи автоспортивного інформаційно-статистичного порталу, призначеного для зберігання, опрацювання та представлення автоспортивної інформації й статистики. Результати роботи можуть бути використані для створення та подальшого розвитку тематичних спортивних порталів, а також як приклад реалізації сучасної інформаційної системи на основі вебтехнологій.

За темою випускної кваліфікаційної роботи бакалавра було опубліковано тези Міжнародної науково-практичної конференції «Теорія і практика сучасної науки та освіти», 19-20 червня 2026 року. Львів: «Львівський науковий форум» [31, 32].

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				8

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1. Постановка задачі

Розробка автоспортивної інформаційно-статистичної вебплатформи є поетапним та комплексним процесом, основною метою якого є створення єдиного цифрового ресурсу для зберігання, обробки та представлення даних про автоспортивні змагання, а також забезпечення взаємодії між користувачами. Проєкт орієнтований на об'єднання фанатів різних дисциплін спортивних перегонів у межах однієї інформаційної платформи.

Архітектура вебплатформи передбачає наявність клієнтської частини, яка є “відображенням” платформи, а також серверної частини з підключенням до бази даних, яка забезпечує централізоване зберігання та обробку інформації. У базі даних зберігаються відомості про серії перегонів, сезони, події, траси, команди, пілотів, боліди, користувацькі акаунти, а також статті з абзацами до них, обговорення та коментарі. Такий підхід дозволяє забезпечити цілісність даних, їх безпечне зберігання та швидкий доступ до інформації.

Основою взаємодії користувачів із платформою є система авторизації та автентифікації, яка забезпечує створення облікових записів та розмежування доступу відповідно до ролей. На платформі реалізовано ролі звичайного користувача (має доступ до перегляду контенту, статистики, участі в обговореннях, залишення коментарів і реакцій), редактора (має можливість створювати, редагувати та публікувати статті, аналітичні матеріали та новини) та адміністратора (володіє повним набором функціональних можливостей, включно з управлінням користувачами, контентом, статистичними даними та системними налаштуваннями).

Платформа підтримує систему коментарів і реакцій як до публікацій і тредів, так і до окремих коментарів, що сприяє активному обговоренню автоспортивних подій. Для забезпечення якості контенту передбачені механізми

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
		№ докум.				9

подання скарг і модерації, що дозволяє оперативно реагувати на порушення правил користування.

Однією з ключових функціональних особливостей платформи є робота з великими масивами статистичних даних. Бекенд відповідає за обробку, агрегацію та оптимізовану передачу даних клієнтській частині, що дозволяє зберігати високу швидкодію навіть при значних обсягах інформації. Реалізуються механізми фільтрації, пагінації та кешування статистичних даних про пілотів, траси, боліди, команди та події.

Вебплатформа реалізується у форматі односторінкового застосунку, що забезпечує швидке завантаження сторінок та сучасний підхід до побудови вебдодатків. Додатково реалізується підтримка перекладу декількома мовами, яка дозволяє користувачам обирати мову інтерфейсу.

Інтеграція сторонніх сервісів здійснюється вибірково. Зокрема, Google OAuth [27] інтегрується для можливості виконати вхід або реєстрацію на порталі за допомогою наявного облікового запису Google, що надає можливість зручного доєднання нових користувачів до порталу.

Функціональна частина системи включає реалізацію операцій для основних сутностей, зокрема користувачів, статей та абзаців до них, коментарів та статистичних даних. Також додано функціонал для адміністраторів, який забезпечує управління платформою та доступ до аналітичної інформації.

З огляду на обробку значних обсягів даних та наявність різних рівнів доступу, особливу увагу приділено питанням безпеки, зокрема захисту процесів авторизації й автентифікації, контролю доступу до ресурсів та безпечному зберіганню даних у БД.

Результатом реалізації проєкту є сучасний автоспортивний інформаційно-статистичний портал з повноцінною об'єктно-орієнтованою архітектурою, яка забезпечує високу продуктивність, безпечність та зручність використання. Продумана система ролей, інтеграція зовнішніх сервісів і підтримка великих обсягів даних дозволяють платформі ефективно виконувати поставлені завдання та відповідати актуальним потребам користувачів [10].

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				10

1.2. Аналіз аналогів програмного продукту

Перед початком створення власного порталу потрібно проаналізувати наявні аналоги, щоби зрозуміти, який функціонал варто взяти для порталу, а також які недоліки варто враховувати та виправляти в разі їх знаходження.

Під час аналізу порталів, які або мають автоспортивну тематику, або чудово поєднують статистику та користувацькі обговорення, було обрано наступні відомі портали: Motorsport.com, The Race та Flashscore. Нижче наведено детальний огляд кожної платформи.

Motorsport.com (рис. 1.1) є одним із найбільших та найавторитетніших медіаресурсів у сфері автоспорту. Платформа працює на міжнародному рівні та підтримує велику кількість мовних локалізацій, що дозволяє їй охопити широку аудиторію користувачів з різних країн світу, а також висвітлює як чемпіонати найвищого рівня, так і молодіжні чи регіональні серії. Основна увага приділяється новинам та аналітичному контенту, який регулярно оновлюється редакцією. Користувачі мають доступ до календарів, результатів, турнірних таблиць, кваліфікаційних заїздів та гонок. Дані організовані за сезонами, серіями та подіями, що полегшує навігацію. Інтерфейс платформи дозволяє швидко перемикатися між окремими серіями та сезонами, що особливо важливо для користувачів, які аналізують історичні результати змагань. Важливою перевагою є інтеграція з власною платформою Motorsport Stats, яка є однією з найбільших баз автоспортивної статистики. Водночас Motorsport.com не підтримує створення коментарів на власній платформі, а використовує сторонні плагіни. Платформа насамперед орієнтована на споживання контенту, а не на взаємодію з користувачами. Також відсутні повноцінні засоби персоналізації контенту та можливість створювати власні списки улюблених гонщиків чи команд, що дещо обмежує залученість аудиторії у довготривале використання сервісу.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				11
	.	№ докум.				

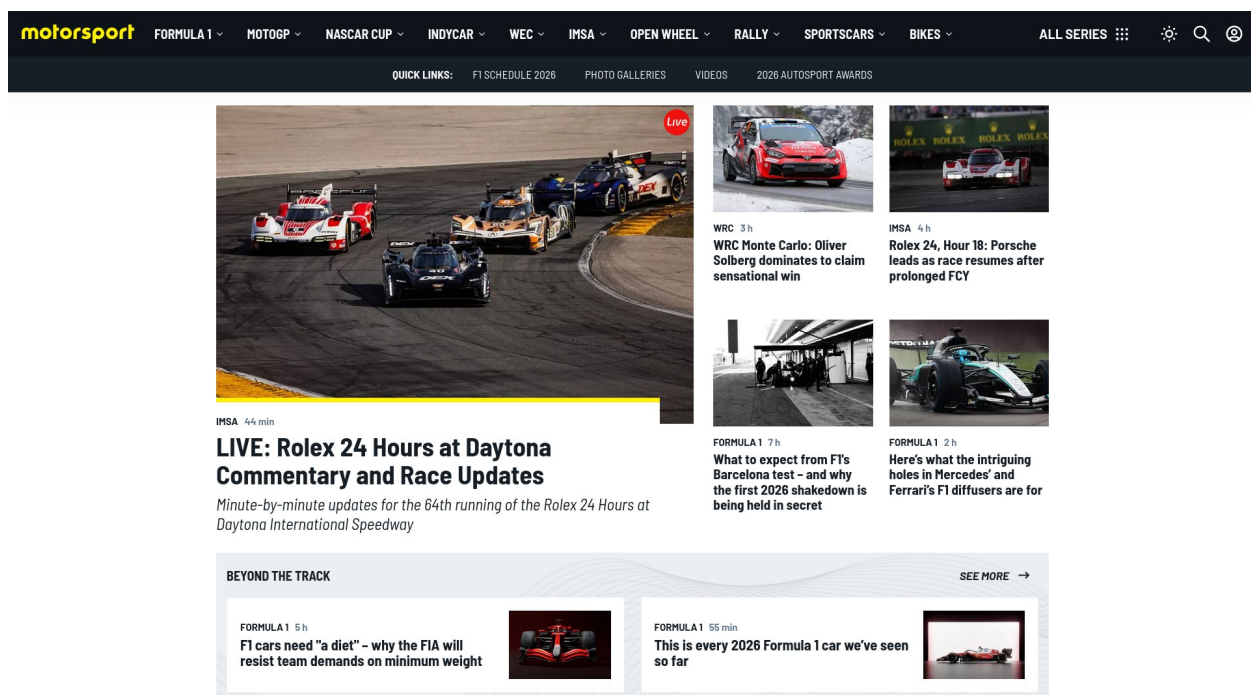


Рис. 1.1. Головна сторінка Motorsport.com

The Race (рис. 1.2) є сучасним цифровим ЗМІ, яке спеціалізується на журналістиці та детальному експертному аналізі. Видання активно співпрацює з професійними журналістами та експертами з автоспорту, які мають багаторічний досвід роботи у провідних командах та ЗМІ. Платформа обробляє значно менше даних, ніж Motorsport.com, але вирізняється якістю контенту та орієнтованістю на авторів. Основна аудиторія складається з уболівальників, які прагнуть зрозуміти внутрішні перипетії команд, експертний аналіз болідів та підтвердження чи спростування різноманітних чуток. Сайт має лаконічний інтерфейс, оптимізований для читання та перегляду мультимедійного контенту. Велика увага приділяється власному відео- та аудіоконтенту, що створює відчуття «преміум»-продукту. Однак платформа практично не використовує статистичні дані у вигляді таблиць, фільтрів або історичних оглядів. Крім того, The Race дозволяє користувачам коментувати лише після підписки на їхній Patreon. Це створює додатковий бар'єр для входу для частини аудиторії, але водночас підвищує рівень залученості та фінансової підтримки проекту з боку найбільш зацікавлених користувачів.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				12

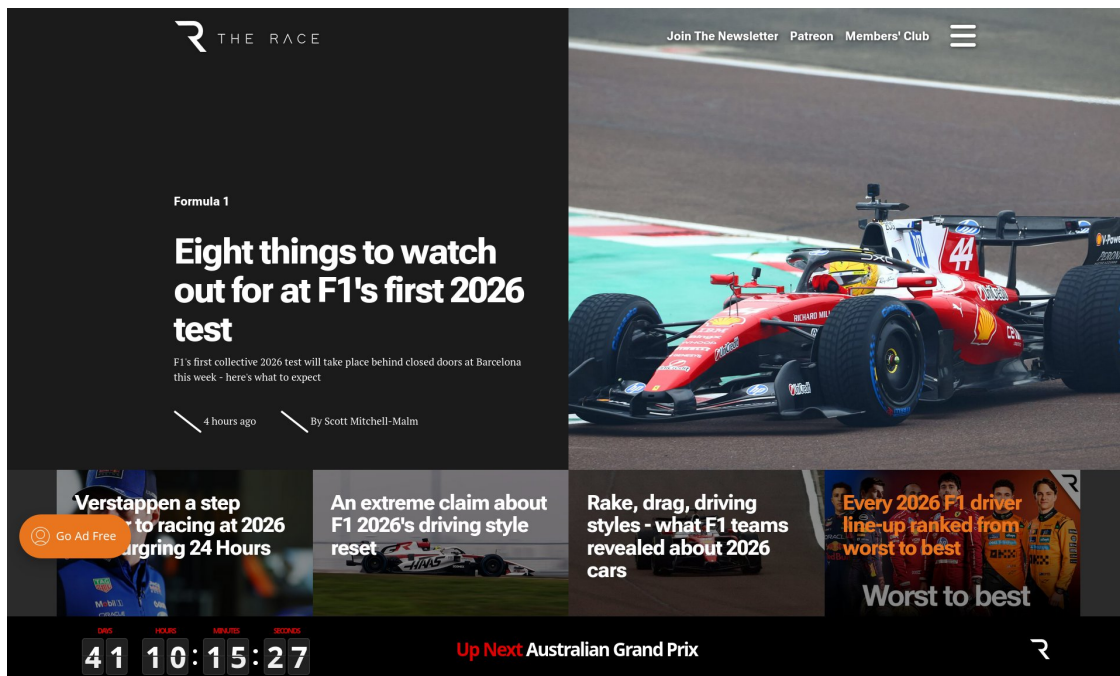


Рис. 1.2. Головна сторінка The Race

Flashscore (рис. 1.3) є універсальною спортивною платформою, головна перевага якої полягає у швидкості оновлення даних. Сервіс спеціалізується на обробці результатів у режимі реального часу для великої кількості спортивних дисциплін, включаючи футбол, теніс, баскетбол та автоспорт. Хоча автоспорт не є її основним напрямком, сервіс демонструє еталонний підхід до обробки даних у прямому ефірі, календарів та результатів у реальному часі. Архітектура Flashscore побудована на оптимізованій обробці великих обсягів подій, що дозволяє платформі миттєво оновлювати інформацію без перевантаження клієнтської сторони. Дані подаються у стислому форматі з акцентом на швидкий доступ, без глибокої аналітики чи історичних пояснень. Платформа майже повністю позбавлена соціальних функцій: тут немає коментарів, реакцій чи будь-якої форми внутрішньої спільноти. Flashscore – це насамперед інструмент для моніторингу подій, а не медіа- чи фан-портал. Таким чином, сервіс займає окрему нішу між статистичними базами даних та медіаплатформами, зосереджуючись виключно на швидкій та точній трансляції спортивної інформації.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				13

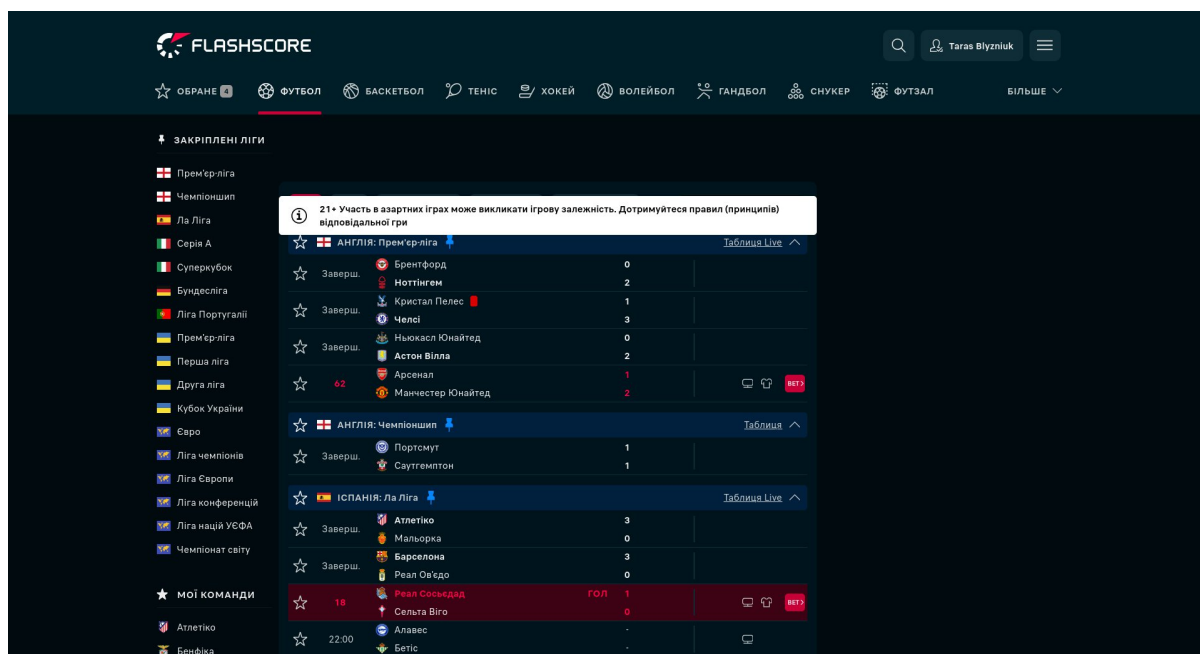


Рис. 1.3. Головна сторінка Flashscore

Порівняння в таблиці 1.1 показує, що Flashscore краще працює з даними та має чудову підтримку мобільних пристроїв, але є загальним порталом. The Race є новинним сайтом, але має доволі нішеві користувацькі можливості, хоча його дизайн є доволі привабливим. Motorsport.com є автоспортивним порталом, який чудово вбирає у себе всі потреби користувачів даної платформи, але теж має певні нюанси, де можливі покращення. Враховуючи всі переваги та недоліки розглянутих порталів, можна будувати архітектуру власного portalу. Також до таблиці було додано функціонал, який у подальшому буде реалізованим на порталі REVITUP.

Таблиця 1.1

Порівняння платформ-аналогів

Функціонал	Motorsport.com	The Race	Flashscore	REVITUP
Автоспортивні серії	+	+	+	+
Live-результати	+	-	+	-
Статистика/дані	+	-	+	+
Новини/аналітика	+	+	-	+
Медіа	+	+	-	+
Коментарі/спільнота	-	-	-	+

Підтримка дискусій/тредів	+	+	(без вільного доступу)	-	+
Рейтинги/оцінки	+	(таблиці учасників)	-	-	-
Мобільний доступ	+	+	-	+	+
Увага на глибокі дані	+	-	+	+	+

1.3. Вибір архітектури веборієнтованої системи

Вибір архітектури є фундаментальним етапом у розробці програмного проєкту, оскільки він визначає, як взаємодіють окремі компоненти системи, механізми передачі даних, а також принципи, що регулюють проєктування та представлення користувацького інтерфейсу. Архітектурні рішення впливають на структуру реалізації програмного забезпечення, розподіл відповідальності між модулями та спосіб їх взаємодії. Добре спроектована архітектура дозволяє масштабувати систему в майбутньому, спрощує її обслуговування та зменшує ризик виникнення критичних помилок під час розробки проєкту [15].

Для цього проєкту було обрано архітектуру на основі REST API, що дозволяє чітко розділити клієнтські та серверні компоненти додатка. Такий підхід забезпечує незалежність фронтенду та бекенду, що в довгостроковій перспективі дає змогу модифікувати або повністю замінити один із компонентів без внесення змін до іншого. Це особливо важливо з огляду на стрімкий розвиток фронтенд-технологій та потенційну необхідність використання нових фреймворків у майбутньому.

REST API забезпечує гнучку модель взаємодії між клієнтом і сервером, оскільки серверна сторона несе повну відповідальність за обробку запитів і повернення структурованих даних у форматі JSON. Відтворення та інтерпретація цих даних повністю покладаються на клієнтську сторону, що значно спрощує розробку, тестування та масштабування системи.

Серверна частина проєкту реалізована з використанням фреймворку NestJS [16], який базується на модульному підході до розробки додатків. Код організовано в окремі модулі, що відповідають сутностям домену, таким як

користувачі, статті, події, статистика та коментарі. Така структура покращує читабельність коду, полегшує обслуговування та спрощує розширення функціональності. NestJS надає широкий спектр вбудованих механізмів, включаючи гвардії, перехоплювачі та труби, які використовуються для реалізації контролю доступу, обробки запитів та інтеграції авторизації та автентифікації JWT [11].

Для оптимізації обробки великих обсягів даних та зменшення навантаження на основну базу даних на стороні сервера використовується база даних у пам'яті Valkey [28], яка є сумісною із Redis [22]. Valkey використовується як проміжний рівень кешування для даних, що часто запитуються, зокрема статистики перегонів, профілів водіїв та результатів подій. Використання кешування значно скорочує час відгуку сервера, покращує загальну продуктивність системи та забезпечує безперебійну роботу платформи навіть під час значного навантаження.

Для забезпечення надійності та безпеки системи серверна частина використовує багаторівневий механізм обробки запитів. Вхідні дані перевіряються перед виконанням бізнес-логіки, що запобігає неправильному або потенційно небезпечному введенню. Крім того, використання JWT-автентифікації та рольової моделі доступу забезпечує контроль над виконанням операцій та захист ресурсів платформи від несанкціонованого доступу.

Клієнтська частина вебплатформи побудована на базі фреймворку React [21], що дозволяє створювати динамічний та масштабований користувацький інтерфейс у вигляді SPA. Обмін даними між клієнтом і сервером здійснюється за допомогою бібліотеки Ку [14] – сучасної та компактної альтернативи Axios [1], побудованої на базі стандартного Fetch API. Використання Ку дозволяє зменшити обсяг коду на стороні клієнта, забезпечує прозору обробку HTTP-запитів та спрощує обробку помилок.

Бібліотека TanStack Query [25] використовується для управління асинхронними запитами та станом даних на стороні клієнта, забезпечуючи автоматичне отримання, кешування та синхронізацію даних із сервером. Хук

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				16

useQuery дозволяє здійснювати орієнтоване на UI/UX управління часом очікування відповіді сервера, відображенням стану завантаження та належним обробленням помилок. Усі HTTP-запити здійснюються через протокол HTTP(S) та перевіряються на відповідність політиці CORS.

З архітектурної точки зору клієнтська частина побудована за багаторівневою архітектурою, в якій кожен рівень відповідає за певний тип логіки. Базовий рівень складається з шару API-сервісів, реалізованих за допомогою Ку, які відповідають за прямий зв'язок із серверною частиною. Рівень управління станом реалізовано за допомогою TanStack Query, що дозволяє централізовано керувати даними та їхнім життєвим циклом. Окрім отримання даних, дана бібліотека автоматично кешує результати запитів, перезавантажує дані за потреби та синхронізує стан інтерфейсу із сервером.

Логіка візуалізації інтерфейсу розподілена між двома основними рівнями: папкою «pages», компоненти якої виступають контролерами для окремих маршрутів і відповідають за підготовку даних, та папкою «components», яка містить багаторазові атомарні елементи інтерфейсу для візуалізації інформації. Навігація між функціональними блоками системи реалізована за допомогою маршрутизатора, що забезпечує плавний перехід між сторінками без їх перезавантаження, зберігаючи цілісність користувацького досвіду.

Запропонована архітектура (зображена на рис. 1.4) забезпечує чітку структуру коду, мінімізує дублювання та гарантує, що проєкт буде простим в обслуговуванні та масштабуванні в майбутньому. Кожен компонент системи виконує певний набір завдань, що зменшує концентрацію логіки в одному місці та підвищує загальну надійність і стабільність вебплатформи.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				17

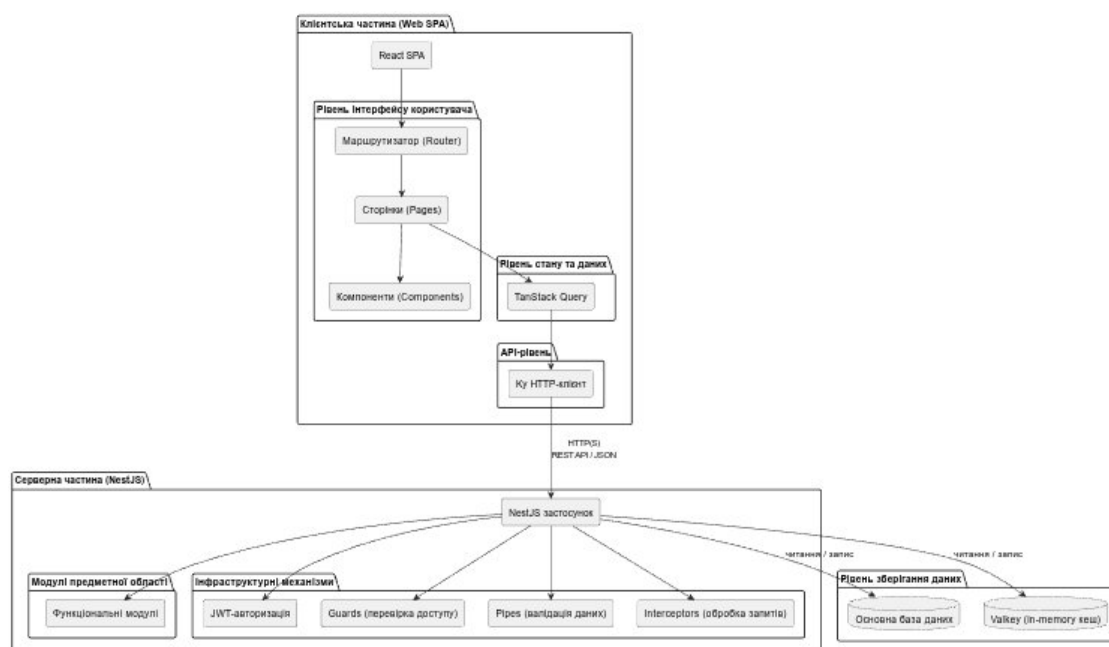


Рис. 1.4. Діаграма архітектури додатка

1.4. Обґрунтування вибору інструментальних засобів

Для реалізації серверної частини вебплатформи було обрано фреймворк NestJS, який побудований на базі Node.js [19] з використанням Express та мови програмування TypeScript [26]. Такий підхід дозволяє реалізувати модульну архітектуру проєкту, у межах якої логіка взаємодії з окремими сутностями бази даних ізольована одна від одної [4]. Це спрощує підтримку коду, його масштабування та подальший розвиток системи.

NestJS надає розширений інструментарій для обробки HTTP-запитів ще до їх потрапляння до контролерів, зокрема за допомогою guards, interceptors та middleware. Це дозволяє централізовано реалізувати механізми авторизації та автентифікації на основі JWT, контроль доступу до ресурсів і валідацію даних. Використання статичної типізації TypeScript сприяє виявленню значної кількості помилок на етапі розробки, що підвищує стабільність і безпечність серверної частини.

З метою підвищення продуктивності та зменшення навантаження на основну базу даних у серверній частині використовується Valkey як сховище даних в пам'яті, сумісне з Redis. Valkey застосовується для кешування часто використовуваних або ресурсоємних запитів, зокрема агрегованої статистики, списків сутностей та результатів обчислень. Такий підхід дозволяє значно

скоротити час відповіді сервера та забезпечити стабільну роботу платформи за умов високої кількості одночасних запитів.

Клієнтська частина застосунку реалізована на основі React із використанням TypeScript та збірника Vite [29], що дозволяє створити сучасний Single Page Application. Використання React забезпечує компонентний підхід до побудови інтерфейсу, а механізм віртуального DOM оптимізує процес оновлення сторінки, зменшуючи навантаження на браузер та покращуючи загальний користувацький досвід [2].

Для стилізації інтерфейсу було обрано Tailwind CSS [24], який використовується як основне джерело стилів. Tailwind CSS надає утилітарний підхід до верстки, що дозволяє швидко створювати адаптивні та консистентні інтерфейси без необхідності написання великої кількості власних CSS-файлів. Це сприяє підтримуваності коду та зменшує ризик конфліктів стилів.

На базі Tailwind CSS у проєкті використовується бібліотека компонентів shadcn/ui, яка надає набір доступних, стилістично узгоджених і легко кастомізованих UI-компонентів. Компоненти shadcn/ui інтегруються безпосередньо у код проєкту, що дозволяє повністю контролювати їхню логіку та зовнішній вигляд. Такий підхід поєднує переваги готових UI-рішень із гнучкістю власної реалізації та забезпечує єдину дизайн-систему застосунку [30].

Взаємодія між клієнтською та серверною частинами реалізується за допомогою бібліотеки Ку, яка є сучасною легкою обгорткою над Fetch API. Ку відповідає за формування HTTP-запитів, налаштування базового URL, передачу заголовків авторизації та обробку відповідей сервера. Асинхронна логіка отримання та завантаження даних реалізується з використанням бібліотеки TanStack Query, яка забезпечує керування станами завантаження на клієнт (запити) та на сервер (мутації), кешування отриманих даних на клієнтській стороні та синхронізацію з сервером.

Особливу увагу під час розробки було приділено організації архітектури програмного коду. Як клієнтська, так і серверна сторони використовують

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				19

модульний підхід, за якого функціональність розділена на незалежні компоненти та сервіси. Така структура спрощує підтримку проєкту, дозволяє повторне використання окремих частин коду та зменшує складність внесення змін у майбутньому. Крім того, модульна архітектура сприяє швидшому масштабуванню системи та розширенню її функціональності.

Для забезпечення високого рівня безпеки реалізовано комплекс заходів щодо захисту даних користувачів. На стороні сервера використовуються механізми перевірки прав доступу до ресурсів, а також валідації вхідних даних за допомогою спеціалізованих інструментів NestJS. Авторизація базується на токенах JWT, що дозволяє безпечно ідентифікувати користувачів та обмежувати доступ до окремих функцій системи відповідно до їхніх ролей. Такий підхід мінімізує ризики несанкціонованого доступу до даних та внутрішніх сервісів платформи.

Важливою складовою клієнтської частини є адаптивний дизайн інтерфейсу. Завдяки використанню Tailwind CSS, усі сторінки платформи коректно відображаються на різних типах пристроїв – від мобільних телефонів до широкоекранних моніторів. Адаптивна верстка забезпечує зручний доступ до функціональності незалежно від роздільної здатності екрана та покращує загальний користувацький досвід під час роботи з порталом.

Для підвищення надійності програмного забезпечення під час розробки активно використовувалися інструменти для набору тексту та контролю якості коду. Використання TypeScript дозволило зменшити кількість потенційних помилок ще на етапі написання програмного коду, а використання інструментів форматування та статичного аналізу забезпечило єдиний стиль розробки в межах усього проєкту. Це позитивно вплинуло на читабельність коду, спростило процес налагодження та подальшої підтримки системи.

Як основну систему управління даними було обрано PostgreSQL [20], яка забезпечує високу продуктивність, підтримку великих обсягів інформації та надійність транзакцій. PostgreSQL добре інтегрується з TypeScript через ORM-

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
		№ докум.				20

рішення та підтримує складні реляційні зв'язки між таблицями, що є критично важливим для системи з великою кількістю взаємопов'язаних сутностей.

Для роботи з базою даних використовується TypeORM, який забезпечує тісну інтеграцію з NestJS та дозволяє описувати структуру таблиць і зв'язки між ними безпосередньо у коді. Використання ORM зменшує обсяг SQL-запитів, підвищує читабельність коду та спрощує процес супроводу й масштабування проєкту.

Для зберігання мультимедійних ресурсів у межах вебплатформи використовується Firebase Storage [8], який забезпечує безпечне та масштабоване хмарне сховище файлів. Даний сервіс застосовується для збереження зображень, медіаматеріалів та інших статичних ресурсів і дозволяє зменшити навантаження на серверну частину, підвищивши загальну стабільність і продуктивність системи. Для полегшення авторизації користувачів також буде використано авторизацію за допомогою Google OAuth, що значно полегшить реєстрацію та вхід для користувачів.

У результаті вибору технологічного стеку було сформовано сучасну та гнучку архітектуру, що відповідає актуальним вимогам до розробки вебзастосунків. Використані технології забезпечують високу швидкодію, безпечність обробки даних та можливість подальшого розвитку проєкту шляхом інтеграції нових сервісів і функціональних модулів без необхідності кардинальної перебудови існуючої системи.

Висновки до першого розділу

У першому розділі визначено основні завдання та технічні вимоги до розробки вебплатформи автоспортивного інформаційно-статистичного порталу. Проведено аналіз існуючих ресурсів, що надають доступ до статистики, результатів змагань та медіаконтенту, з метою оцінки їх функціональних можливостей, переваг і обмежень. На основі отриманих результатів сформовано вимоги до функціональності, структури та архітектури розроблюваної системи.

Архітектура платформи побудована за клієнт-серверною моделлю з урахуванням вимог до масштабованості, продуктивності та безпеки даних.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				21

Серверна частина реалізована на базі NestJS, що забезпечує модульну організацію коду та спрощує подальше розширення функціональності. Для підвищення швидкодії використовується Valkey як кеш у пам'яті, що дозволяє обробляти великі масиви статистичних даних про пілотів, команди, боліди та події без додаткового навантаження на основну базу даних. Клієнтська частина розроблена з використанням React та збірника Vite, що дозволяє створити сучасний SPA. Взаємодія клієнта з сервером реалізована через Ку та TanStack Query, що забезпечує асинхронне отримання даних, їх кешування та ефективне відображення у користувацькому інтерфейсі. Для стилізації застосовано Tailwind CSS, а готові UI-компоненти реалізовано через shadcn/ui, що дозволяє створювати адаптивний та сучасний дизайн без конфліктів стилів. Для зберігання та обробки мультимедійних матеріалів (зображень пілотів та статей) використовується Firebase Storage, що забезпечує надійне хмарне зберігання файлів, швидкий доступ до них і зменшує навантаження на серверну частину системи. Також буде використано авторизацію за допомогою Google OAuth для полегшення реєстрації/входу користувачів. Запропонований технологічний стек та архітектурні рішення забезпечують гнучку, масштабовану та безпечну платформу, готову до подальшого розвитку, інтеграції нових функціональних модулів і сторонніх сервісів, а також ефективної роботи з великою кількістю користувачів та даних у режимі реального часу.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				22

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Процес аналізу вимог передбачає визначення та детальне описування функціональних і нефункціональних характеристик ПЗ. На цьому етапі оцінюється час відгуку системи на запити користувачів, моделюються основні сценарії використання та розробляється архітектурна модель, що описує структуру компонентів і принципи їх взаємодії. Такий підхід дозволяє глибше зрозуміти потреби цільової аудиторії та сформулювати технічні вимоги, необхідні для створення продукту. У таблиці 2.1 детально розписано наявні ролі користувачів.

Таблиця 2.1

Ролі користувачів

Користувач	Користувач є базовою роллю системи та має доступ до основного функціоналу порталу. Неавторизовані користувачі можуть переглядати загальнодоступну інформацію, зокрема результати гонок, статистику та статті. Після реєстрації та авторизації користувач отримує можливість взаємодії з контентом, зокрема залишати коментарі, брати участь в обговореннях та кастомізувати свій профіль Користувач не має доступу до адміністративних функцій або можливості змінювати системні дані
Редактор	Редактор відповідає за створення та редагування контенту, зокрема статей і новин. Дана роль передбачає доступ до інструментів управління контентом. Редактор має можливість створювати та редагувати нові статті, а також додавати медіа-матеріали. Редактор не має доступу до системних налаштувань платформи або управління користувачами.
Адміністратор	Адміністратор має повний доступ до функціоналу системи та відповідає за її управління, налаштування та підтримку. Адміністратор має можливість управляти користувачами (створення / редагування), призначати ролі, керувати контентом (у тому числі редагування та видалення будь-яких матеріалів) та налаштовувати інтеграції та системні параметри; Адміністратор несе відповідальність за стабільність, безпеку та коректне функціонування платформи.

В таблиці 2.2 описано основний функціонал користувачів за ролями.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.	I			23

Функціонал користувачів

Роль	Найменування	Формулювання
Користувач	Авторизація	Дозволяє вводити власні облікові дані для отримання доступу до системи.
	Взаємодія з іншими користувачами	Дозволяє брати участь в обговореннях у коментарях та тредах.
	Перегляд статей та інформації	Дозволяє переглядати статті, новини, інформацію про події, команди, пілотів, автомобілі, дисципліни та статистику.
	Перегляд обговорень	Дозволяє відкривати існуючі дискусії та перечитувати коментарі.
	Написання коментарів	Дозволяє залишати коментарі під статтями або в дискусіях.
	Створення обговорення	Дозволяє ініціювати нові теми для обговорення.
Редактор	Кастомізація профілю	Дозволяє змінювати аватар та біографію профілю.
	Створення та редагування статей	Дозволяє створювати нові статті, редагувати текст, додавати медіафайли, оновлювати інформацію та публікувати матеріали.
Адміністратор	Модерація контенту статей	Дозволяє вичитувати, коригувати інформаційні матеріали, додавати або змінювати дані про дисципліни, команди, події та пілотів.
	Керування правами доступу	Дозволяє змінювати ролі користувачів (user / editor / admin).
	Редагування профілів користувачів	Дозволяє змінювати дані профілю будь-якого користувача, зокрема при порушенні правил або потребі виправлення інформації.
	Видалення користувачів	Дозволяє повністю видаляти користувачів разом із прив'язаними даними (статті, зображення, коментарі).
	Керування контентом та системними сутностями	Дозволяє створювати, редагувати або видаляти сутності системи, а саме події, статті, дисципліни, водіїв тощо.
	Загальний контроль системи	Дозволяє контролювати працездатність сервісів, кешування, інтеграції з Firebase [7], Valkey та ін.

Отже, створення діаграми використання є одним із ключових етапів проєктування ПЗ. Це дозволяє структурувати вимоги користувачів та перетворити їх на конкретні функціональні завдання для системи. Аналіз сценаріїв взаємодії на ранніх етапах розробки знижує ризики, допомагає уникнути логічних помилок та гарантує відповідність функціоналу визначеним технічним вимогам.

Об'єктно-орієнтована структура системи базується на модульному підході, що використовується в NestJS. Код поділений на окремі модулі, кожен з яких відповідає за роботу з певною сутністю або групою функцій. Кожен модуль включає контролер, сервіс, DTO, сутність та файл модуля. Контролер обробляє HTTP-запити, сервіс реалізує бізнес-логіку, DTO відповідають за типізацію даних, а сутності відображають структуру таблиць у базі даних. Такий підхід забезпечує чітку архітектуру, спрощує обслуговування коду та дозволяє масштабувати систему без шкоди для її цілісності.

Для представлення структури системи та взаємодії між її компонентами використовується діаграма класів, яка ілюструє взаємозв'язки між основними модулями та їхніми сутностями.

Модуль `users` відповідає за управління користувачами, включаючи зберігання даних акаунтів і виконання базових операцій. Модуль `auth` реалізує механізми автентифікації та авторизації, включаючи реєстрацію, вхід і роботу з ролями користувачів. Модуль `articles` забезпечує створення, редагування та відображення статей і новин. Модуль `paragraphs` надає можливість створювати абзаци для розподілу даних до статей. Модуль `comments` відповідає за систему коментарів і взаємодію користувачів у межах контенту. Модуль `threads` реалізує функціонал тредів і обговорень між користувачами. Модуль `drivers` відповідає за зберігання та обробку інформації про пілотів. Модуль `disciplines` зберігає інформацію про автоспортівні дисципліни. Модуль `statistics` реалізує обчислення, збереження та відображення статистичних даних, а також надання інформації про сезони перегонів за дисциплінами (включно з самими перегонами)

Таким чином, обраний підхід до побудови системи підвищує її керованість та стабільність під час подальшої розробки. Розділення функціональних частин дозволяє ізолювати зміни в окремих компонентах та зменшує ризик впливу на інші частини системи. Це спрощує супровід коду, пришвидшує внесення змін та робить систему більш передбачуваною в експлуатації.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				25

2.2. Розробка бази даних системи

Для зберігання та структурування даних у рамках платформи було обрано PostgreSQL, яка є сучасною реляційною СУБД з відкритим кодом. Цей вибір обумовлений необхідністю забезпечити високу надійність, підтримку транзакцій ACID та ефективну обробку складних взаємозв'язків між об'єктами, що є критично важливим для порталу про автоспорт, який базується на інформації та статистиці [6].

PostgreSQL забезпечує стабільну продуктивність під навантаженням та підтримує масштабування у міру зростання обсягів даних. Додатковою перевагою є зручна інтеграція з бібліотекою TypeORM, що дозволяє визначати структуру бази даних безпосередньо в коді додатка, забезпечуючи типізацію та спрощуючи обробку даних. БД виступає основним сховищем системної інформації та забезпечує зберігання даних користувачів, механізмів контролю доступу, статей, обговорень, статистичної інформації, а також посилань на мультимедійні матеріали.

Для забезпечення цілісності даних застосовуються обмеження та зв'язки між таблицями, що допомагає уникнути логічних помилок та дублювання інформації [12]. З метою оптимізації продуктивності використовуються індекси, зокрема для складних запитів, що передбачають статистику, пошук та фільтрування даних [13].

Варто відзначити можливість використання розширених функцій PostgreSQL, зокрема представлень та збережених запитів, що дозволяє інкапсулювати складну логіку пошуку даних на рівні БД. Це спрощує реалізацію аналітичних запитів у системі та зменшує навантаження на серверну частину програми.

На рис. 2.1 зображено схему БД порталу з 20 таблицями.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				26

продовж. табл. 2.3

roles	enum (user, editor, admin)[[]]	Масив ролей користувача
profile_img_url	text	Посилання на аватар користувача
is_verified	boolean	Чи підтверджена пошта
created_at	timestamp	Дата створення акаунта
updated_at	timestamp	Дата останнього оновлення
favorite_driver_id	int (FK)	ID улюбленого водія

Таблиця articles визначає основну структуру статей, а також вказує авторство користувача та дисципліну, до якої вона відноситься (табл. 2.4).

Таблиця 2.4

Опис таблиці articles

Назва	Тип	Опис
id	int	Унікальний ID статті
title	text	Заголовок статті
preview_text	text	Прев'ю (короткий опис) статті
main_image_url	text	Посилання на головне зображення статті
author_id	int (FK)	ID автора статті
status	enum (draft, published)	Статус публікації
created_at	timestamp	Дата створення статті
updated_at	timestamp	Дата останнього оновлення статті
deleted_at	timestamp	Дата останнього видалення статті
discipline_id	int (FK)	ID дисципліни, на яку посилається стаття

Таблиця threads містить інформацію про всі обговорення на порталі, а також категорію, до якої кожне обговорення відноситься, та його автора (табл. 2.5).

Таблиця 2.5

Опис таблиці threads

Назва	Тип	Опис
id	int	Унікальний ID треду
title	text	Заголовок треду
description	text	Основний текст треду
author_id	int (FK)	ID автора треду
category_id	Int (FK)	ID категорії треду
created_at	timestamp	Дата створення треду
updated_at	timestamp	Дата останнього оновлення треду

Таблиця thread_categories визначає наявні категорії для обговорень задля кращого групування (табл. 2.6).

Таблиця 2.6

Опис таблиці thread_categories

Назва	Тип	Опис
id	int	Унікальний ID категорії
name	text	Заголовок категорії
description	text	Опис категорії
color	text	Колір категорії
icon	text	Кодовий ID іконки категорії
short_code	text	Скорочений код категорії

Таблиця images виступає основною таблицею для подальшого зберігання інформацію про фотофайли та їх метадані (табл. 2.7).

Таблиця 2.7

Опис таблиці images

Назва	Тип	Опис
id	uuid	Унікальний ID зображення
file_name	text	Назва зображення
file_type	text	Тип зображення
bytes_size	int	Розмір зображення в байтах
url	text	Посилання на зображення
created_at	timestamp	Час завантаження зображення

Таблиця user_images є таблицею-посередником між таблицями users та images, дозволяючи користувачам зберігати декілька зображень профілю (табл. 2.8).

Таблиця 2.8

Опис таблиці user_images

Назва	Тип	Опис
user_id	Int (FK)	ID користувача
image_id	Uuid (FK)	ID зображення
created_at	timestamp	Час створення аватару

Таблиця comments дозволяє зберігати коментарі з указанням авторства користувачем, батьківського коментаря та інформацію про сутність, до якої вона належить (табл. 2.9).

Таблиця 2.9

Опис таблиці comments

Назва	Тип	Опис
id	int	Унікальний ID коментаря
content	text	Вміст коментаря
author_id	int (FK)	ID автора коментаря
parent_id	int	ID батьківського коментаря
entity_source	enum (article, thread)	Тип джерела коментаря
entity_id	int	ID джерела коментаря
created_at	timestamp	Час створення коментаря

Таблиця circuits містить інформацію про наявні траси, побудовані спеціально для перегонів (табл. 2.10).

Таблиця 2.10

Опис таблиці circuits

Назва	Тип	Опис
id	int	Унікальний ID траси
name	text	Назва траси
description	text	Опис (біографія) траси
location	text	Місце розташування траси
other_names	text[]	Масив інших назв траси
length	numeric	Довжина траси в кілометрах
capacity	int	Місткість глядачів на трасі

Таблиця disciplines зберігає дані автоспортивних дисциплін, які є на порталі (табл. 2.11).

Таблиця 2.11

Опис таблиці disciplines

Назва	Тип	Опис
id	int	Унікальний ID дисципліни
title	text	Назва дисципліни
main_img_url	text	Посилання на логотип дисципліни
background_color	text	Фоновий колір категорії
short_code	text	Скорочений код категорії

Таблиця driver_disciplines є таблицею-посередником між таблицями drivers та disciplines задля відображення декількох дисциплін, у яких змагається водій (табл. 2.12).

Опис таблиці driver_disciplines

Назва	Тип	Опис
driversId	int (FK)	ID пілота
disciplinesId	int (FK)	ID дисципліни

Таблиця driver_images є таблицею-посередником між таблицями drivers та images, яка є створеною задля реалізацію галереї водія (табл. 2.13).

Таблиця 2.13

Опис таблиці driver_images

Назва	Тип	Опис
id	int	Унікальний ID зображення пілота
imageUrl	text	Посилання на зображення пілота
order	int	Індекс зображення пілота
driverId	int (FK)	ID пілота

Таблиця drivers містить основну інформацію про водіїв та їх біографію (табл. 2.14).

Таблиця 2.14

Опис таблиці drivers

Назва	Тип	Опис
id	int	Унікальний ID пілота
first_name	text	Ім'я пілота
last_name	text	Прізвище пілота
country	text	Країна громадянства пілота
date_of_birth	timestamp	Дата народження пілота
profileImgUrl	text	Посилання на основне фото пілота
biography	text	Біографія пілота
created_at	timestamp	Дата додавання пілота
updated_at	timestamp	Дата останнього оновлення інформації про пілота
number	int	Особистий номер пілота

Таблиця events зберігає дані про всі події, які стосуються автоспорту (табл. 2.15).

Таблиця 2.15

Опис таблиці events

Назва	Тип	Опис
id	int	Унікальний ID події
title	text	Назва події
description	text	Опис події
main_img_url	text	Титульне зображення події
location	text	Місце проведення події
created_at	timestamp	Дата додавання події
updated_at	timestamp	Дата останнього оновлення інформації про подію
start_date	timestamp	Дата початку події
end_date	timestamp	Дата кінця події

Таблиця paragraphs дає можливість статтям детальніше відображати свій вміст, зберігаючи в собі абзаци та даючи їм назву та порядок, а також відношення до потрібної статті (табл. 2.16).

Таблиця 2.16

Опис таблиці paragraphs

Назва	Тип	Опис
id	int	Унікальний ID абзацу
content	text	Вміст абзацу
order	text	Індекс абзацу
created_at	timestamp	Дата створення абзацу
updated_at	timestamp	Дата останнього оновлення абзацу
article_id	Int (FK)	ID статті, до якої належить абзац
title	text	Назва абзацу

Таблиця password_reset_requests дозволяє відстежувати всі запити скидання паролю (табл. 2.17).

Таблиця 2.17

Опис таблиці password_reset_requests

Назва	Тип	Опис
id	uuid	Унікальний ID запиту
updated_at	timestamp	Дата останнього оновлення запиту
user_id	int (FK)	ID автора запиту

Таблиця race_classifications містить інформацію про всі фініші (останні кола) в перегонах з іншими важливими даними, а саме причиною можливого сходу, час фінішу та чи є коло найшвидшим (табл. 2.18).

Таблиця 2.18

Опис таблиці race_classifications

Назва	Тип	Опис
id	int	Унікальний ID класифікації
timeMs	int	Час фінішу кола в мілісекундах
finish_position	int	Позиція пілота при фініші гонки
early_end_reason	enum (DSQ, DNS, DNF)	Тип причин раннього завершення гонки для пілота
race_event_id	int (FK)	ID гоночної події
is_fastest_lap	boolean	Чи дане коло найшвидше
race_entry_id	int (FK)	ID боліда-учасника
driver_id	Int (FK)	ID пілота

Таблиця race_entries зберігає інформацію про наявні боліди, які були у перегонах, а також сезон та дисципліну, до якої вони відносяться (табл. 2.19).

Таблиця 2.19

Опис таблиці race_entries

Назва	Тип	Опис
id	int	Унікальний ID учасника
name	text	Назва учасника
entry_number	int	Номер учасника
car	text	Назва боліду учасника
season_id	int (FK)	ID сезону
discipline_id	int (FK)	ID дисципліни

Таблиця race_events містить дані про гоночні події (табл. 2.20).

Таблиця 2.20

Опис таблиці race_events

Назва	Тип	Опис
id	int	Унікальний ID гонки
name	text	Назва гонки
description	text	Опис гонки
start_date	timestamp	Дата початку гонки
end_date	timestamp	Дата кінця гонки

продовж. табл. 2.20

circuit_id	int (FK)	ID траси
season_id	int (FK)	ID сезону
event_stage	int	Номер гонки в сезоні
laps	int	Кількість кіл у гонці

Таблиця race_seasons зберігає інформацію про кожен сезон автоспортивної дисципліни (табл. 2.21).

Таблиця 2.21

Опис таблиці race_seasons

Назва	Тип	Опис
id	int	Унікальний ID сезону
season_year	int	Рік сезону
discipline_id	int	ID дисципліни
stages	int	Кількість етапів у сезоні

Таблиця refresh_tokens зберігає у собі refresh-токени JWT кожного користувача та дає можливість створити обмеження у можливу кількість сеансів входу (табл. 2.22).

Таблиця 2.22

Опис таблиці refresh_tokens

Назва	Тип	Опис
id	int	Унікальний ID токена
refresh_token	varchar	Токен
created_at	timestamp	Дата додавання токена
updated_at	timestamp	Дата останнього оновлення токена
user_id	int (FK)	ID користувача

2.3. Проектування та реалізація алгоритмів роботи системи

Алгоритмічна структура є ключовим елементом, що визначає ефективність та коректну роботу платформи. Вона охоплює обробку запитів користувачів, перевірку даних, управління авторизацією, формування статистики, а також взаємодію з контентом та обговореннями. Для формалізації операційної логіки системи та опису сценаріїв взаємодії були використані діаграми активності. Такий підхід дозволяє виявляти потенційні вузькі місця,

оптимізувати алгоритми обробки даних та покращувати користувацький досвід завдяки ефективнішій взаємодії з системою.

Окрім опису основних бізнес-процесів, діаграми діяльності дозволяють відображати перевірки умов, альтернативні сценарії виконання та можливу обробку помилок. Це дозволяє аналізувати поведінку системи не лише за стандартних умов використання, але й у випадках некоректного введення даних, відсутності необхідних прав доступу або помилок під час взаємодії із зовнішніми сервісами.

Діаграми послідовностей та взаємодії компонентів є важливими інструментами для документування та аналізу поведінки системи. Їх можна використовувати для демонстрації того, як система реагує на конкретні події, як компоненти співпрацюють для виконання певного завдання та як компоненти системи взаємодіють між собою.

На рис. 2.2 зображено послідовність процесів для створення статті. Користувач обирає опцію “Створення статті”, після чого йому відкривається форма створення чернетки статті, в якій він може вказати ім’я статті та її прев’ю-текст. При успішному створенні користувач автоматично переходить на сторінку чернетки де може редагувати/завантажити інформацію (назва, прев’ю-текст, дисципліна, а також головне зображення статті), а також працювати з абзацами (додання, редагування та видалення). Користувач може зберегти зміни у чернетці та опублікувати статтю (оновити її статус до PUBLISHED), після цього користувача буде переправлено на загальну сторінку статті.

Під час кожного етапу система перевіряє правильність введених даних та права доступу користувачів. Це запобігає створенню некоректного контенту та гарантує відповідність дій користувача його ролі в системі. У разі помилки користувач отримує відповідне повідомлення із зазначенням причини відмови у виконанні операції.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				35

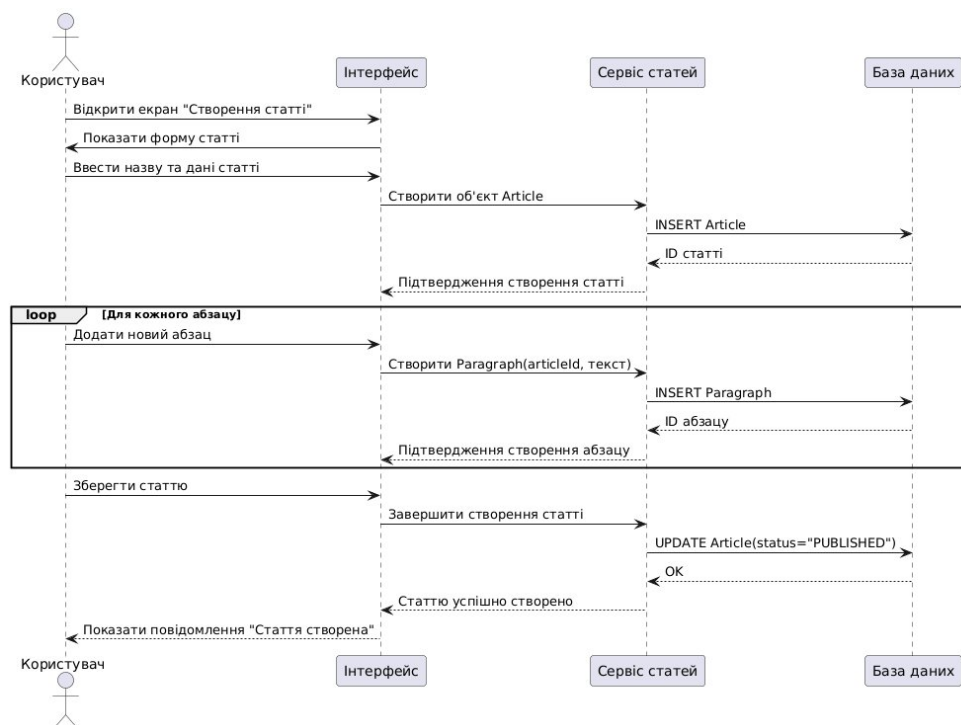


Рис. 2.2. Діаграма послідовності для створення статті

Далі зображено діаграму послідовності завантаження нового фото для профілю користувача на рис. 2.3.

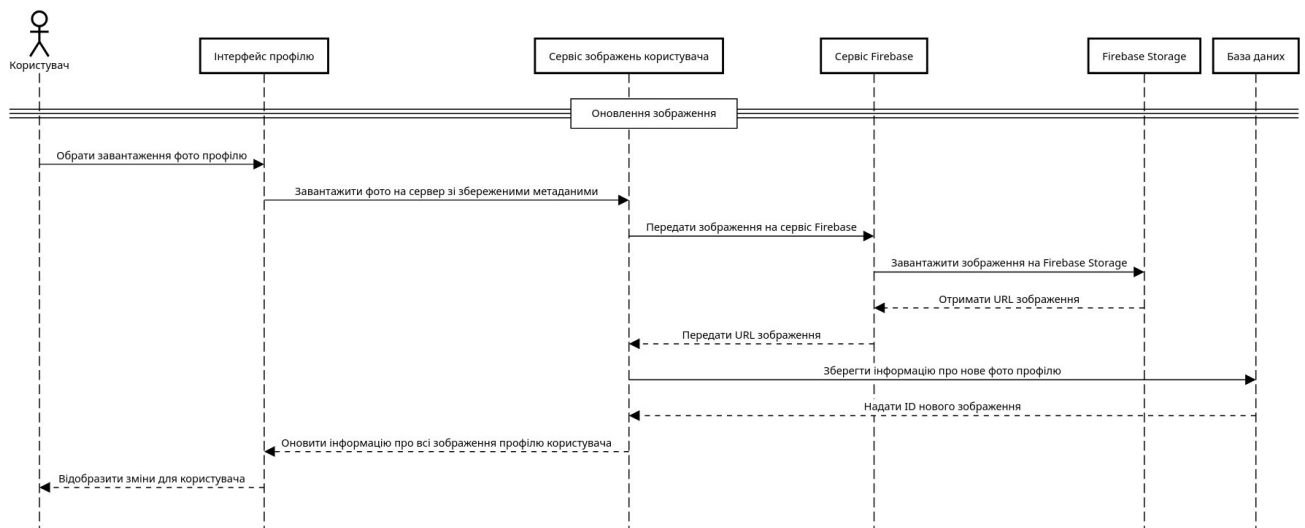


Рис. 2.3. Діаграма послідовності для завантаження фото профілю

Користувач може переглянути всі свої фото профілю, а також додати нове фото. Для цього йому потрібно обрати опцію “Завантажити нове фото” з контекстного меню користувача, після чого йому відкриється діалог завантаження фото. Користувач обирає потрібне фото та завантажує його на сервер. На сервері сервіс фото профілю переправляє його на сервіс сховища Firebase, який далі опрацьовує саме завантаження на сховище. Після успішного завантаження фото на сховище сервіс повертає URL даного фото, яке в

подальшому бере сервіс фото профілю та зберігає його у БД з іншими метаданими файлу, повертаючи ID зображення. Далі ця інформація переходить на UI профілю, де вже оновлюється список зображень профілю користувача.

Останньою діаграмою є діаграма компонентів API (бекенд-частини) додатку на рис. 2.4. Саме API оброблює великі масиви даних та повертає їх на клієнт, а також керує авторизацією та автентифікацією користувачів, дає можливість розширити функціонал додатку та підтримувати стабільність усього порталу.

Представлення системи у вигляді діаграми компонентів дозволяє чітко визначити межі відповідальності окремих підсистем та їх взаємозв'язки. Це спрощує процес підтримки програмного забезпечення, полегшує пошук потенційних проблем під час розробки та створює основу для подальшого масштабування системи шляхом додавання нових модулів або інтеграції зовнішніх сервісів.

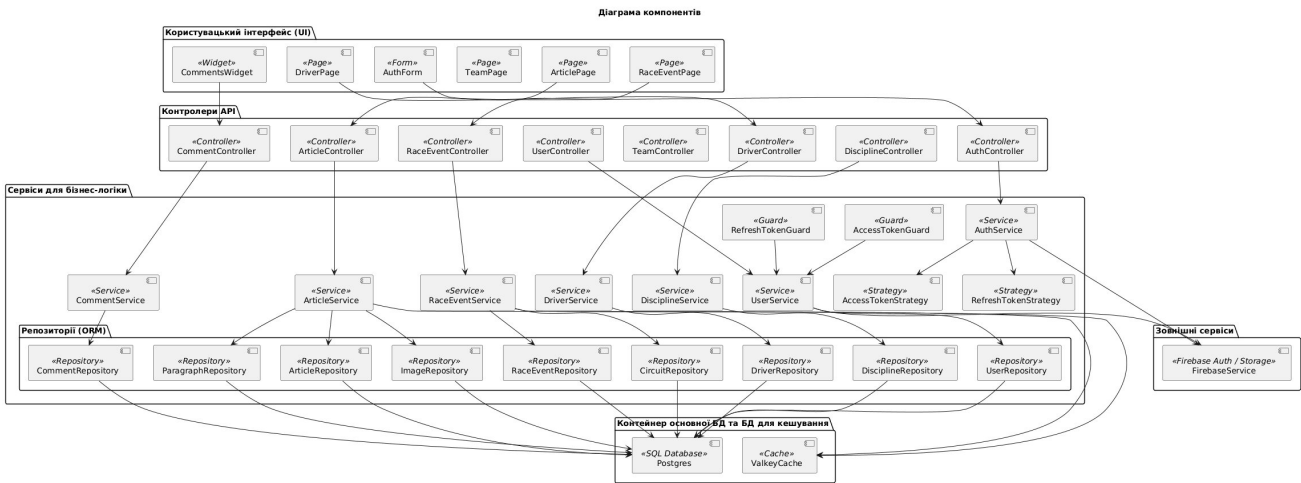


Рис. 2.4. Діаграма компонентів

Бекенд у даному випадку поділяється на контролери, які керують HTTP-запитами, сервіси бізнес-логіки та репозиторії для роботи з основною БД за допомогою ORM. Також наявні дві БД (PostgreSQL як основна БД та Valkey як мінімалістичне сховище формату “ключ-значення” для кешування даних) та зовнішні сервіси, а саме Firebase для сховища.

2.4. Реалізація функціоналу системи

Створення нової статті є одним з основних сценаріїв використання порталу. Користувач викликає POST-запит у ArticleController на створення чернетки статті (так як статус статті за замовчуванням у таблиці articles – DRAFT). Перед цим в запиті перевіряється access-токен користувача (JWT) за допомогою AccessTokenGuard, з якого береться основна інформація про користувача. Якщо токен відсутній – користувач отримує UnauthorizedException зі статусом 401. В іншому випадку система перевіряє масив ролей користувача за допомогою декоратору Roles, який звіряє задані ролі для маршруту із ролями користувача. В разі відсутності потрібних ролей користувач отримує ForbiddenException зі статусом 403.

В успішному випадку система отримує об'єкт ArticleCreateDto, в якому міститься потрібна інформація для створення чернетки – назва статті та її прев'ю-текст. Також береться ID користувача з токену для прив'язки автора до чернетки. Ця інформація переходить до ArticleService, після чого – до ArticleRepository, іншої абстракції, яка інкапсулює в собі логіку роботи з БД за допомогою TypeORM. У ArticleRepository ми викликаємо INSERT-запит до таблиці articles, після чого створюється нова стаття зі статусом DRAFT.

Перед створенням чернетки виконується автоматична перевірка вхідних даних за допомогою механізмів NestJS та бібліотеки class-validator [5]. Для полів об'єкта ArticleCreateDto встановлюються обмеження на обов'язкове заповнення, мінімальну та максимальну довжину тексту. Якщо хоча б одне з полів не відповідає встановленим вимогам, запит не передається до бізнес-логіки сервісу, а користувач отримує повідомлення про помилку, що вказує на некоректність даних. Такий підхід дозволяє забезпечити цілісність інформації ще на етапі отримання запиту сервером.

Після успішного створення чернетки користувача автоматично переносить на сторінку чернетки, де він може далі редагувати нову статтю. Наступний етап – додання іншої інформації, такої як дисципліна, до якої належить стаття, абзаци та фото статті.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				38

Дисципліни підтягуються з DisciplineController за допомогою GET-запиту, після чого вони відображаються в select-компоненті на сторінці редагування чернетки. Робота з абзацами відбувається на UI – користувач додає новий абзац, після чого додає йому назву та його вміст. Аналогічно з фото статті – користувач обирає потрібні фото за допомогою input-компоненту з типом file.

Для покращення взаємодії користувача з редагуванням статті інтерфейс розроблено таким чином, що всі зміни вносяться без перезавантаження сторінки. Додавання, видалення та редагування абзаців здійснюється динамічно на стороні клієнта, що дозволяє автору одразу бачити структуру майбутньої публікації. Такий підхід скорочує час на підготовку матеріалу та робить процес створення контенту зручнішим.

Оновлення чернетки статті відбувається за допомогою PATCH-запиту update/:id на оновлення статті. Перед цим відбувається аналогічна перевірка на токен користувача та його ролі, а також перевірка на наявний фото-файл у тілі запиту. Перевірка фото відбувається за допомогою інтерцептору FileInterceptor, який бере фото з потрібного поля тіла та додає валідацію на тип файлу. Якщо на бекенд приходить відмінний від фото тип файлу, користувач отримує BadRequestException.

У разі успішної перевірки усіх полів контроллер передає до сервісу ID статті з параметру, тіло типу ArticleEditDto та фото-файл. В сервісі відбувається перевірка на наявність фото. Якщо фото присутнє, то викликається завантаження файлу на Firebase Storage та отримання валідного URL-шляху фото за допомогою сервісу FirebaseStorageService. Отримана інформація, включно з URL зображення статті та ID дисципліни, передається до ArticleRepository. В репозиторії оновлюється інформація про статтю та створюються абзаци за допомогою іншого репозиторію ParagraphRepository.

Зберігання мультимедійних файлів у зовнішньому хмарному сховищі зменшує навантаження на серверну частину та основну базу даних. У базі даних зберігаються лише URL-посилання на завантажені ресурси, що позитивно

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				39

впливає на продуктивність системи та спрощує подальше масштабування платформи зі збільшенням кількості користувачів та контенту.

Використання механізму чернетки дозволяє користувачеві поступово формувати зміст статті без необхідності негайної публікації матеріалу. Всі внесені зміни можна багаторазово зберігати шляхом багаторазового оновлення чернетки, що забезпечує зручний процес підготовки великих статей. Завдяки цьому автор має можливість доповнювати текст, редагувати структуру абзаців та оновлювати мультимедійний контент перед остаточною публікацією матеріалу.

Важливою особливістю реалізації є розподіл відповідальності між контролерами, сервісами та репозиторіями. Контролери відповідають виключно за обробку HTTP-запитів та генерацію відповідей, сервіси реалізують бізнес-логіку застосунку, а репозиторії забезпечують взаємодію з базою даних. Такий підхід відповідає принципам багаторівневої архітектури програмного забезпечення, підвищує читабельність коду та спрощує подальшу підтримку та тестування системи.

Під час створення та редагування статей система також перевіряє наявність пов'язаних сутностей, включаючи дисципліни та авторів. Це запобігає неправильним зв'язкам між записами бази даних та забезпечує цілісність інформації, що зберігається в системі.

Останнім етапом є публікація статті. Після отримання всієї доступної інформації користувач викликає ще один PATCH-запит `publish/:id` з подальшою перевіркою користувача та його ролей. Далі, ID чернетки береться з маршруту та передається до `ArticleService`, де перевіряється наявність чернетки з таким ID. Якщо її немає, користувач отримує `NotFoundException` зі статусом 404. В іншому випадку, ID передається до репозиторію, де статус статті оновлюється з `DRAFT` на `PUBLISHED`. В результаті користувач автоматично потрапляє на сторінку опублікованої статті.

Таким чином, процес створення та публікації статей реалізовано як послідовність взаємопов'язаних етапів, що включають перевірку прав доступу,

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				40

валідацію даних, роботу з мультимедійними ресурсами та зберігання інформації в базі даних. Такий підхід забезпечує надійність системи, зберігає цілісність даних та створює зручні умови для підготовки та публікації контенту на порталі REVITUP.

Висновки до другого розділу

У другому розділі було виконано проектування архітектури portalу REVITUP та визначено основні принципи його функціонування. На основі аналізу функціональних вимог було визначено ролі користувачів (звичайний користувач, редактор та адміністратор), їх права доступу та сформовано набір варіантів використання системи. Також було спроектовано структуру даних платформи, центральними сутностями якої стали користувачі, автоспортивні дисципліни, пілоти, боліди, траси, події, інформаційні матеріали, коментарі та мультимедійний контент. Це дозволяє забезпечити як інформаційно-статистичну складову платформи, так і можливість взаємодії між користувачами та підтримки активної спільноти.

Для зберігання даних було обрано реляційну систему керування базами даних PostgreSQL. Структуру бази даних побудовано відповідно до принципів нормалізації, що дозволяє мінімізувати дублювання інформації та підтримувати цілісність даних. Для підвищення продуктивності системи передбачено використання Valkey як проміжного шару кешування статистичних даних і результатів часто виконуваних запитів.

У процесі проектування було сформовано діаграми таблиць БД, класів та послідовності, які дозволили детально описати взаємодію компонентів системи та логіку виконання основних бізнес-процесів. Зокрема було проаналізовано процеси авторизації та автентифікації користувачів, створення та редагування статей, обробки коментарів, а також роботи зі статистичними даними автоспортивних подій.

Архітектура серверної частини базується на модульному підході фреймворку NestJS, що забезпечує низьку зв'язність між компонентами та спрощує подальший розвиток системи. Взаємодія з базою даних реалізується

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				41

через TypeORM із використанням патерну Repository, що дозволяє абстрагувати логіку доступу до даних від бізнес-логіки застосунку. Клієнтська частина спроектована як Single Page Application на основі React та Vite, що забезпечує швидку роботу інтерфейсу.

Таким чином, у другому розділі було сформовано повноцінну архітектурну основу майбутньої вебплатформи, визначено структуру даних, механізми взаємодії між компонентами.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				42

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ПОРТАЛУ

3.1. Встановлення варіантів використання системи

Система порталу REVITUP розробляється для бакалаврської роботи за визначеними раніше вимогами. Для розробки архітектури системи було обрано моноліт на боці бекенду з чітким модульним поділом, готовий до винесення в мікросервіси [3]. Фронтенд та бекенд системи, а також БД (основна та для кешування), ізольовані одна від одної. Це логічно, коли портал необхідно розширити у майбутньому на інші платформи (наприклад використання API бекенду для розробки нативного мобільного додатку). Розгортання системи не передбачене, але в подальшому є цілком можливим у майбутньому на такі сервіси, як Railway або Render.

Крім того, слід зазначити, що навіть у рамках монолітної реалізації система розроблена з урахуванням контейнеризації. Використання такого підходу дозволяє в майбутньому розгортати застосунок у стандартизованому середовищі, що спрощує міграцію між різними хмарними платформами та підвищує відтворюваність середовища розробки.

Важливо зазначити, що навіть за використання монолітного підходу, проєктна система структурно близька до мікросервісної [17] архітектури завдяки чіткому розподілу обов'язків між основними модулями [18]. Кожен модуль інкапсулює власну бізнес-логіку, сервіси та доступ до даних, що дозволяє мінімізувати залежності між різними частинами системи [9]. Така організація коду забезпечує гнучкість у разі подальшого розділення окремих модулів на незалежні сервіси без суттєвих змін у загальній логіці роботи застосунку [23].

Додатковою перевагою такого підходу є спрощення командної розробки. Оскільки функціональність системи розділена на незалежні модулі, робота над окремими підсистемами може виконуватися паралельно без значного ризику конфліктів між змінами. Це позитивно впливає на супровід проєкту та пришвидшує впровадження нового функціоналу.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				43

Вебплатформа portalу побудована на основі модульної архітектури задля розбиття на основні компоненти системи. Бекенд системи побудований на NestJS, а фронтенд являє собою додаток React у поєднанні зі збірником Vite. Для зберігання даних використовуються основна БД на базі СУБД PostgreSQL, яка є надійним варіантом зберігання даних, та легка БД в пам'яті Valkey, яка грає роль кешу для оптимізації запитів та збереження ресурсів системи.

Варто відзначити використання чіткого розподілу обов'язків між клієнтською та серверною частинами, що дозволяє мінімізувати зв'язок між ними. Така структура спрощує тестування, оскільки кожен рівень системи можна тестувати незалежно, а також підвищує гнучкість при внесенні змін до бізнес-логіки або інтерфейсу користувача.

Окрему увагу під час проектування системи було приділено принципам масштабованості та розширюваності. Використання NestJS на бекенді та React на фронтенді дозволяє незалежно розвивати клієнтську та серверну частини застосунку. Це забезпечує можливість паралельної роботи різних команд розробки, а також спрощує оновлення окремих компонентів без впливу на інші частини системи. Такий підхід є особливо важливим для довготривалого розвитку платформи.

Крім того, використання сучасного технологічного стеку дозволяє ефективно використовувати інструменти автоматизації розробки та процеси CI/CD. Це забезпечує швидке тестування та внесення змін до системи, що є критично важливим для підтримки стабільності та якості коду в контексті безперервної розробки проекту.

Використання комбінації реляційної бази даних PostgreSQL та кешу Valkey в оперативній пам'яті дозволяє досягти оптимального балансу між надійністю зберігання даних та продуктивністю системи. PostgreSQL забезпечує цілісність та узгодженість даних завдяки своїй транзакційній моделі, тоді як Valkey значно зменшує кількість звернень до основної бази даних, кешуючи часто використовувані запити. Це особливо важливо для сторінок з високим навантаженням, де потрібна швидка реакція сервера.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				44

Кешування також зменшує кількість обчислень для ресурсоємних операцій, пов'язаних зі статистикою та агрегованою інформацією. В результаті навантаження на сервер зменшується, а система працює стабільніше при одночасному обслуговуванні великої кількості користувачів.

Для зберігання файлів використовується сховище Firebase як вигідний варіант надійного зберігання файлів різних типів. При завантаженні файлів бекенд отримує URL-посилання на файли, які зберігає у БД. Такий підхід забезпечує надійність системи без подальших збоїв, підтримку для створення нових модулів і розширення наявного функціоналу, а також масштабованість системи для подальшого розгортання на хмарні сервери.

Також важливо, що використання хмарного сховища зменшує навантаження на серверну інфраструктуру та підвищує доступність медіаконтенту. Це особливо важливо для систем з великою кількістю зображень та статичних ресурсів, які активно використовуються на стороні клієнта.

Крім того, слід зазначити, що обрана архітектура дозволяє легко інтегрувати нові зовнішні сервіси та API без суттєвого впливу на існуючу систему. Завдяки використанню незалежних рівнів доступу до даних та бізнес-логіки, платформа може бути адаптована до нових вимог, таких як мобільні додатки або інтеграції зі сторонніми розробниками. Це робить систему REVITUP готовою до подальшого розвитку та розширення функціональності в майбутньому.

Архітектура системи також враховує вимоги безпеки та контролю доступу. Основні механізми автентифікації, авторизації та перевірки прав користувачів реалізовані на стороні сервера, що дозволяє централізовано контролювати доступ до функціональності платформи та мінімізує ризики несанкціонованого використання системних ресурсів.

Окремою перевагою архітектури є можливість поступового масштабування функціональності без необхідності повної перебудови системи. Це дозволяє впроваджувати нові модулі поетапно, зберігаючи стабільність існуючих компонентів та мінімізуючи ризики під час оновлень.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				45

3.2. Структура інтерфейсу системи

Система автоспортивного інформаційно-статистичного порталу доступна через localhost, але у майбутньому готується розгортання системи на віддалений сервер. При потраплянні на головну сторінку (рис. 3.1) користувач бачить в основному секторі останні додані статті на портал, а також компоненти випадкових статі та водія на лівій стороні та компонент з останніми 5 тредями на правій стороні екрану. Також користувач може виконати пошук сутностей та виклик контекстного меню

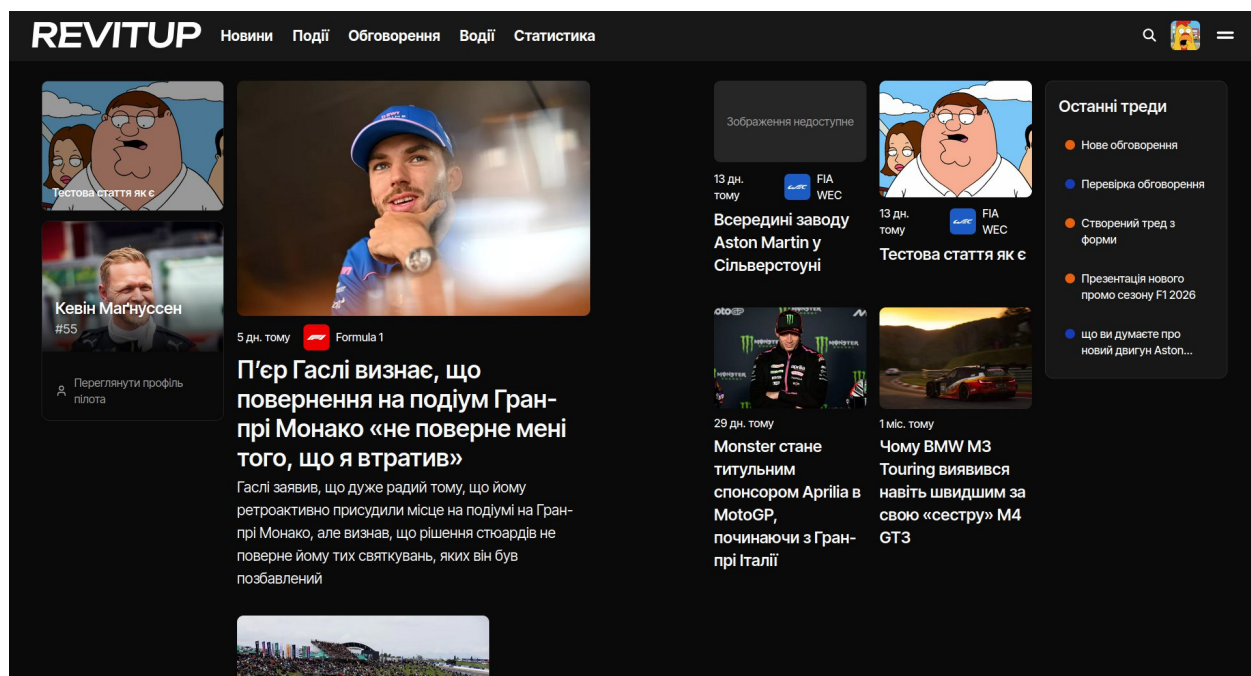


Рис. 3.1. Вигляд головної сторінки

При відкритті пошуку (рис. 3.2) користувач може бачити останні результати, а також виконати пошук потрібного об'єкту. Потім буде виконана навігація користувача до потрібного елементу, а результат пошуку збережено.

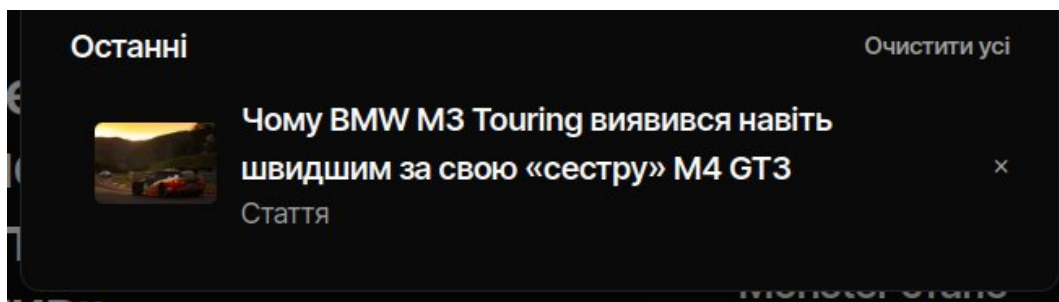


Рис. 3.2. Пошук об'єктів

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
		№ докум.				46

Система підтримує наявність перекладів декількома мовами. Для найкращого прикладу ми використовуємо українську та англійську мови. Для початку користувач відкриває контекстне меню в хедері порталу (рис. 3.3).

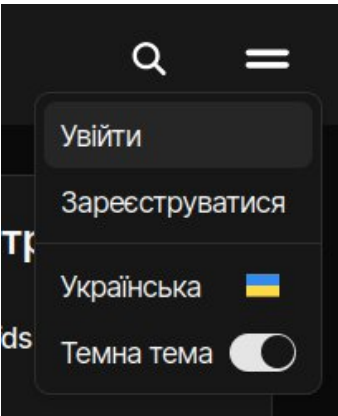


Рис. 3.3. Контекстне меню

Клікаємо на опцію з поточною мовою інтерфейсу, після чого відкриється випадаючий список доступних мов для вибору (рис. 3.4). Після вибору потрібної мови інтерфейс порталу автоматично оновлюється, а всі текстові елементи відображаються відповідно до обраної локалізації.

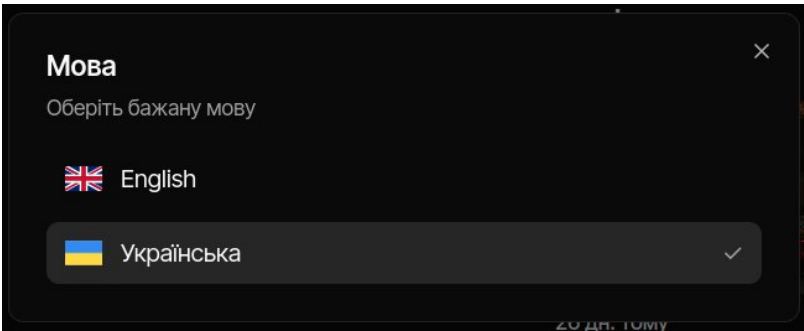


Рис. 3.4. Список мов інтерфейсу

Змінюємо мову на англійську та бачимо, як підписи в інтерфейсі змінюються. Далі повертаємось до української мови (рис. 3.5).

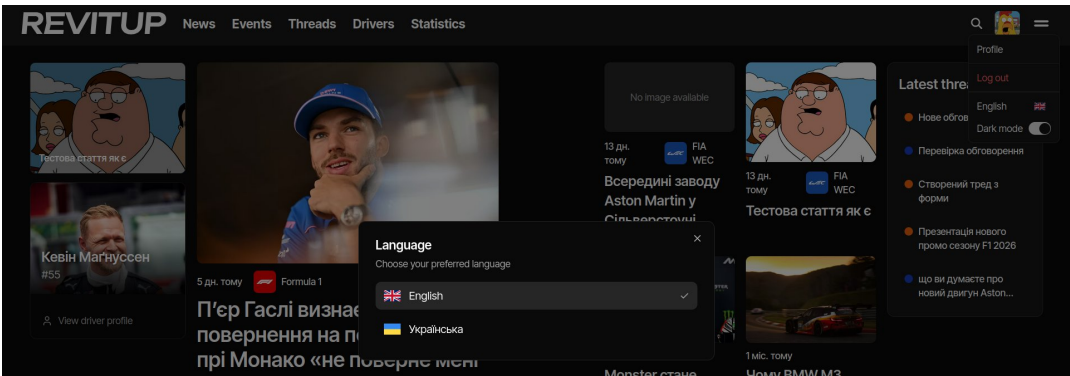


Рис. 3.5. Зміна мови інтерфейсу на англійську

Як раніше й було сказано, результати пошуку зберігаються в порталі у локальному сховищі браузера. Спочатку можемо очистити результати за допомогою кнопки “Очистити всі” (рис. 3.6).

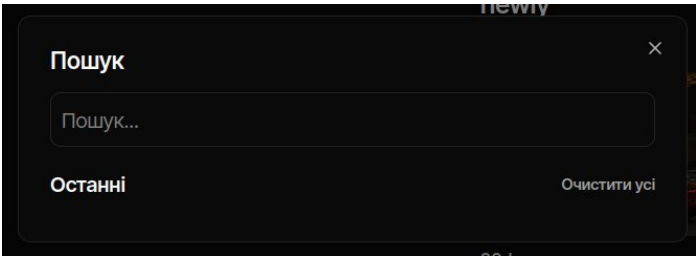


Рис. 3.6. Очищення результатів пошуку

Далі спробуємо знайти будь-який об’єкт. Для прикладу шукаємо об’єкт, який містить у назві ім’я водія П’єр Гаслі (рис. 3.7).

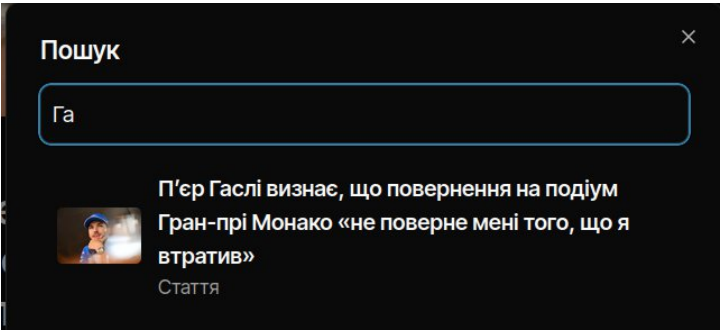


Рис. 3.7. Результати пошуку

При натисканні на об’єкт, а саме на статтю, у результатах пошуку користувач потрапляє на сторінку цієї статті (рис. 3.8). Після переходу система завантажує та відображає детальну інформацію про вибраний матеріал, забезпечуючи швидкий доступ до його повного змісту та пов’язаних з ним даних.

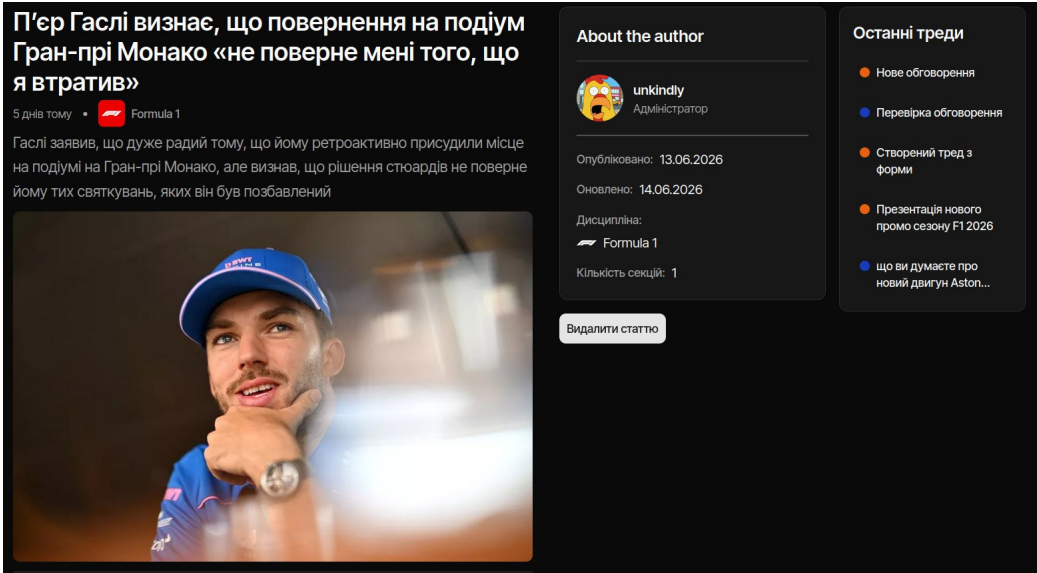


Рис. 3.8. UI статті

Після цього ми можемо спостерігати, як об'єкт статті додається до списку останніх результатів пошуку (рис. 3.9). Збереження історії пошуку покращує використання системи, пришвидшуючи доступ до останніх об'єктів з пошуку.

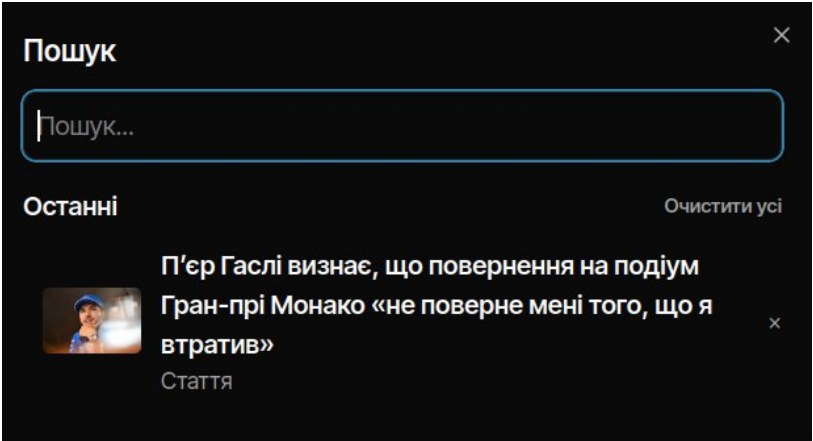


Рис. 3.9. Оновлення останніх результатів пошуку

Самі статті відображаються на сторінці “Новини”, на яку можна потрапити з хедеру portalу. UI сторінки відображає наявні статті з пагінацією та пошуком (рис. 3.10).

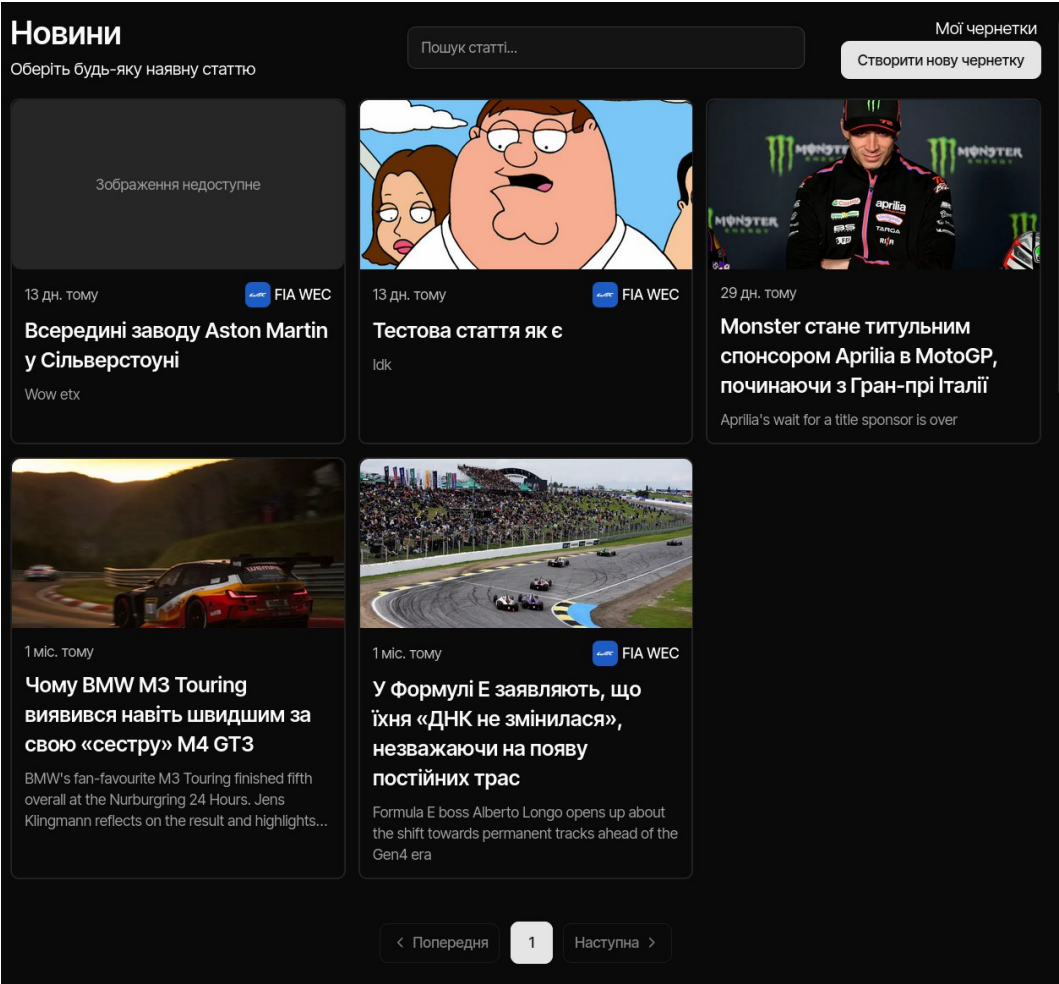


Рис. 3.10. Сторінка пошуку статей

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				49

При пошуку статей за допомогою відповідного поля змінюються і результати пошуку. Також на сторінці є пагінація, яка покращує завантаження та пошук потрібних статей (рис. 3.11).

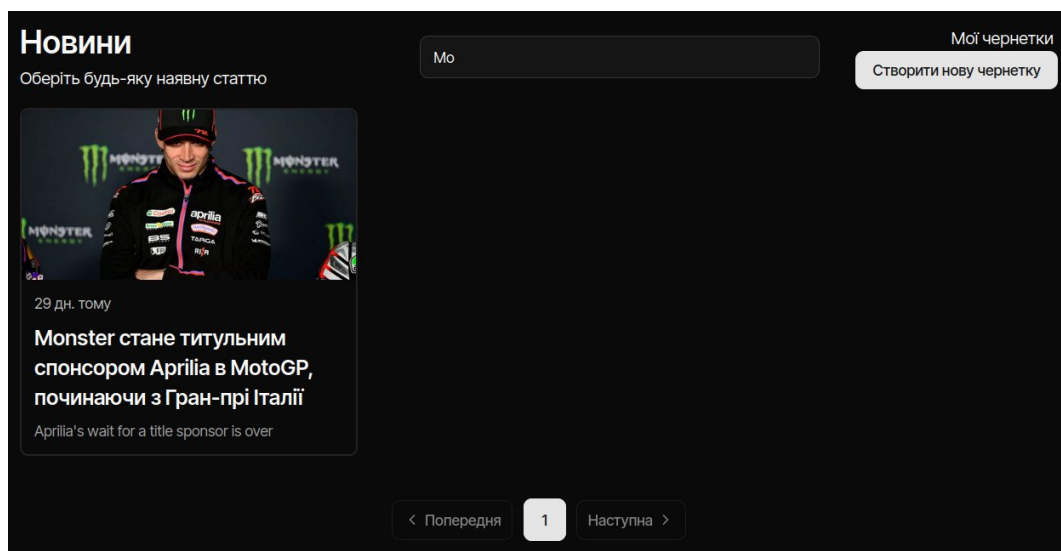


Рис. 3.11. Пошук статей

Стаття складається з назви, прев'ю-тексту, основного фото та абзаців у вигляді окремих об'єктів, а також коментарів та інформації про автора статті. Раніше вже було показано першу частину статті, тому далі можемо перейти до абзаців та коментарів. Абзаци статей складаються з назви абзацу та його вмісту (рис. 3.12).

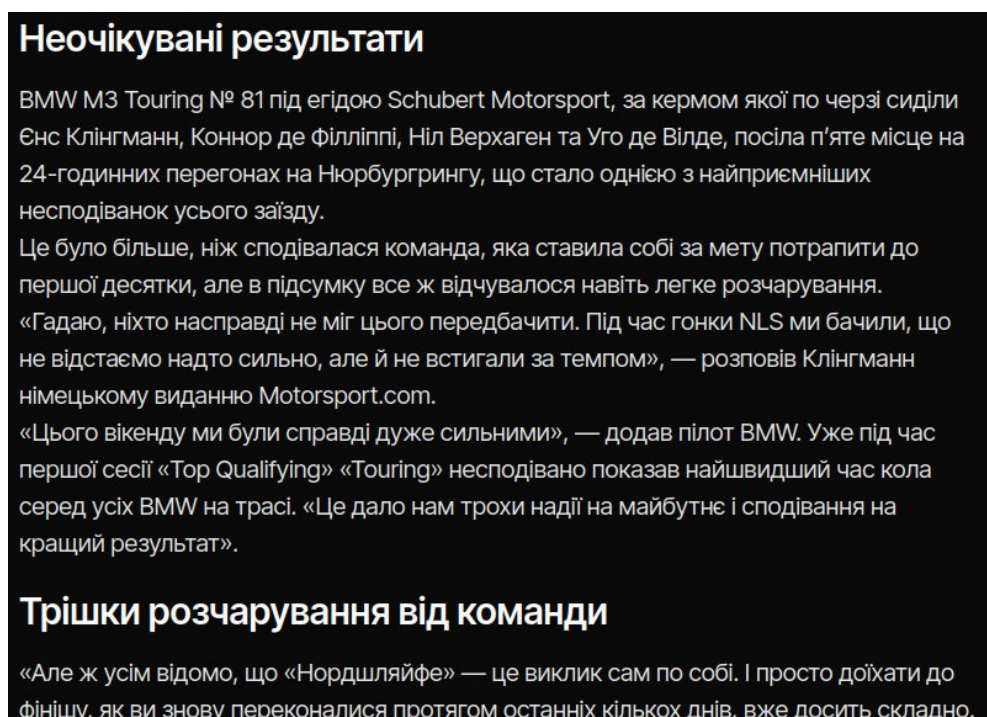


Рис. 3.12. Абзаци статей

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				50

У випадку, якщо в статті відсутні абзаци за якихось причин, відображається повідомлення про те, що вміст статті на даний момент пустий (рис. 3.13). Коректне відображення станів UI покращує використання системи користувачами.

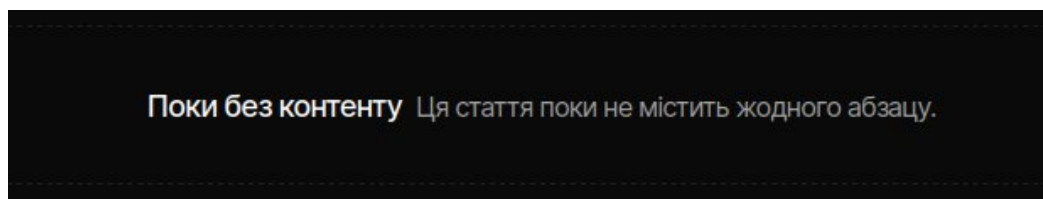


Рис. 3.13. Повідомлення про відсутність вмісту статті

До статті також можна додати коментар, але це дозволено робити лише авторизованим користувачам. Тому для початку можемо авторизуватися в портал. Наявними способами входу та реєстрації на порталі є звичайні запити до API та авторизація через обліковий запис Google. Оберемо другий варіант для цього потрібно у контекстному меню хедеру обрати варіант “Увійти”, після чого буде доступна форма входу (рис. 3.14).

Рис. 3.14. Форма входу

Далі ми обираємо потрібний обліковий запис Google для авторизації користувача (рис. 3.15). Далі відбувається вхід користувача, якщо він наявний БД, або його реєстрація в іншому випадку.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				51

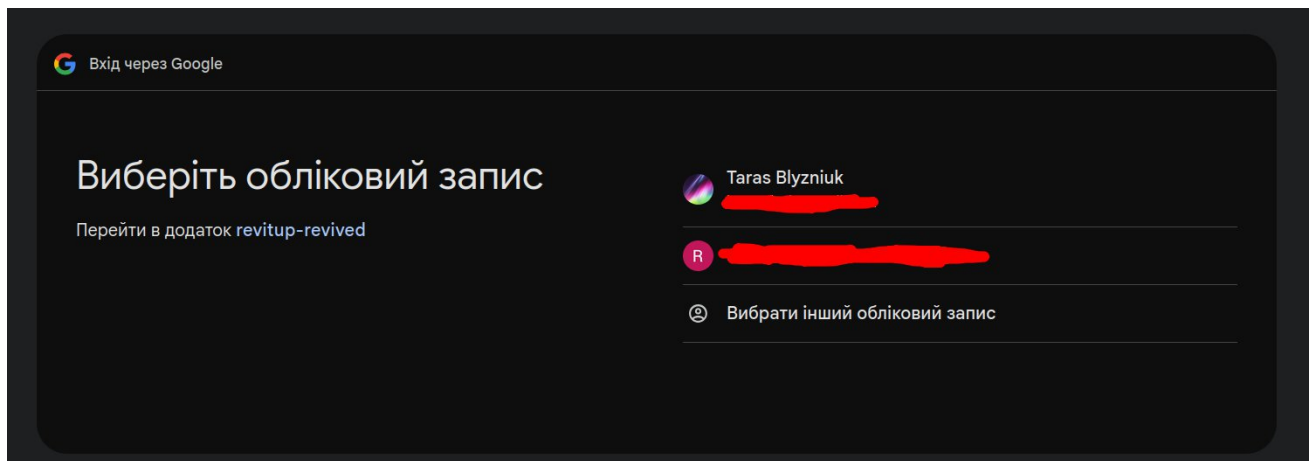


Рис. 3.15. Вибір облікового запису для авторизації

Після моментів очікування користувач потрапляє на основу сторінку, але зі змінами в шапці профілю (рис. 3.16), а саме іконкою фото профілю користувача. Це вказує на те, що користувач авторизований та може перейти до свого профілю.

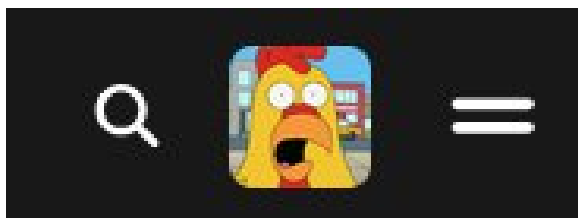


Рис. 3.16. Зміни в шапці профілю при успішній авторизації

Тепер користувач може успішно додати коментар до статті. Сам блок коментарів (рис. 3.17) окрім них відображає і їхню кількість.

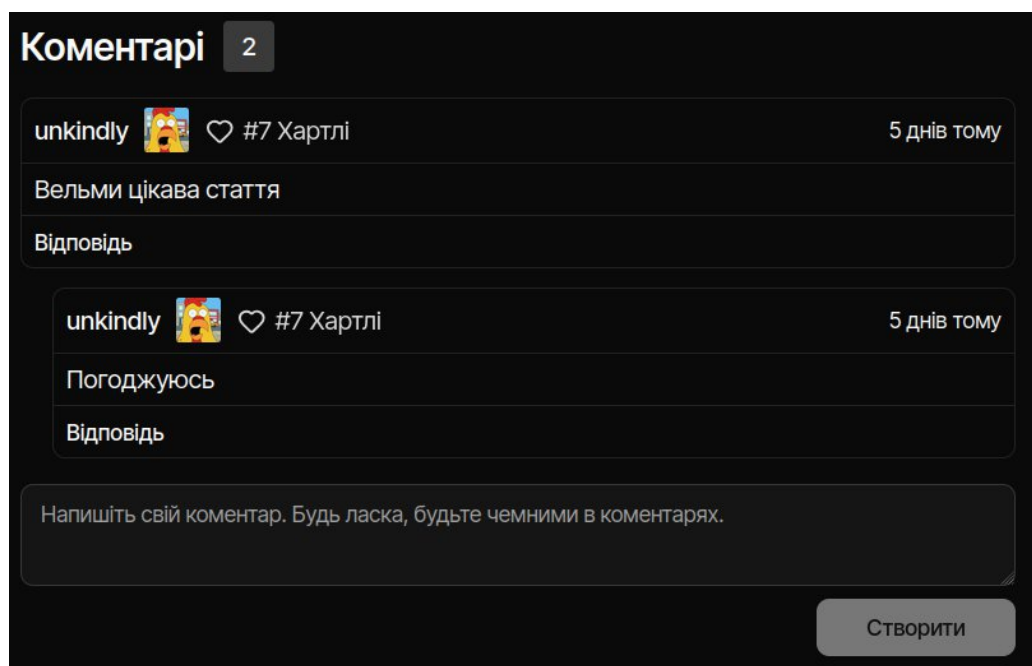


Рис. 3.17. Створення коментаря

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				52
	.	№ докум.				

Коментарі на порталі також підтримують вкладеність, тобто можливість відповідати на той чи інший коментар. При кліку на кнопку “Відповідь” на коментарі над полем написання коментаря з’являється індикатор з коментарем (рис. 3.18), на який користувач хоче відповісти.

Рис. 3.18. Індикатор коментаря

При відповіді на коментар можна бачити вкладене відображення (рис. 3.19) коментарів та оновлену кількість усіх коментарів до статті. Ієрархічна структура відображення коментарів забезпечує чітке відображення відповідей до коментарів.

Рис. 3.19. Вкладене відображення коментарів

Маючи права редактора чи адміністратора, ми можемо створити нову чернетку, перейшовши на сторінку новин. При переході ми можемо побачити (рис. 3.20) кнопку, яка відкриває форму створення статті, а також посилання на сторінку наших чернеток.

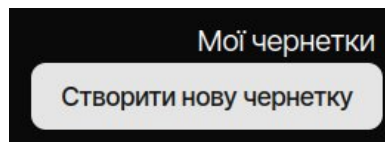


Рис. 3.20. Елементи чернеток статей

Відкривши форму (рис. 3.21), можна заповнити її та таким чином створити нову чернетку. У результаті цієї дії створиться чернетка статті зі статусом DRAFT.

Рис. 3.21. Форма створення нової чернетки

Після заповнення форми користувач автоматично переходить на сторінку чернеток (рис. 3.22). На самій сторінці чернеток з'являється сповіщення про успішне створення чернетки (рис. 3.23).

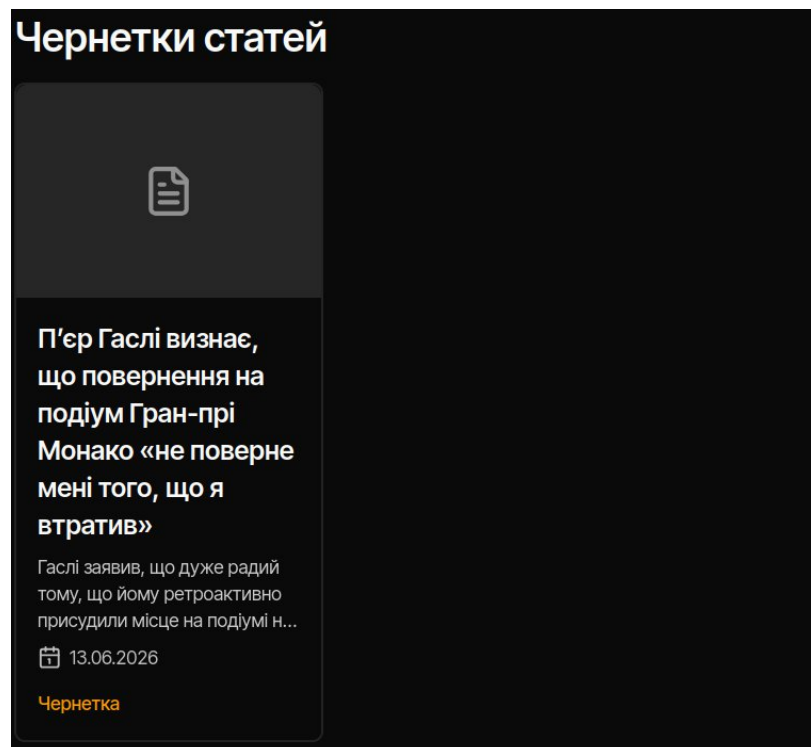


Рис. 3.22. Сторінка чернеток

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				54
	.	№ докум.				

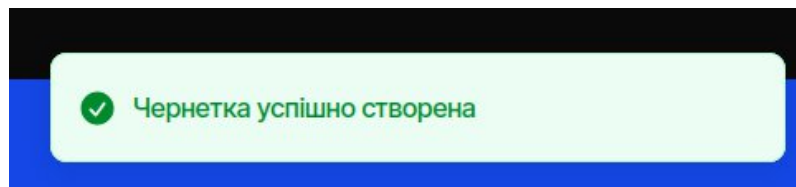


Рис. 3.23. Сповіщення про успішне створення чернетки

При навігації на сторінку чернетки ми можемо бачити її основну інформацію для редагування (рис. 3.24), а саме назву та її прев'ю-текст, дисципліну, до якої належатиме стаття, завантаження зображення статті та її абзаци. Таким чином можна розпочати створення нової статті.

Редагування чернетки

Назва статті

П'єр Гаслі визнає, що повернення на подіум Гран-прі Монако «не поверне мені того, що я втратив»

Прев'ю-текст статті

Гаслі заявив, що дуже радий тому, що йому ретроактивно присудили місце на подіумі на Гран-прі Монако, але визнає

Рис. 3.24. Редагування чернетки статті

До статті можемо обрати дисципліну з наявних (рис. 3.25), які отримуються з БД. Вони розмежовують статті за її змістом відносно автоспортивної дисципліни.

Дисципліна

- Formula 1
- FIA WEC
- Formula 1
- WRC
- Formula E

Рис. 3.25. Вибір дисципліни для статті

Користувач може обрати зображення до статті з файлового менеджера або за допомогою drag-and-drop. Після цього можна бачити обране фото (рис. 3.26) на сторінці.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				55

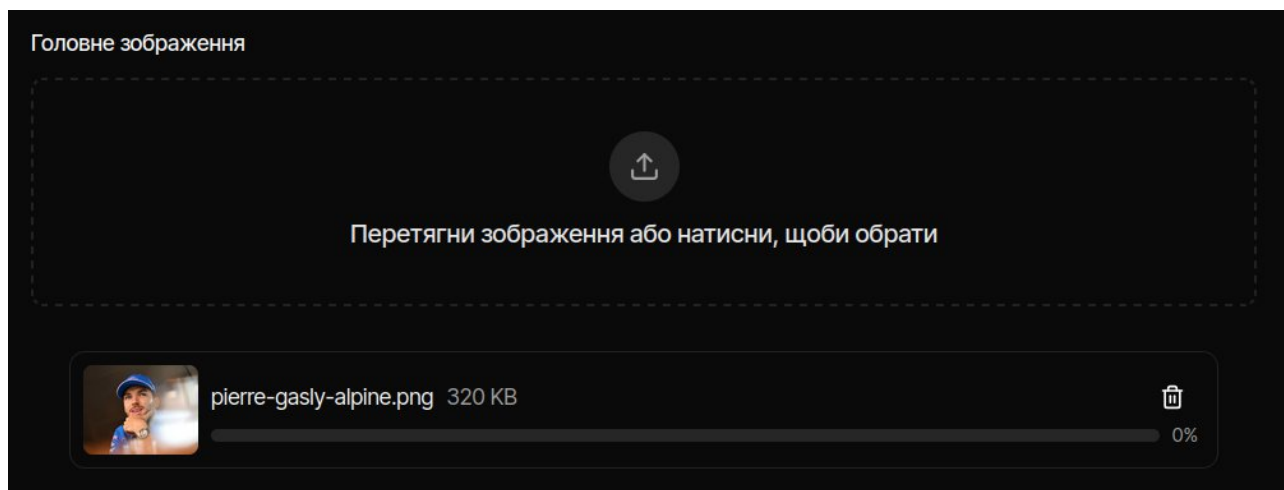


Рис. 3.26. Обрання фото до статті

Невід’ємною частиною статей є абзаци тексту, які формують її вміст. Для цього ми можемо створити новий абзац, натиснувши на кнопку “Додати абзац”. Далі заповнюємо інформацію до нього та одразу спостерігаємо за змінами (рис. 3.27).

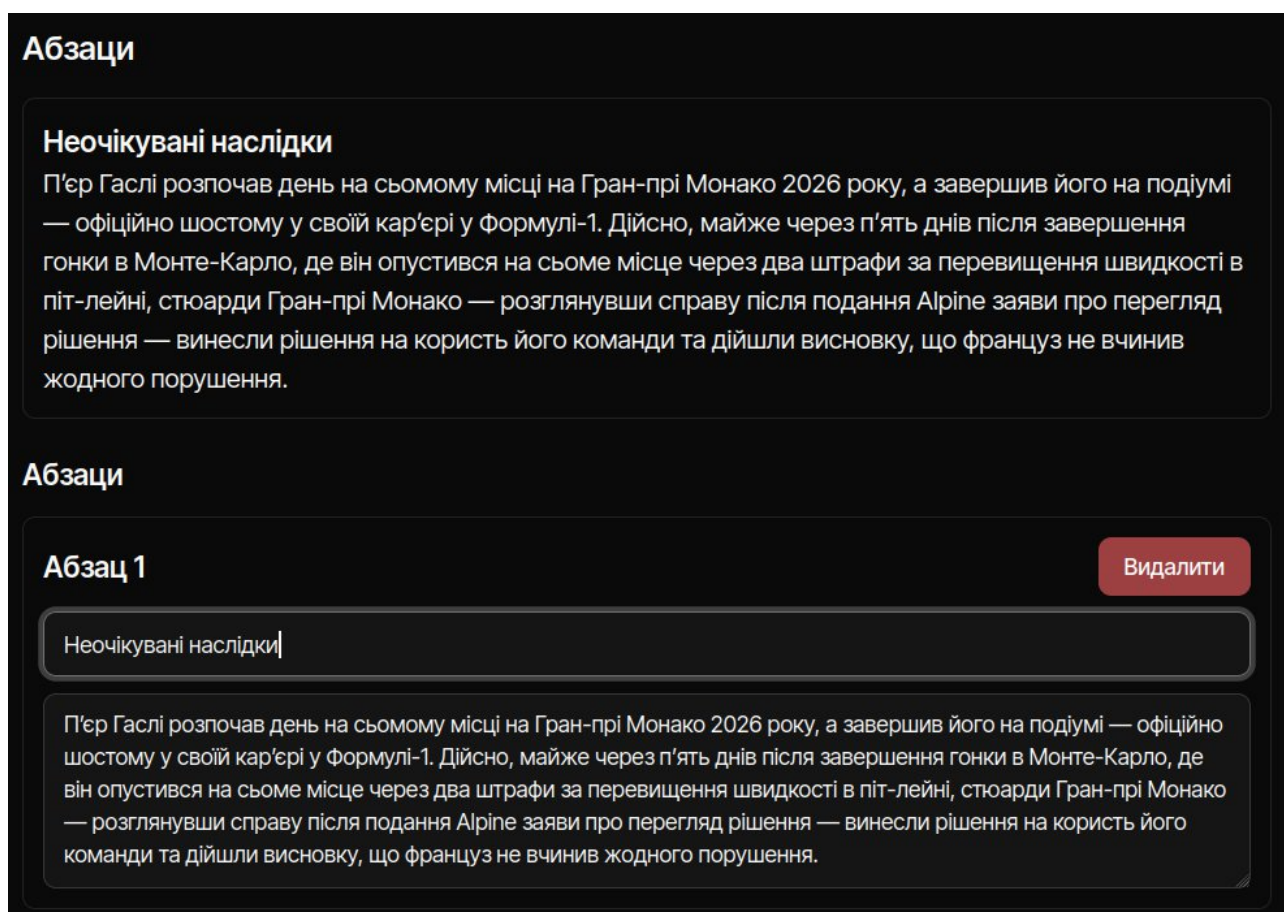


Рис. 3.27. Додання нового абзацу

Наступним етапом є збереження чернетки. Дана дія виконується при натисканні кнопки “Зберегти чернетку”, після чого зберігаються всі дані до

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				56

сховища файлів на Firebase та основної БД. При успішному збереженні з'являється сповіщення з текстом “Чернетка оновлена” (рис. 3.28).

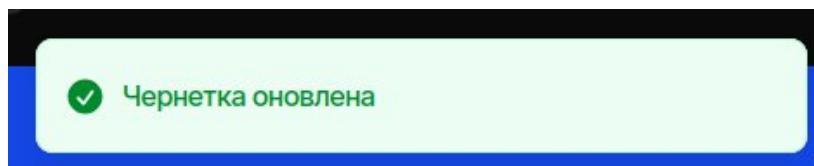


Рис. 3.28. Сповіщення про оновлення чернетки

У результаті збереження записки із завантаженим зображенням його можна бачити коло поля завантаження фото (рис. 3.29). Це забезпечує візуальне підтвердження успішного завантаження файлу.

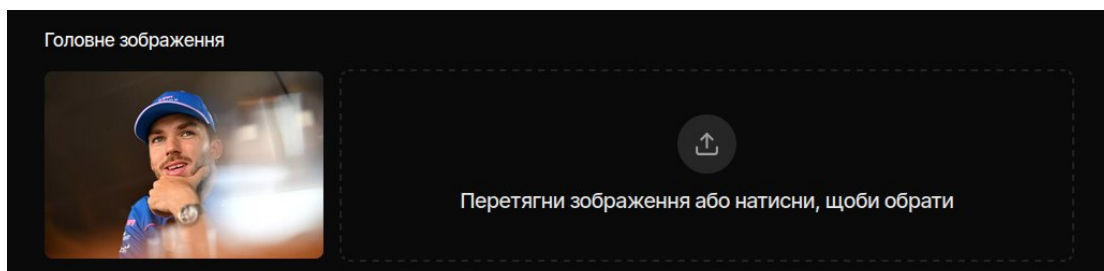


Рис. 3.29. Завантажене фото

Останнім етапом створення статті є її публікація. Натискання на кнопку “Опублікувати” завершить її написання та оновить її статус на PUBLISHED. Далі з'явиться сповіщення про успішну публікацію статті та відбудеться навігація на її сторінку (рис. 3.30).

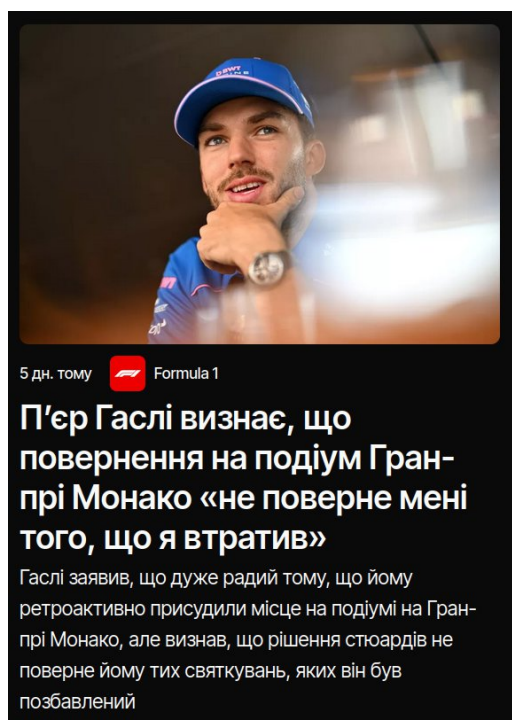


Рис. 3.30. Наявність опублікованої статті

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				57

Наступною важливою частиною portalу є обговорення. Це місце (рис. 3.31), де користувачі можуть створювати власні дискусії на будь-які теми за наявними категоріями. Вони працюють за схожим принципом зі статтями, а саме обговореннями в коментарях.

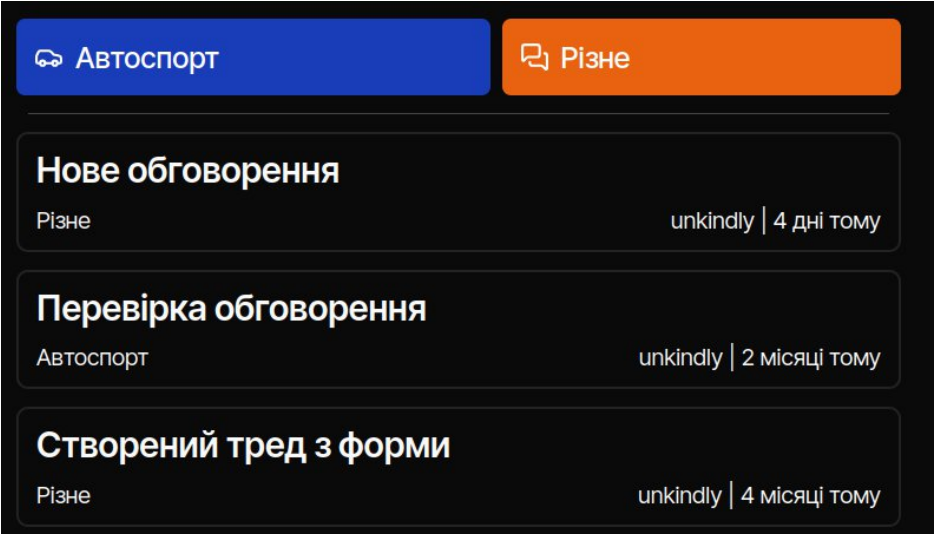


Рис. 3.31. Сторінка обговорень

Кожен тред за структурою не сильно відрізняється від статті (рис. 3.32). Обое сутностей мають свій контент та коментарі, але головна перевага обговорень у тому, що для їх створення достатньо прав звичайного користувача.

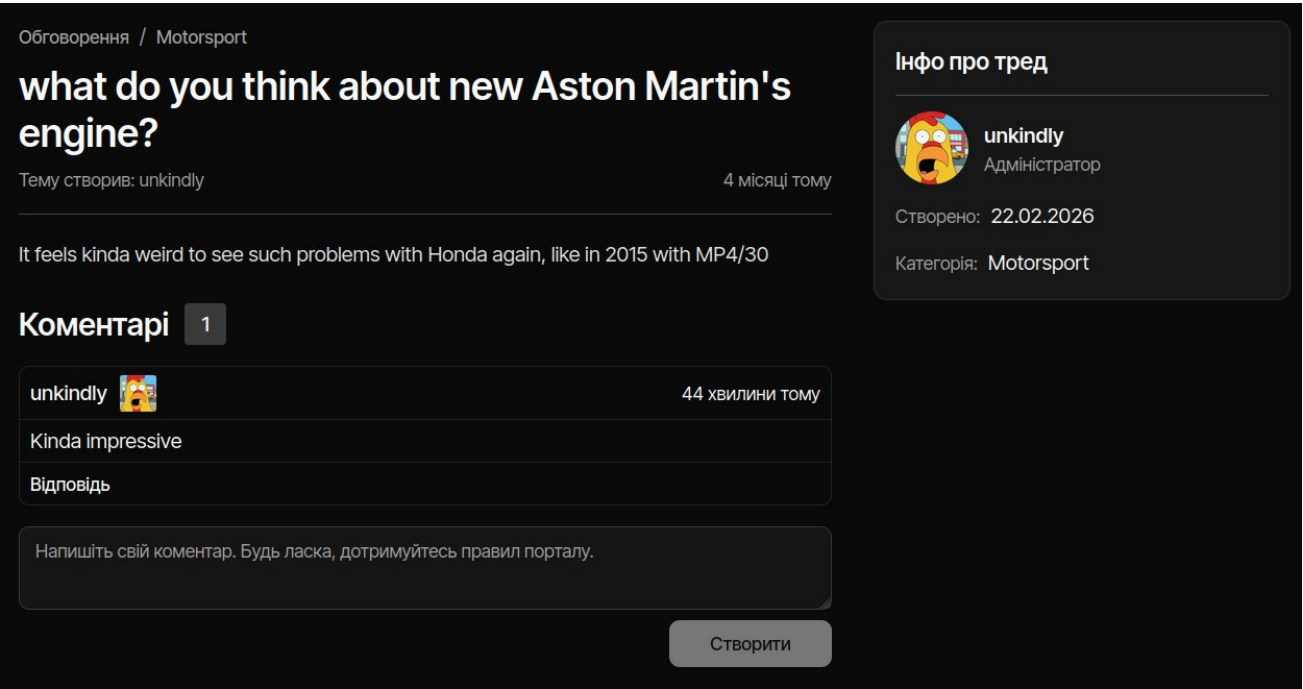


Рис. 3.32. Вигляд обговорення

Щоби створити нове обговорення, потрібно натиснути на кнопку “Створити обговорення на основній сторінці” (користувач має бути

авторизованим, в інакшому випадку кнопка буде вимкненою, а при наведенні на неї показуватиметься підказка з проханням увійти в портал). Після цього відкриється форма створення обговорення (рис. 3.33).

Рис. 3.33. Форма створення обговорення

Після успішного створення обговорення об’єкт з’являється на основній сторінці обговорень, а також в боковому компоненті останніх обговорень (рис. 3.34). Також паралельно відображається сповіщення про успішне створення обговорення.

Рис. 3.34. Успішне створення треду

Іншою важливою частиною порталу є водії. На порталі REVITUP можна знайти водіїв, які вас цікавлять. На основній сторінці (рис. 3.35) наявні пошук та фільтрація за дисциплінами, в яких беруть участь пілоти.

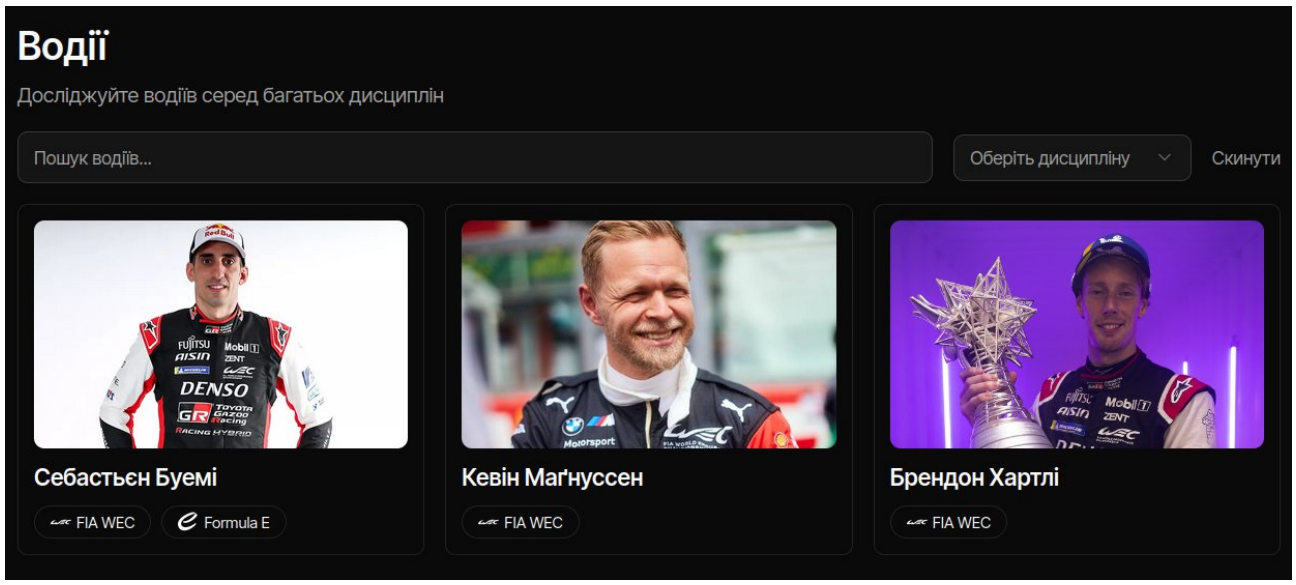


Рис. 3.35. Основна сторінка водіїв

При переході на сторінку водія ми можемо бачити основну інформацію про нього (рис. 3.36), а саме ім'я, дату та країну народження, основний номер пілота, а також біографію та основне фото пілота. Додатково присутня галерея фото, на яких присутній даний пілот.

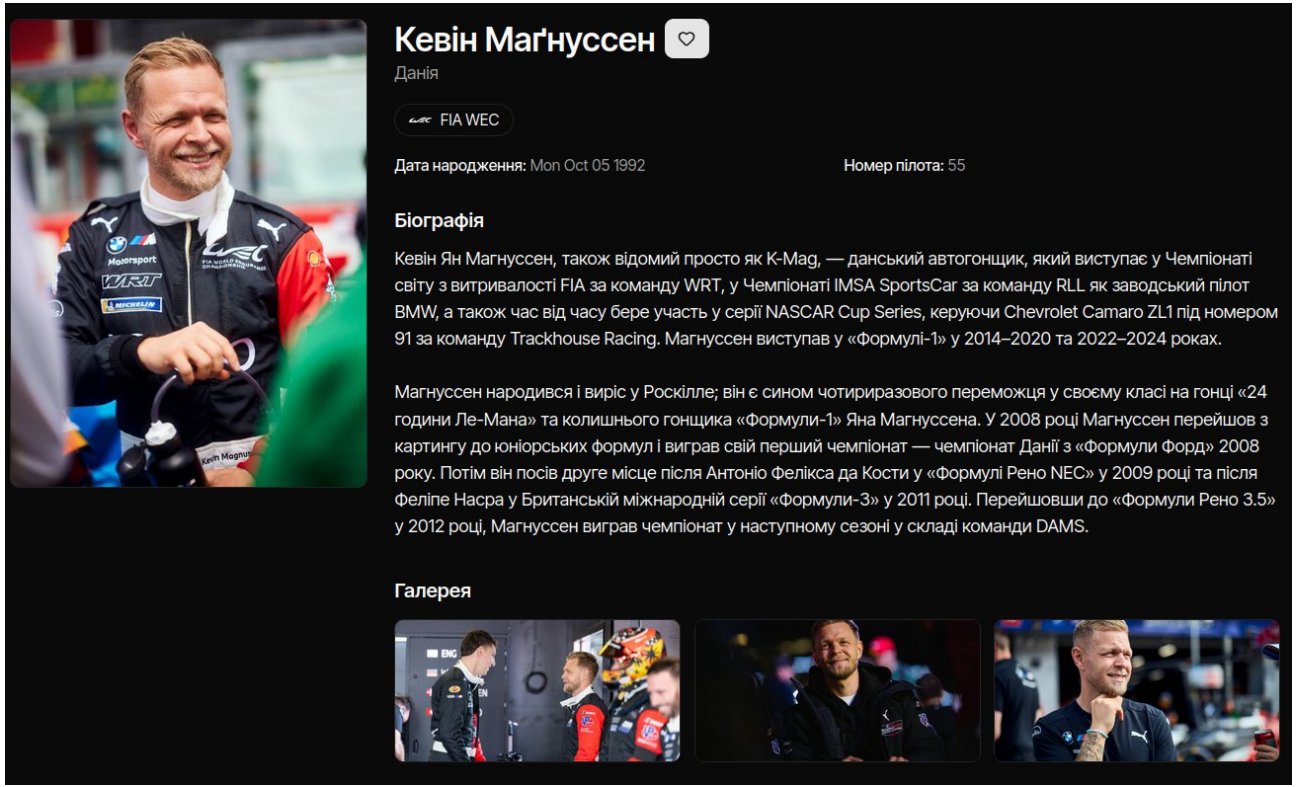


Рис. 3.36. Сторінка водія

Також можна обрати водія улюбленим у профілі, натиснувши на кнопку навпроти імені пілота. Після цього в коментарях та профілі користувача буде відображатися (рис. 3.37) його прізвище та номер.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				60
		№ докум.				

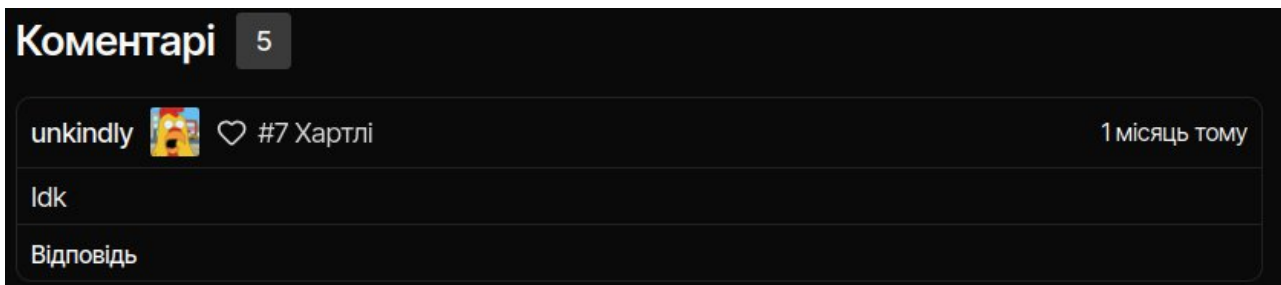


Рис. 3.37. Відображення улюбленого пілота користувача в його коментарях

Користувачам важливо переглядати статистичну інформацію про гонки, сезони, водіїв тощо. На порталі реалізована секція під назвою “Статистика”, яка дозволяє користувачам знайти потрібні для них дані. Наразі реалізовано лише перегляд сезонів дисциплін та результатів гонок, які тривали в цих сезонах, але в майбутньому можливе розширення функціоналу.

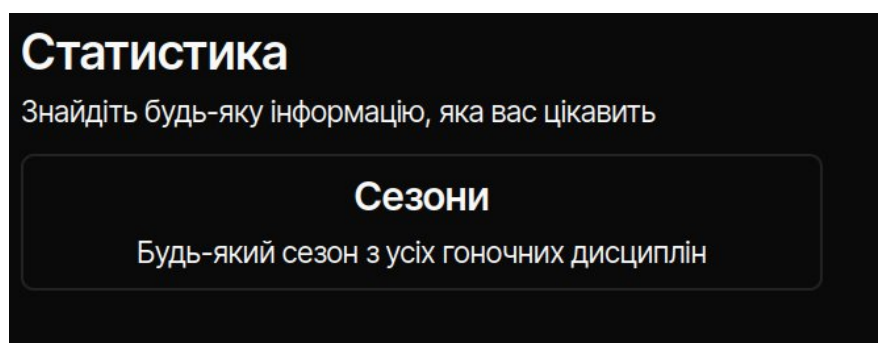


Рис. 3.38. Сторінка статистики

При потраплянні на сторінку сезонів ми можемо обрати наявні сезони серед автоспортивних дисциплін (рис. 3.39). Також можливе фільтрування сезонів за дисципліною та роком.

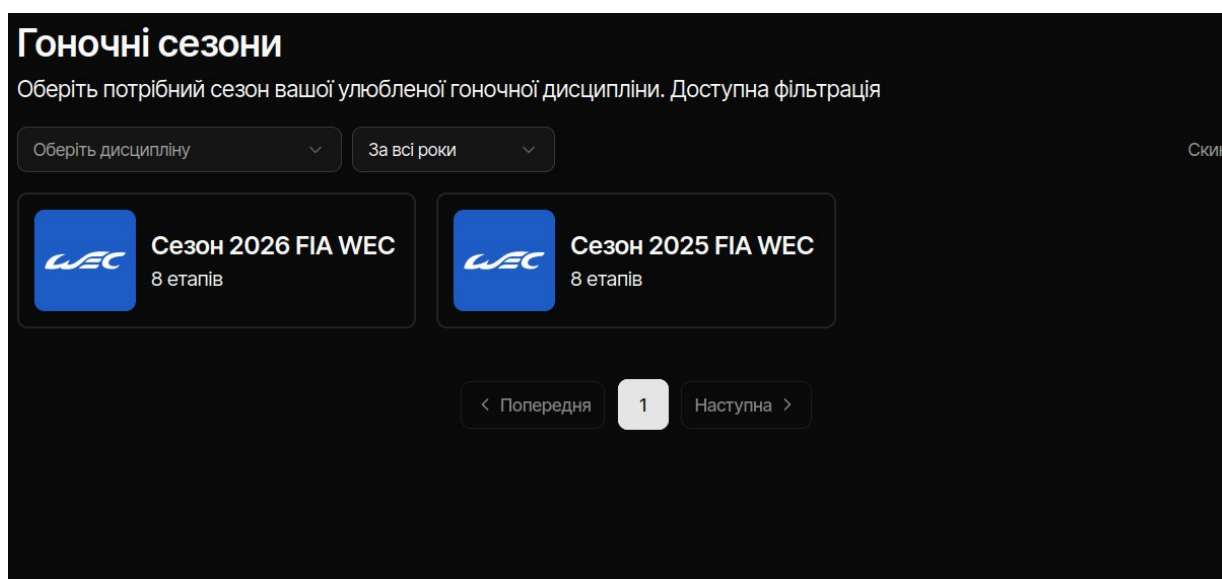


Рис. 3.39. Перегляд наявних сезонів

При відкритті будь-якого сезону ми можемо бачити події (рис. 3.40), які відбувалися або відбудуться у ньому. На сторінці відображується основна інформацію про кожну гонку сезону.

 Сезон 2026 FIA WEC 8 етапів 		
1. 2026 6 годин Імоли  Імола 213 кіл (1045.617 km)  Fri Apr 17 2026 - Sun Apr 19 2026	2. 2026 6 годин Спа-Франкоршамс  Спа-Франкоршам 152 кола (1064.608 km)  Thu May 07 2026 - Sat May 09 2026	3. 2026 24 години Ле-Ману  Ла Сарт 387 кіл (5273.262 km)  Wed Jun 10 2026 - Sun Jun 14 2026
4. 2026 6 годин Сан-Пауло  Інтерлагос 242 кола (1042.778 km)  Fri Jul 10 2026 - Sun Jul 12 2026	5. 2026 Lone Star Ле-Ман  Кільце Америк 120 кіл (661.560 km)  Fri Sep 04 2026 - Sun Sep 06 2026	6. 2026 6 годин Фуджі  Траса Фуджі 202 кола (921.726 km)  Fri Sep 25 2026 - Sun Sep 27 2026
7. 2026 1812 КМ Катару  Міжнародний автодром «Лусаїл» 335 кіл (1815.365 km)  Thu Oct 22 2026 - Sat Oct 24 2026	8. 2026 8 годин Бахрейну  Міжнародний автодром Бахрейну 237 кіл (1282.644 km)  Wed Nov 04 2026 - Fri Nov 06 2026	

Рис. 3.40. Перегляд гонок у сезоні

При відкритті події ми можемо бачити її вміст (рис. 3.41) у вигляді опису гонки, інформації про дисципліну, трасу, на якій відбувалась гонка, дати проведення гонки, а також кола та довжина події. Також наявна інформація про її класифікацію.

1. 2026 6 годин Імоли						
«6 годин Імоли» 2026 року — це гонка на витривалість серед спортивних автомобілів, яка відбулася 17–19 квітня 2026 року і стала першим із восьми етапів Чемпіонату світу з гонок на витривалість FIA 2026 року. Це був четвертий за рахунком заїзд, а також третій поспіль, що проводився в рамках Чемпіонату світу з гонок на витривалість.						
 Імола 213 кіл (1045.617 km)  Fri Apr 17 2026 - Sun Apr 19 2026						
Сезон :  2026 FIA WEC						
Класифікація події						
Поз ↑	#	Учасник	Болід	Час	Найшвидше коло	Водій
1	7	#7 Toyota Racing	Toyota GR010 Hybrid	1:25.059		Брендон Хартлі
2	8	#8 Toyota Racing	Toyota GR010 Hybrid	1:27.559	-	Себастьян Бюемі
3	15	#15 BMW M Team WRT	BMW M Hybrid V8	1:29.559	-	Кевін Магнуссен

Рис. 3.41. Сторінка гонки

		Близнюк Т.Р.		
		Савіцький Р.С		
	.	№ докум.		

ІПЗ.КР.Б-121-26-ПЗ

Арк.

62

Таблиця класифікації гонки підтримує сортування даних за колонками (рис. 3.42). Дана особливість покращує та полегшує аналіз даних гонки.

Класифікація події						
Поз	#	Учасник	Болід	Час ↓	Найшвидше коло	Водій
3	15	#15 BMW M Team WRT	BMW M Hybrid V8	1:29.559	-	Кевін Магнуссен
2	8	#8 Toyota Racing	Toyota GR010 Hybrid	1:27.559	-	Себастьян Буемі
1	7	#7 Toyota Racing	Toyota GR010 Hybrid	1:25.059		Брендон Хартлі

Рис. 3.42. Сортування таблиці результатів

Таким чином було спроектовано та реалізовано функціонал порталу REVITUP. Створений функціонал забезпечує читання та створення статей, обговорення, пошук водіїв та перегляд наявної статистики.

3.3. Тестування системи

Кожна система потребує належного тестування та перевірки на баги чи неочікувану поведінку. Такий підхід гарантує надійність та довготривалість життєвого циклу системи. В даному випадку потрібно переконатися у тому, чи працює функціонал, вказаний у розробці, без відхилень та помилок.

Тестування відбувалося як на бекенді, так і на фронтенді порталу. На бекенді одним з головних факторів є валідація даних, які приходять з клієнтської частини порталу. Різноманітні фактори, такі як відсутність даних у полі або неправильний тип даних, можуть порушити роботу порталу для користувача або спричинити порушення у відображенні наявних даних. Для цього використовується валідація даних, які надходять від користувачів, а саме тіл запиту, query-рядків та параметрів URL-посилань. При помилці валідації даних на бекенді користувач отримує BadRequestException зі статусом 400 та масивом messages з поясненнями помилок. На рис. 3.43 зображено приклад помилки валідації даних на POST-ендпоінті для створення коментарів, де вказано неіснуюче значення entitySource.

Окрім перевірки коректності вхідних даних, було проведено тестування обробки винятків. Зокрема, було протестовано сценарії доступу до неіснуючих ресурсів, спроб виконання операцій над видаленими записами та передачі некоректних ідентифікаторів у параметрах маршруту. У таких випадках система повинна коректно повертати відповідні коди помилок HTTP та зрозумілі повідомлення, що спрощує пошук та виправлення помилок як для користувачів, так і для розробників.

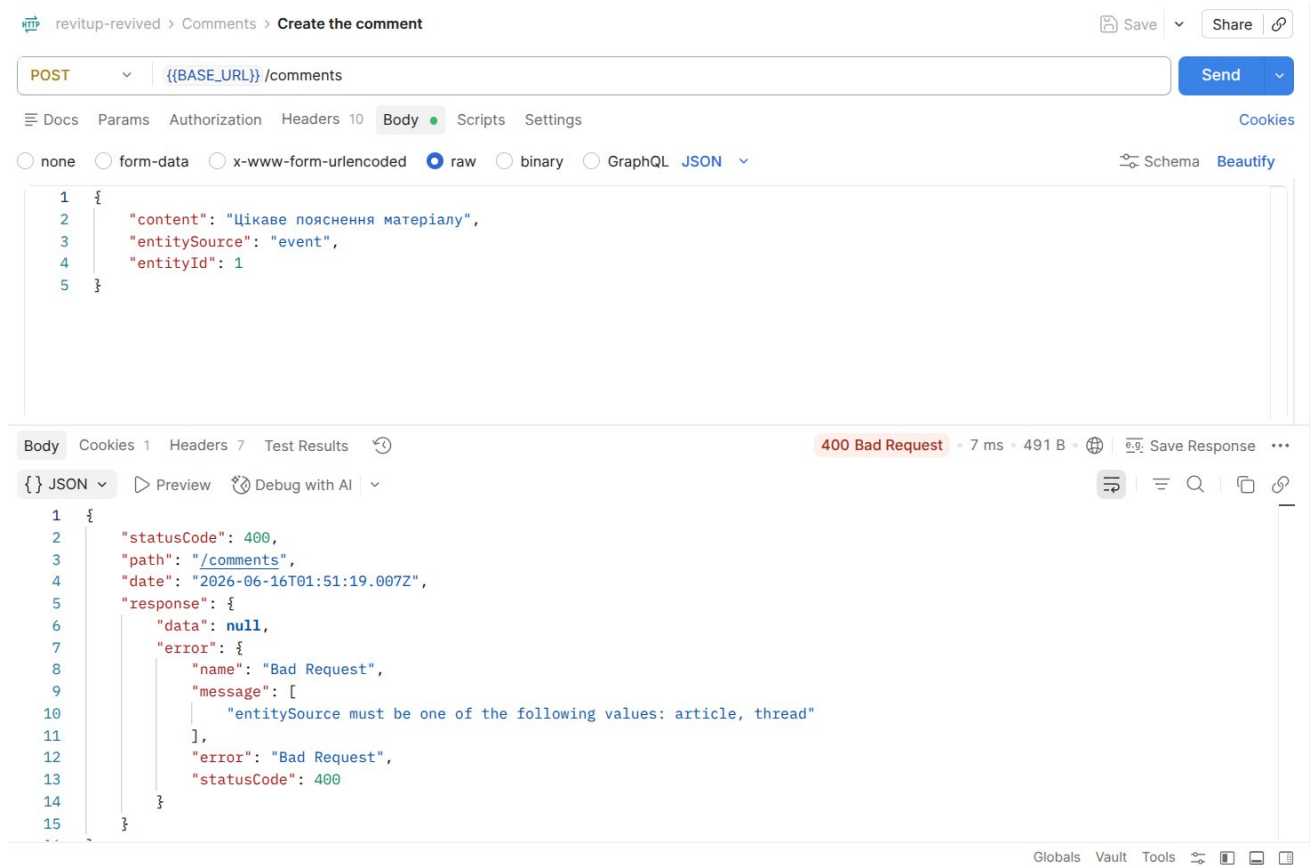


Рис. 3.43. Приклад помилки валідації даних

Іншим важливим фактором при тестуванні бекенду є перевірка авторизації та аутентифікації порталу. До певних ендпоінтів користувачі можуть отримати доступ лише якщо вони авторизовані, тобто мають валідний (справжній) JWT-токен авторизації, в якого не збіг час життя. В іншому випадку користувач отримує `UnauthorizedException` зі статусом 401. Але навіть якщо користувач має справжній токен, не кожен ендпоінт може бути доступним із правами даного користувача. В такому випадку отримується інформація про користувача, а саме його ролі, та звіряється з тими ролями, які вказані на доступ на даному ендпоінті. Якщо ролі не сходяться, користувач отримує `ForbiddenException` зі статусом 403.

	Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
	Савіцький Р.С				64
.	№ докум.				

Додатково, коректність роботи API перевірялася під час виконання типових сценаріїв використання системи. Було протестовано створення, редагування та отримання даних для основних сутностей порталу, включаючи статті, коментарі, обговорення та профілі користувачів. Результати тестування підтвердили коректну взаємодію між контролерами, сервісами та репозиторіями, а також коректну обробку відповідей сервером.

На фронтенді тестування даних має в собі декілька важливих нюансів, такі як перевірка полів на відповідність даних, відображення контенту у відповідності до стану авторизації та доступів користувача та адаптивність інтерфейсу.

Кожна форма на клієнтській частині порталу перевіряється на наявність помилок чи невідповідностей до її схеми. При неправильних даних у полі під ним показується помилка, а кнопка надсилання даних є неактивною, допоки дані у формі будуть заповнені правильно.

Наступним важливим аспектом є відображення даних у відповідності до стану авторизації та доступів користувача. Звичайний користувач не може виконувати вищі дії, такі як створення чернеток статей. Щоби забезпечити валідність доступів, клієнтська частина перевіряє, чи є справжній JWT-токен у користувача. В успішному випадку клієнтська частина отримує інформацію про користувача та зберігає її у пам'яті. Якщо даних користувача немає, то при спробі доступу до контенту, для якого потрібен авторизований доступ, користувач автоматично переходить на головну сторінку зі сповіщенням про потребу у вході/реєстрації на порталі. Якщо користувач не має відповідних ролей для доступу до контенту, який потребує їх, то даний контент не відображається користувачу.

Останнім аспектом варто назвати адаптивність UI. Подібні портали часто використовуються на мобільних пристроях через їхню зручність у використанні та деталізований інтерфейс. Такий підхід надасть користувачам зручність у використанні та підтримку різноманітних макетів. Для перевірки на адаптивність

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
		№ докум.				65

можна просто зайти в режим розробника та змінити ширину видимої області UI. На рис. 3.44 можна бачити, як інтерфейс підлаштувався під мобільну версію.

Під час тестування також оцінювалася загальна стабільність роботи клієнтської частини при навігації між сторінками та виконанні послідовних запитів до сервера. Особлива увага приділялася коректному відображенню станів завантаження, повідомлень про помилки та оновленню даних після виконання операцій створення або редагування контенту. Це дозволило забезпечити цілісний користувацький досвід та мінімізувати можливість виникнення візуальних або функціональних помилок.

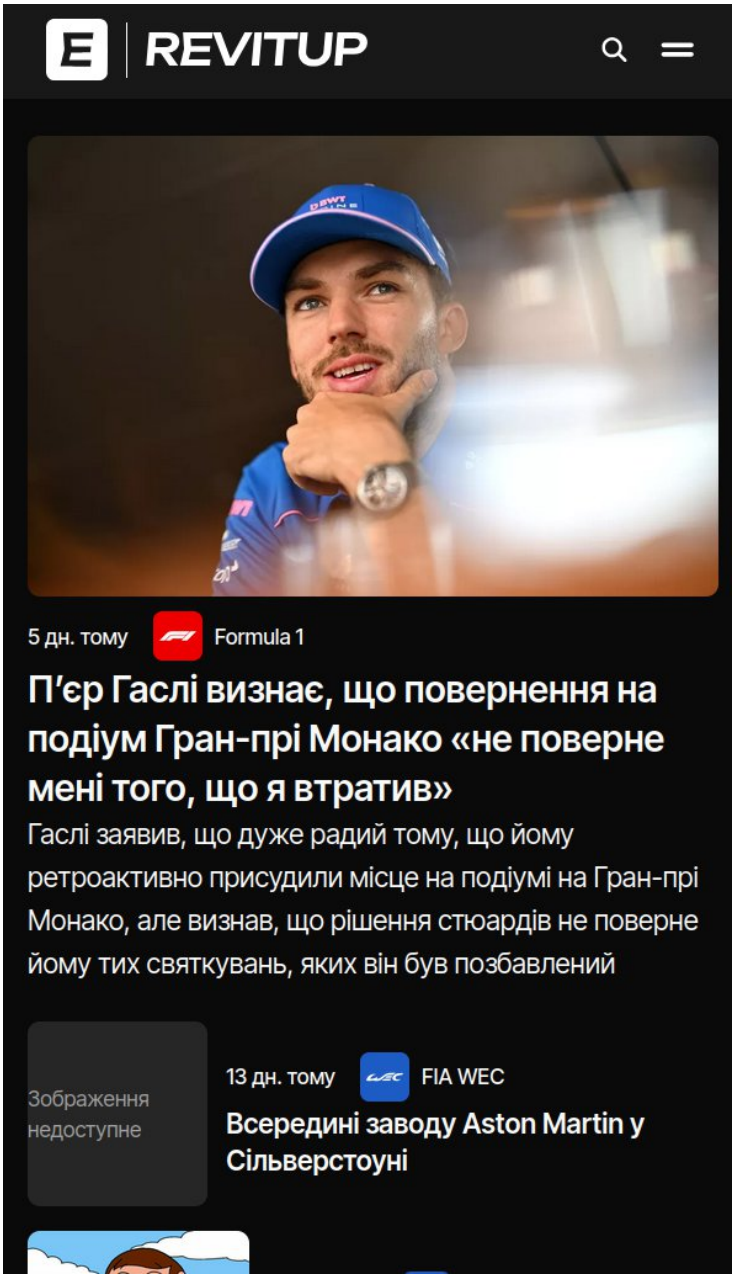


Рис. 3.44. Адаптивність UI

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				66
		№ докум.				

Висновки до третього розділу

У третьому розділі було детально описано технічну частину та функціонал порталу REVITUP, зокрема реалізацію основних модулів системи та їхню взаємодію, а також проведено тестування основних аспектів системи.

Спочатку було описано компоненти, які використовувалися для побудови системи порталу та її вимоги для використання. Було обрано основні технології для клієнтської та серверної частин системи, а також основні БД для зберігання та кешування даних і сервіси для збереження файлів, в тому числі й зображень.

Далі було покроково описано та показано функціонал порталу на прикладі статей, обговорень, водіїв та статистики. Продемонстровано роботу із чернетками статей, публікацією коментарів на статтях та обговореннях, пошуком об'єктів, а також показано локалізацію інтерфейсу порталу та статистичні дані результатів перегонів.

Тестування платформи було виконано як на клієнтській, так і на серверній частинах порталу. На бекенді було проведено перевірку валідації даних, поведінку ендпоінтів API при різних станах авторизації та аутентифікації та стабільність критичного функціоналу. На фронтенді було проведено перевірку на валідацію даних на UI, відображення даних при різних станах користувача, а також адаптивність верстки.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				67

ВИСНОВКИ

У рамках кваліфікаційної бакалаврської роботи було досліджено тематику спортивних інформаційних систем та розроблено автоспортивний інформаційно-статистичний портал REVITUP. Головною метою проєкту було створення сучасної вебплатформи, яка поєднує інформаційний контент, статистичні дані та можливості взаємодії з користувачем у рамках єдиного програмного рішення. Особливу увагу було приділено темі автоспорту, яка вимагає своєчасного відображення результатів змагань, статистики сезону, а також інформації про водіїв, дисципліни та спортивні події.

У першому розділі було проаналізовано існуючі спортивні портали та інформаційні ресурси. На основі порівняння їхніх функціональних можливостей, переваг та недоліків було сформульовано функціональні та нефункціональні вимоги до системи REVITUP. Також було обґрунтовано вибір клієнт-серверної архітектури, з урахуванням вимог до масштабованості, продуктивності та безпеки даних.

У другому розділі описано архітектуру системи. Було визначено основних користувачів порталу та їхні права доступу, а також розроблено структуру бази даних для зберігання інформації про користувачів, статті, коментарі, дисципліни, сезони, етапи змагань та результати перегонів. Для опису функціонування системи було створено діаграми випадків використання та об'єктно-орієнтовану доменну модель.

У рамках цього проєкту було реалізовано основну функціональність порталу. Система підтримує створення та публікацію статей, роботу з чернетками, коментування контенту та тематичні обговорення. Також було створено модуль статистики для перегляду сезонів, етапів чемпіонату та результатів гонок. Було реалізовано окремий модуль, присвячений гонщикам, що містить біографічні відомості, фотографії та можливість вибрати улюбленого спортсмена.

Значна увага приділялася питанням безпеки. Були впроваджені механізми автентифікації та авторизації користувачів з керуванням доступом на основі

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.	I			68

ролей, а також перевірка та валідація даних як на стороні клієнта, так і на стороні сервера. Це забезпечує надійність системи та захищає її від некоректних запитів.

На завершальному етапі було протестовано клієнтську та серверну компоненти portalу. Було перевірено коректність функціонування бізнес-логіки, стабільність API та взаємодію між компонентами системи. Результати тестування підтвердили функціональність програмного забезпечення та його відповідність заданим вимогам.

Отримані результати свідчать про досягнення поставленої мети та виконання основних завдань кваліфікаційної роботи. Розроблений портал REVITUP є повноцінною інформаційно-статистичною платформою для автоспортивної тематики та може слугувати основою для подальшого розвитку й розширення функціональних можливостей.

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				69
	.	№ докум.				

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Axios [Електронний ресурс]. URL: <https://axios.rest> (дата звернення: 17.06.2026).
2. Banks A., Porcello E. Learning React. O'Reilly Media, 2020. - 350 p.
3. Bass L., Clements P., Kazman R. Software Architecture in Practice. Addison-Wesley, 2021. - 560 p.
4. Casciaro M., Mammino L. Node.js Design Patterns. Packt Publishing, 2020. - 526 p.
5. Class-validator [Електронний ресурс]. URL: <https://github.com/typestack/class-validator> (дата звернення: 17.06.2026).
6. Evans E. Domain-Driven Design. Addison-Wesley, 2003. - 560 p.
7. Firebase [Електронний ресурс]. - URL: <https://firebase.google.com/docs> (дата звернення: 17.06.2026).
8. Firebase Storage [Електронний ресурс]. URL: <https://firebase.google.com/products/storage> (дата звернення: 17.06.2026).
9. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002. - 533 p.
10. Grigorik I. High Performance Browser Networking. O'Reilly Media, 2013. - 400 p.
11. JWT Introduction [Електронний ресурс]. URL: <https://jwt.io/introduction> (дата звернення: 17.06.2026).
12. Kleppmann M. Designing Data-Intensive Applications. O'Reilly Media, 2017. - 616 p.
13. Coronel C., Morris S. Database Systems: Design, Implementation, and Management. Cengage Learning, 2023. - 832 p.
14. Ky HTTP Client [Електронний ресурс]. URL: <https://github.com/sindresorhus/ky> (дата звернення: 17.06.2026).
15. Martin R. C. Clean Architecture. Prentice Hall, 2017. - 432 p.
16. NestJS [Електронний ресурс]. URL: <https://docs.nestjs.com/> (дата звернення: 17.06.2026).

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.		І		70

17. Newman S. Building Microservices. O'Reilly Media, 2015. - 280 p.
18. Newman S. Monolith to Microservices. O'Reilly Media, 2019. - 350 p.
19. Node.js Documentation [Електронний ресурс]. URL: <https://nodejs.org/en/docs> (дата звернення: 17.06.2026).
20. PostgreSQL [Електронний ресурс]. URL: <https://www.postgresql.org/docs/> (дата звернення: 17.06.2026).
21. React [Електронний ресурс]. URL: <https://react.dev/> (дата звернення: 17.06.2026).
22. Redis [Електронний ресурс]. URL: <https://redis.io/docs> (дата звернення: 17.06.2026).
23. Richardson C. Microservices Patterns. Manning, 2018. - 520 p.
24. Tailwind CSS [Електронний ресурс]. URL: <https://tailwindcss.com> (дата звернення: 17.06.2026).
25. TanStack Query [Електронний ресурс]. URL: <https://tanstack.com/query/latest> (дата звернення: 17.06.2026).
26. TypeScript [Електронний ресурс]. URL: <https://www.typescriptlang.org/docs/> (дата звернення: 17.06.2026).
27. Using OAuth 2.0 to Access Google APIs [Електронний ресурс]. URL: <https://developers.google.com/identity/protocols/oauth2> (дата звернення: 18.06.2026).
28. Valkey [Електронний ресурс]. URL: <https://valkey.io/> (дата звернення: 18.06.2026).
29. Vite [Електронний ресурс]. URL: <https://vite.dev/> (дата звернення: 17.06.2026).
30. Wathan A., Schoger S. Refactoring UI. Refactoring LLC, 2018. - 200 p.
31. Близнюк Т.Р., Савіцький Р.С. Застосування RBAC-підходу при використанні json web token. Тези XIX Міжнародної науково-практичної конференції «Теорія і практика сучасної науки та освіти», 19-20 червня 2026 року. Львів: «Львівський науковий форум», 2026. [Подано до друку]

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С.				
	.	№ докум.				71

32. Близнюк Т.Р., Савіцький Р.С., Толстой І.А. Виявлення вразливостей у системах авторизації на базі json web token. Тези XIX Міжнародної науково-практичної конференції «Теорія і практика сучасної науки та освіти», 19-20 червня 2026 року. Львів: «Львівський науковий форум», 2026. [Подано до друку]

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				72

ДОДАТКИ

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.	I			73

Лістинг класу ArticleModule:

```
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { Article } from './article.entity';
import { ArticleController } from './article.controller';
import { ArticleService } from './article.service';
import { ArticleRepository } from './article.repository';
import { ParagraphModule } from './paragraphs/paragraph.module';
import { FirebaseModule } from '../common/firebase/firebase.module';
import { CommentModule } from '../comments/comment.module';
```

```
@Module({
  imports: [
    TypeOrmModule.forFeature([Article]),
    ParagraphModule,
    FirebaseModule,
    CommentModule,
  ],
  controllers: [ArticleController],
  providers: [ArticleService, ArticleRepository],
})
export class ArticleModule {}
```

Лістинг класу ArticleEntity:

```
import {
  Column,
  CreateDateColumn,
  DeleteDateColumn,
  Entity,
  JoinColumn,
  ManyToOne,
  OneToMany,
  PrimaryGeneratedColumn,
  UpdateDateColumn,
} from 'typeorm';

import { ArticleStatus } from 'features/articles/enums/article-status.enum';
import { ParagraphEntity } from 'features/paragraphs/paragraph.entity';
import { UserEntity } from '../user/user.entity';
import { DisciplineEntity } from '../disciplines/discipline.entity';

@Entity('articles')
export class Article {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({
    length: 125,
```

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				74

```

    })
    title: string;

    @Column({
      length: 200,
      name: 'preview_text',
      nullable: true,
    })
    previewText: string;

    @Column({ name: 'main_img_url', nullable: true })
    mainImageUrl: string;

    @CreateDateColumn()
    createdAt: Date;

    @UpdateDateColumn({ nullable: true })
    updatedAt: Date;

    @DeleteDateColumn({ name: 'deleted_at', nullable: true })
    deletedAt: Date | null;

    @OneToMany(() => ParagraphEntity, (p) => p.article)
    paragraphs: ParagraphEntity;

    @Column({
      type: 'enum',
      enum: ArticleStatus,
      default: ArticleStatus.DRAFT,
    })
    status: ArticleStatus;

    @ManyToOne(() => UserEntity)
    @JoinColumn({ name: 'author_id' })
    author: UserEntity;

    @ManyToOne(() => DisciplineEntity, { nullable: true })
    @JoinColumn({ name: 'discipline_id' })
    discipline: DisciplineEntity;
  }

```

Лістинг класу ArticleController:

```

/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */
import {
  Body,
  Controller,

```

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				75

```

Delete,
Get,
Param,
ParseIntPipe,
Patch,
Post,
Query,
UploadedFile,
UseGuards,
UseInterceptors,
} from '@nestjs/common';

import { ArticleService } from 'features/articles/article.service';
import { ArticleCreateDto } from './dto/article-create.dto';
import { ArticleEditDto } from './dto/article-edit.dto';
import { CurrentUser } from '../auth/decorators/user.decorator';
import { AccessTokenGuard } from '../auth/guards/access-token.guard';
import { UserPayloadDto } from '../auth/dto';
import { FileInterceptor } from '@nestjs/platform-express';
import { imageFileFilter } from '../../common/file-upload/image-file.filter';
import { Roles } from '../auth/decorators/roles.decorator';
import { UserRole } from '../user/enums/user-role.enum';
import { ArticleQueryDto } from './dto/article-query.dto';
import { CacheTTL } from '@nestjs/cache-manager';

@Controller('articles')
export class ArticleController {
  constructor(private service: ArticleService) {}

  @Get()
  @CacheTTL(10000)
  async getArticles(@Query() query: ArticleQueryDto) {
    return this.service.findArticles(
      query.page ?? 1,
      query.take ?? 10,
      query.search,
    );
  }

  @Get('random')
  @CacheTTL(10000)
  async getRandomArticle() {
    return this.service.getRandomArticle();
  }

  @Get('/:id')
  @CacheTTL(10000)
  async findArticleById(@Param('id', ParseIntPipe) id: number) {
    return await this.service.findArticleById(id);
  }
}

```

```

@Get('drafts/me')
@UseGuards(AccessTokenGuard)
async findMyDrafts(@CurrentUser() user: UserPayloadDto) {
  return this.service.findMyDrafts(user.id);
}

```

```

@Get('drafts/:id')
@UseGuards(AccessTokenGuard)
async getDraftById(
  @Param('id', ParseIntPipe) id: number,
  @CurrentUser() user: UserPayloadDto,
) {
  return this.service.getDraftById(id, user.id);
}

```

```

@Post()
@Roles([UserRole.EDITOR, UserRole.ADMIN])
@UseGuards(AccessTokenGuard)
async createArticle(
  @Body() article: ArticleCreateDto,
  @CurrentUser() user: UserPayloadDto,
) {
  await this.service.createArticle(article, user.id);

  return { message: 'Article was created successfully' };
}

```

```

@Patch('publish/:id')
@Roles([UserRole.EDITOR, UserRole.ADMIN])
@UseGuards(AccessTokenGuard)
async publishArticle(
  @Param('id', ParseIntPipe) id: number,
  @CurrentUser() user: UserPayloadDto,
) {
  await this.service.publishArticle(id, user.id);

  return { message: 'Article published successfully' };
}

```

```

@Patch('update/:id')
@Roles([UserRole.EDITOR, UserRole.ADMIN])
@UseGuards(AccessTokenGuard)
@UseInterceptors(
  FileInterceptor('image', {
    fileFilter: imageFileFilter,
  }),
)
async updateArticle(
  @Param('id', ParseIntPipe) id: number,

```

		Близнюк Т.Р.		
		Савицький Р.С		
	.	№ докум.		

```

@Body() partialArticle: any,
@UploadedFile() file?: Express.Multer.File,
){
  const body = {
    ...partialArticle,
    paragraphs: partialArticle.paragraphs
      ? JSON.parse(partialArticle.paragraphs)
      : undefined,
  } as ArticleEditDto;

  await this.service.updateArticle(id, body, file);

  return { message: 'Article was updated successfully' };
}

@Patch('revert-soft-delete/:id')
@Roles([UserRole.EDITOR, UserRole.ADMIN])
async revertSoftDelete(@Param('id', ParseIntPipe) id: number) {
  await this.service.revertSoftDelete(id);

  return { message: "Article's deletion is reverted" };
}

@Delete('soft/:id')
@Roles([UserRole.EDITOR, UserRole.ADMIN])
async softDelete(@Param('id', ParseIntPipe) id: number) {
  await this.service.softDeleteArticle(id);

  return { message: 'Article was soft-deleted successfully' };
}
}

```

Лістинг класу ArticleService:

```

import { Injectable, NotFoundException } from '@nestjs/common';
import { ArticleRepository } from '../article.repository';
import { Article } from '../article.entity';
import { plainToInstance } from 'class-transformer';
import { ArticleShortDto } from '../dto/article-short.dto';
import { ArticleCreateDto } from '../dto/article-create.dto';
import { ArticleEditDto } from '../dto/article-edit.dto';
import { NotNull } from '../../common/constants/database.constants';
import { FirebaseStorageService } from '../../common/firebase/firebase-storage.service';
import { ArticleStatus } from '../enums/article-status.enum';
import { CommentService } from '../comments/comment.service';
import { CommentSource } from '../comments/types/comment-source.enum';

@Injectable()
export class ArticleService {
  constructor(

```

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				78

```

private repo: ArticleRepository,
private firebaseStorage: FirebaseStorageService,
private commentService: CommentService,
) {}

async findArticles(page: number, take: number, search?: string) {
  const [articles, count] = await this.repo.findArticles(page, take, search);

  return {
    items: plainToInstance(ArticleShortDto, articles),
    totalCount: count,
    query: {
      page,
      take,
      totalPages: Math.ceil(count / take),
    },
  };
}

async findArticleById(id: number): Promise<Article> {
  return await this.repo.findArticleById(id);
}

async getRandomArticle() {
  return this.repo.getRandomArticle();
}

async findMyDrafts(userId: number) {
  return this.repo.findDraftsByAuthor(userId);
}

async getDraftById(articleId: number, userId: number) {
  return this.repo.findDraftById(articleId, userId);
}

async createArticle(
  article: ArticleCreateDto,
  userId: number,
): Promise<void> {
  await this.repo.createArticle(article, userId);
}

async updateArticle(
  id: number,
  partialArticle: ArticleEditDto,
  file?: Express.Multer.File,
): Promise<void> {
  const { disciplineId, ...rest } = partialArticle;
  let imageUrl: string | undefined;

```

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				79
	.	№ докум.				

```

if (file) {
    const path = await this.firebaseStorage.uploadFile({
        pathname: 'articles',
        originalName: file.originalname,
        buffer: file.buffer,
        metadata: { contentType: file.mimetype },
    });

    imageUrl = await this.firebaseStorage.getFileLink(path);
}

await this.repo.updateArticle(id, {
    ...rest,
    ...(imageUrl ? { mainImgUrl: imageUrl } : {}),
    ...(disciplineId ? { discipline: { id: disciplineId } } : {}),
});
}

async publishArticle(id: number, userId: number): Promise<void> {
    const draft = await this.repo.findDraftById(id, userId);

    if (!draft) {
        throw new NotFoundException('Draft not found');
    }

    await this.repo.updateArticle(id, {
        status: ArticleStatus.PUBLISHED,
    });
}

async revertSoftDelete(id: number): Promise<void> {
    const isAvailableForReversion = await this.repo.existsBy({
        id,
        deletedAt: NotNull,
    });
    if (isAvailableForReversion)
        throw new NotFoundException('Such article may already exist/not exist');

    await this.repo.updateArticle(id, { deletedAt: null });
}

async softDeleteArticle(id: number): Promise<void> {
    await this.repo.softDeleteArticle(id);

    await this.commentService.removeCommentsForEntity({
        entityId: id,
        entitySource: CommentSource.ARTICLE,
    });
}

```

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				80
	.	№ докум.				

```
}
}
```

Лістинг класу ArticleRepository:

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Article } from './article.entity';
import { FindOptionsWhere, ILike, IsNull, Repository } from 'typeorm';
import { ArticleCreateDto } from './dto/article-create.dto';
import { QueryDeepPartialEntity } from 'typeorm/query-builder/QueryPartialEntity';
import {
  ARTICLES_SELECT_MANY_OBJ,
  ARTICLES_SELECT_ONE_OBJ,
} from './article.constants';
import { ArticleStatus } from './enums/article-status.enum';
import { ParagraphRepository } from '../paragraphs/paragraph.repository';
```

```
@Injectable()
export class ArticleRepository {
  constructor(
    @InjectRepository(Article) private repo: Repository<Article>,
    private paragraphRepo: ParagraphRepository,
  ) {}
```

```
  async findArticles(
    page: number = 1,
    take: number = 10,
    search?: string,
  ): Promise<[Article[], number]> {
    return this.repo.findAndCount({
      select: ARTICLES_SELECT_MANY_OBJ,
      where: {
        title: search ? ILike(`%${search}%`) : undefined,
        deletedAt: IsNull(),
        status: ArticleStatus.PUBLISHED,
      },
      order: {
        createdAt: 'DESC',
      },
      take,
      skip: (page - 1) * take,
      relations: ['discipline'],
    });
  }
```

```
  async findArticleById(id: number): Promise<Article> {
    const entity = await this.repo.findOne({
      select: ARTICLES_SELECT_ONE_OBJ,
      where: {
```

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				
	.	№ докум.				81

```

        id,
        deletedAt: IsNull(),
        status: ArticleStatus.PUBLISHED,
    },
    relations: ['paragraphs', 'discipline', 'author'],
});
if (!entity) {
    const isSoftDeleted = await this.repo.exists({
        where: { id },
        withDeleted: true,
    });

    throw new NotFoundException(
        isSoftDeleted
        ? 'This article is currently unavailable, please check later'
        : 'Such article does not exist',
    );
}

return entity;
}

async getRandomArticle() {
    return await this.repo
        .createQueryBuilder('a')
        .select(['a.id', 'a.title', 'a.mainImgUrl'])
        .where('a.status = :status', { status: ArticleStatus.PUBLISHED })
        .orderBy('RANDOM()')
        .findOne();
}

async findDraftsByAuthor(authorId: number): Promise<Article[]> {
    return this.repo.find({
        where: {
            author: {
                id: authorId,
            },
            status: ArticleStatus.DRAFT,
        },
        order: {
            updatedAt: 'DESC',
        },
        select: ARTICLES_SELECT_MANY_OBJ,
    });
}

async findDraftById(id: number, userId: number) {
    const article = await this.repo.findOne({
        where: {
            id,

```

		Близнюк Т.Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Савіцький Р.С				82
	.	№ докум.				

```

    status: ArticleStatus.DRAFT,
    author: {
      id: userId,
    },
  },
  relations: ['author', 'paragraphs', 'discipline'],
});

if (!article) {
  throw new NotFoundException(
    'Draft not found or you do not have access to it',
  );
}

return article;
}

async createArticle(
  article: ArticleCreateDto,
  authorId: number,
): Promise<void> {
  const { disciplineId, ...rest } = article;

  await this.repo.insert({
    ...rest,
    discipline: { id: disciplineId },
    author: { id: authorId },
  });
}

async updateArticle(
  id: number,
  partial: QueryDeepPartialEntity<Article> & {
    paragraphs?: {
      title: string;
      content: string;
      order: number;
    }[];
  },
): Promise<void> {
  const article = await this.repo.findOne({
    where: { id },
    relations: ['paragraphs', 'discipline'],
  });

  if (!article) {
    throw new NotFoundException('Article not found');
  }

  const { paragraphs, ...scalarFields } = partial;

```

```

await this.repo.update(id, scalarFields);

if (paragraphs) {
  await this.paragraphRepo.deleteForArticle(id);

  await this.paragraphRepo.createMany(
    paragraphs.map((p) => ({
      ...p,
      article: { id },
    })),
  );
}

async softDeleteArticle(id: number): Promise<void> {
  const exists = await this.repo.existsBy({ id, deletedAt: IsNull() });
  if (!exists)
    throw new NotFoundException(
      'This article is already soft-deleted or may be removed from the portal',
    );

  await this.repo.softDelete(id);
}

async existsBy(options: FindOptionsWhere<Article>) {
  return await this.repo.existsBy(options);
}
}

```