

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:

«Система онлайн-голосування»

Виконав студент групи ІПЗ-21-1
ОЗОРНІН Ярослав Вікторович

Керівник роботи:
ЛОКТИКОВА Тамара Миколаївна

Рецензент:
ЛОБАНЧИКОВА Надія Миколаївна

Житомир – 2025

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

«26» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ОЗОРНІН Ярослав Вікторович**

Керівник роботи: **ЛОКТИКОВА Тамара Миколаївна**

Тема роботи: **«Система онлайн-голосування»,**

затверджена Наказом закладу вищої освіти від **«14» лютого 2025 р., №61/с**

Вихідні дані для роботи: **об'єктом дослідження є процес організації онлайн-голосувань, що включає створення голосувань, збереження результатів та контроль за їхньою достовірністю, а предметом дослідження є методи та засоби реалізації безпечних та масштабованих систем для голосування, їхні функціональні можливості та переваги у порівнянні з традиційними механізмами проведення голосувань.**

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Локтікова Т.М.	27.02.2025р.	27.02.2025р.
2	Локтікова Т.М.	27.02.2025р.	27.02.2025р.
3	Локтікова Т.М.	27.02.2025р.	27.02.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з розробленого веб-застосунку системи онлайн-голосування та пояснювальної записки. Пояснювальна записка до випускної кваліфікаційної роботи містить 84 сторінки, 30 рисунків та 13 таблиць.

Було створено безпечну, масштабовану та зручну у використанні платформу для проведення онлайн-опитувань та голосувань. Розроблено архітектуру системи, що включає серверну частину на базі фреймворку Spring Boot, яка відповідає за реалізацію бізнес-логіки, управління даними та забезпечення безпеки, та клієнтську частину на основі бібліотеки React, що забезпечує інтерактивний та динамічний користувацький інтерфейс. Система зберігання даних реалізована з використанням СУБД PostgreSQL.

У ході роботи було реалізовано функціонал реєстрації та автентифікації користувачів із застосуванням інструменту JWT, верифікацію електронної пошти, можливості створення та проходження опитувань різних типів, а також перегляд статистики результатів. Окрему увагу приділено розробці системи адміністрування, що включає управління користувачами та їх ролями на основі моделі RBAC, а також механізми модерації контенту. Архітектурні рішення та вибір технологій спрямовані на забезпечення надійності, безпеки та можливості подальшого масштабування розробленої системи онлайн-голосування.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУНОК, ОНЛАЙН-ГОЛОСУВАННЯ, JWT, POSTGRESQL, REACT, SPRING BOOT.

					ІПЗ.КР.Б-121-25-ПЗ				
Змн.	Арк.	№ докум.	Підпис	Дата					
Розроб.		Озорнін Я.В.			Система онлайн-голосування. Пояснювальна записка		Літ.	Арк.	Аркуші
Керівник		Локтікова Т.М						2	84
							ФІКТ Гр. ІПЗ-21-1		
Н. контр.		Тичина О.П.							
Зав. каф.		Вакалюк Т.А.							

ABSTRACT

The bachelor's qualification thesis consists of the developed web application and an explanatory note. The explanatory note to the bachelor's qualification thesis contains 84 pages, 30 figures, and 13 tables.

A secure, scalable, and user-friendly platform for conducting online surveys and voting was developed. The system architecture was designed to include a backend built with the Spring Boot framework, responsible for implementing business logic, data management, and security, as well as a frontend based on the React library, providing an interactive and dynamic user interface. Data storage was implemented using the PostgreSQL DBMS.

During the development process, functionality for user registration and authentication using JWT was implemented, along with email verification, the ability to create and participate in various types of surveys, and viewing result statistics. Special attention was given to developing the administration system, which includes user and role management based on the RBAC model, as well as content moderation mechanisms. The architectural decisions and choice of technologies were aimed at ensuring the reliability, security, and future scalability of the developed online voting system.

KEYWORDS: JWT, ONLINE VOTING, POSTGRESQL, REACT, SPRING BOOT, WEB APPLICATION.

		Озорнін Я.В.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		3

ЗМІСТ

РЕФЕРАТ	2
ABSTRACT	3
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	5
ВСТУП.....	6
РОЗДІЛ 1 АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ ОНЛАЙН-ГОЛОСУВАННЯ	8
1.1 Постановка задачі.....	8
1.2 Аналіз аналогів програмного продукту	10
1.3 Технологічний стек та основні рішення	16
Висновки до першого розділу.....	20
РОЗДІЛ 2 ПРОЕКТУВАННЯ СИСТЕМИ ОНЛАЙН-ГОЛОСУВАННЯ.....	22
2.1 Визначення варіантів використання системи	22
2.2 Розробка бази даних системи.....	26
2.3 Проектування та реалізація алгоритмів роботи серверної частини.....	35
2.4 Проектування та реалізація клієнтської частини.....	46
Висновки до другого розділу	53
РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ОНЛАЙН- ГОЛОСУВАННЯ.....	55
3.1 Порядок встановлення та налаштування параметрів системи	55
3.2 Структура інтерфейсу користувача.....	57
3.3 Порядок роботи користувача з системою.....	63
Висновки до третього розділу	76
ВИСНОВКИ.....	78
ДЖЕРЕЛА ІНФОРМАЦІЇ	81
ДОДАТКИ.....	84

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API (Application Programming Interface) – прикладний програмний інтерфейс, набір визначень та протоколів для створення та інтеграції прикладного програмного забезпечення.

БД (База Даних) – структурований набір даних, що зберігається та управляється електронними засобами.

DTO (Data Transfer Object) – об'єкт передачі даних, простий об'єкт, що використовується для передачі даних між шарами або компонентами системи.

JPA (Java Persistence API) – специфікація Java, що описує управління персистентністю та об'єктно-реляційним відображенням.

JDBC (Java Database Connectivity) – стандартний Java API для підключення до реляційних баз даних.

JPQL (Java Persistence Query Language) – об'єктно-орієнтована мова запитів, визначена як частина специфікації JPA.

JWT (JSON Web Token) – веб-токен JSON, відкритий стандарт для створення токенів доступу на основі JSON.

MVC (Model-View-Controller) – архітектурний патерн, що розділяє дані (Model), користувацький інтерфейс (View) та логіку управління (Controller).

ORM (Object-Relational Mapping) – технологія програмування, що пов'язує бази даних з концепціями об'єктно-орієнтованих мов програмування.

REST (Representational State Transfer) – архітектурний стиль взаємодії компонентів розподіленого застосунку у мережі.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		5

ВСТУП

Актуальність теми полягає у необхідності створення безпечних, прозорих та масштабованих систем для проведення онлайн-голосувань. Сучасні цифрові технології дозволяють значно покращити процес голосування, забезпечуючи його доступність для великої кількості користувачів та мінімізуючи ризики маніпуляцій. Використання веб-платформ для голосування має широку сферу застосування – від соціальних опитувань до офіційних виборів, однак існуючі рішення часто мають обмеження у сфері безпеки, автентифікації користувачів та контролю унікальності голосів.

Метою випускної кваліфікаційної роботи бакалавра є розробка онлайн-системи голосування, яка забезпечить безпеку, масштабованість та гнучкість управління правами доступу.

Для досягнення цієї мети необхідно виконати такі **завдання**:

1. Провести аналіз існуючих платформ для голосування, визначити їхні переваги та недоліки.
2. Сформулювати основні вимоги до системи, враховуючи безпеку, захист від шахрайства та зручність використання.
3. Обґрунтувати вибір технологічного стеку для реалізації проекту.
4. Розробити систему онлайн-голосування, використовуючи сучасні технології бекенду та фронтенду.

Об'єктом дослідження є процес організації онлайн-голосувань, що включає створення голосувань, збереження результатів та контроль за їхньою достовірністю.

Предметом дослідження є методи та засоби реалізації безпечних та масштабованих систем для голосування, їхні функціональні можливості та переваги у порівнянні з традиційними механізмами проведення голосувань.

За темою випускної кваліфікаційної роботи було опубліковано тези:

- 1) Озорнін Я.В., Локтікова Т.М., Порівняльний аналіз сучасних систем онлайн-голосувань. Тези доповідей VII Всеукраїнської науково-технічної конференції «Комп'ютерні технології: інновації, проблеми,

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

рішення», 02-03 грудня 2024 року. Житомир: «Житомирська політехніка», 2024. С. 49-50 [1];

- 2) Озорнін Я.В., Локтікова Т.М. Реалізація автентифікації та авторизації у вебзастосунку голосування з використанням JWT і Spring Security. Тези доповідей Всеукраїнської науково-практичної on-line конференції здобувачів вищої освіти і молодих учених, присвяченої Дню науки, 12-17 травня 2025 року. Житомир: «Житомирська політехніка», 2025. С. 54-55 [2].

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1 АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ ОНЛАЙН-ГОЛОСУВАННЯ

1.1 Постановка задачі

Концепція електронного голосування почала формуватися наприкінці ХХ століття, коли розвиток цифрових технологій відкрив нові можливості для модернізації виборчого процесу. Перші експерименти з використанням електронних систем голосування були спрямовані на оптимізацію процедур підрахунку голосів, зменшення людського чинника та підвищення загальної ефективності виборів. Однак поступово ідея трансформувалась у реальність, що дозволяє громадянам реалізовувати своє виборче право через інтернет за допомогою онлайн-голосування.

Однією з перших країн, яка зробила рішучий крок у цьому напрямку, стала Естонія. У 2005 році вона запровадила систему електронного голосування, яка дозволила громадянам брати участь у парламентських виборах, не виходячи з дому. Цей прорив у галузі е-урядування став знаковим прикладом для інших держав і підтвердив, що використання цифрових інструментів у сфері демократичного волевиявлення може бути як практичним, так і ефективним. Естонський досвід продемонстрував, що за наявності належної технічної інфраструктури та довіри громадян до цифрових сервісів, електронне голосування може стати повноцінною альтернативою традиційним методам.

Незважаючи на очевидні переваги, багато існуючих рішень мають суттєві недоліки. Вони часто характеризуються низьким рівнем безпеки, що робить їх вразливими до маніпуляцій та шахрайства. Окрім того, обмежена масштабованість унеможливорює підтримку великої кількості одночасних користувачів, а складність у використанні обмежує їхню доступність для широкої аудиторії. Зважаючи на ці недоліки, виникає необхідність у розробці зручної, безпечної та ефективної онлайн-системи голосування, яка б відповідала

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

сучасним вимогам веб-технологій та забезпечувала прозорість, надійність і масштабованість.

Основною метою цього проекту є створення повноцінного веб-застосунку для онлайн-голосування, який буде характеризуватися інтуїтивно зрозумілим користувацьким інтерфейсом, високим рівнем безпеки даних та здатністю до масштабування без втрати продуктивності. Планується реалізувати повний цикл розробки від етапу проектування до розгортання готової платформи. Особливу увагу буде приділено архітектурі системи, яка повинна забезпечити ефективну взаємодію між клієнтською та серверною частинами, мінімізувати затримки в обробці запитів та забезпечити гнучкість у подальшому масштабуванні та адаптації системи до нових умов.

Серед ключових етапів реалізації проекту варто відзначити побудову архітектури серверної частини з урахуванням вимог до продуктивності, безпеки та модульності, а також реалізація системи автентифікації, авторизації та управління правами доступу, що дозволяє чітко розмежувати функціональність між різними категоріями користувачів. Подальшими важливими кроками є реалізація основного функціоналу платформи: створення реєстраційних та виборчих модулів, проектування та імплементація бази даних для зберігання інформації про користувачів, голосування, результати, а також розробка механізмів інтеграції між фронтендом і бекендом. Особлива увага приділятиметься забезпеченню цілісності даних, захисту від несанкціонованого доступу та підтримці функціоналу для прозорого підрахунку голосів.

Заключним етапом стане розробка дизайну користувацького інтерфейсу, що відіграє ключову роль у забезпеченні зручності та ефективності взаємодії користувача із системою. Цей етап передбачає створення цілісної візуальної концепції платформи, яка відповідатиме принципам сучасного UX/UI-дизайну та забезпечуватиме інтуїтивно зрозумілу навігацію. Зокрема, мова йде про вибір кольорової гами, типографіки, стилістики елементів, а також логічного розташування функціональних блоків на сторінках застосунку.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

1.2 Аналіз аналогів програмного продукту

Онлайн-голосування та системи для створення опитувань стали невід'ємною частиною сучасного цифрового простору, забезпечуючи зручні інструменти для збору думок та прийняття колективних рішень у різноманітних сферах, від освітніх закладів та корпоративного сектору до громадських ініціатив та соціологічних досліджень. Зважаючи на зростаючу потребу в ефективних та доступних механізмах електронного волевиявлення, при аналізі існуючих рішень важливо ретельно розглянути їхні функціональні можливості, основні переваги та характерні обмеження, щоб чітко визначити унікальну ціннісну пропозицію та позиціонувати розроблюваний програмний продукт на ринку.

Одним із найпоширеніших та найдоступніших інструментів у цій ніші, що часто слугує точкою входу для багатьох користувачів, є Google Forms [3]. Ця платформа здобула широку популярність завдяки своїй винятковій простоті використання, що не вимагає спеціалізованих знань, повній безкоштовності для індивідуальних користувачів та освітніх установ, а також тісній та безшовній інтеграції з іншими популярними сервісами екосистеми Google, такими як Google Sheets для автоматизованого збору та подальшого аналізу відповідей. Google Forms дозволяє користувачам швидко та без зайвих зусиль створювати різноманітні форми опитувань, гнучко налаштовувати типи питань, оперативно збирати відповіді в реальному часі та зручно візуалізувати отримані результати за допомогою вбудованих діаграм та графіків. Інтерфейс платформи є інтуїтивно зрозумілим навіть для користувачів без глибоких спеціальних технічних навичок, а його повна адаптивність до мобільних пристроїв забезпечує комфортний доступ до створення та проходження опитувань з будь-яких пристроїв, будь то настільний комп'ютер, планшет або смартфон. На рис. 1.1 представлений процес створення нової форми у Google Forms, що наочно підтверджує декларовану простоту та мінімалістичний, функціонально орієнтований підхід до дизайну, який робить його привабливим для широкого кола завдань.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

Рисунок 1.1 - Платформа для створення голосувань та опитувань Google Forms

Незважаючи на численні переваги, Google Forms має суттєві обмеження, особливо коли йдеться про проведення голосувань, що вимагають вищого рівня контролю за учасниками та унікальністю голосів. Основною проблемою є відсутність вбудованих механізмів надійної автентифікації користувачів, не пов'язаних з обліковими записами Google, що ускладнює контроль за унікальністю кожного голосу та відкриває можливості для повторних голосувань з однієї IP-адреси або через анонімні сесії. Хоча Google Forms може обмежувати відповіді однією на обліковий запис Google, це не завжди є прийнятним або достатнім рішенням, особливо для аудиторій, які не використовують сервіси Google, або для сценаріїв, де потрібна верифікація учасників за іншими критеріями. Відсутність гнучких інструментів для запобігання недобросовісним діям, окрім базових опцій, робить цю платформу менш придатною для критично важливих голосувань, де цілісність та достовірність результатів мають першочергове значення.

Продовжуючи аналіз існуючих платформ, необхідно розглянути рішення, що забезпечують вищий рівень безпеки та орієнтовані на офіційні або суспільно значущі голосування. Яскравим прикладом такого підходу є вебінтерфейс для голосувань, інтегрований у державний портал «Дія» в Україні [4]. Цей інструмент був розроблений для проведення офіційних та громадських

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

опитувань серед громадян України, і його ключовою перевагою є надзвичайно високий рівень безпеки та верифікації учасників. Завдяки інтеграції з єдиною системою ідентифікації, автентифікація користувачів у «Дії» відбувається через підтверджені державні механізми, такі як електронний цифровий підпис, BankID, або через верифікацію за допомогою паспортних даних чи ID-картки. Такий підхід практично унеможливлює фальсифікацію голосів, оскільки кожен голос жорстко прив'язаний до унікальної, верифікованої особи, що забезпечує максимальну довіру до результатів голосування. Інтерфейс платформи «Дія», представлений на рис. 1.2, розроблений з акцентом на простоту та доступність, пропонуючи користувачам зрозумілий механізм участі в активних опитуваннях, що стосуються державних чи суспільних ініціатив.

Дія Платформа Центрів Я – громадянин

Поділіться своєю думкою!

Вкажіть які онлайн-курси, ви пройшли на платформі? *

Введіть назву

Ваша стаття (виберіть один із варіантів)*

☐ Жінка ☐ Чоловік

Вкажіть, наскільки справдилися ваші очікування від навчання (виберіть один із варіантів) *

Рисунок 1.2 - Веб-інтерфейс на платформі «Дія»

Однак, незважаючи на беззаперечні переваги у сфері безпеки та верифікації для офіційних голосувань, платформа «Дія» має значні функціональні обмеження, що роблять її непридатною для широкого кола інших завдань, які може вирішувати система онлайн-голосувань. По-перше, користувачі системи «Дія» не мають можливості самостійно створювати власні опитування чи голосування. Весь контент та ініціативи формуються централізовано державними органами або уповноваженими структурами. Це виключає використання платформи для внутрішніх корпоративних голосувань, освітніх опитувань, збору думок у межах невеликих спільнот чи для будь-яких

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

інших приватних ініціатив. По-друге, система «Дія» орієнтована виключно на громадян України та використовує національні механізми ідентифікації, що унеможлиблює її застосування на міжнародному рівні або для аудиторій, що не мають доступу до українських систем верифікації.

Завершуючи огляд аналогів, варто звернути увагу на спеціалізовані платформи, розроблені безпосередньо для організації виборчих процесів та голосувань, які вимагають підвищеної надійності та специфічного функціоналу. Election Runner є одним із таких рішень, що позиціонується як інструмент для проведення безпечних та прозорих онлайн-виборів для різноманітних організацій, включаючи освітні заклади, асоціації та бізнеси [5]. Ключовими перевагами цієї платформи є акцент на безпеці, що реалізується через використання шифрування SSL для передачі даних, та різні методи автентифікації виборців, що допомагають забезпечити унікальність голосів. Election Runner пропонує можливості для кастомізації голосувань, дозволяючи налаштовувати список кандидатів або варіантів, визначати часові рамки проведення голосування, а також підтримує голосування в реальному часі з подальшим експортом даних для аналізу результатів. Інтерфейс створення опитувань (рис. 1.3) орієнтований на специфіку виборчих процесів, надаючи інструменти для управління списками виборців та налаштування бюлетенів.

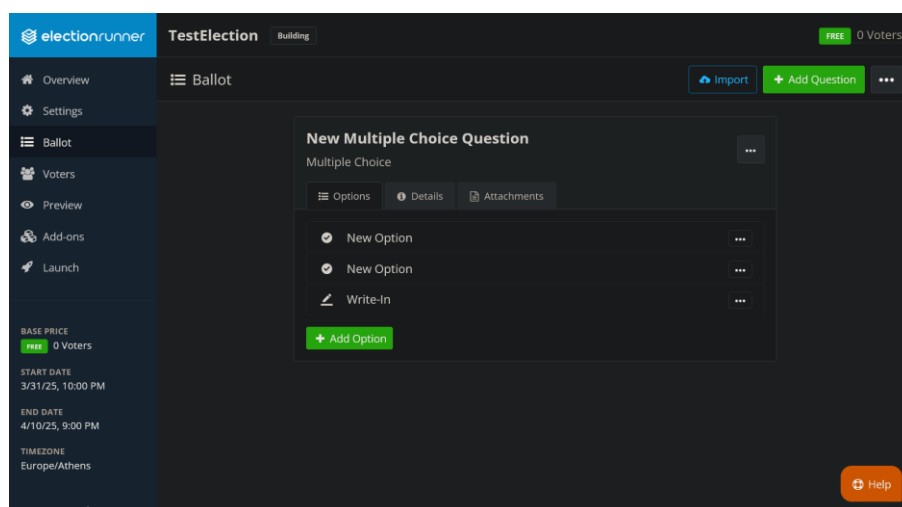


Рисунок 1.3 - Платформа онлайн тестування-голосування ElectionRunner

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

Проте, незважаючи на свою спеціалізацію та фокус на безпеці виборчих процедур, Election Runner має певні недоліки, які обмежують його універсальність. По-перше, сильна орієнтація на виборчі процеси робить платформу менш зручною для проведення звичайних опитувань, збору зворотного зв'язку або неформальних голосувань, де не потрібна така строгість та складність налаштувань, як для виборів. По-друге, значна частина розширеного функціоналу, включаючи підтримку великої кількості виборців або розширені можливості кастомізації, доступна лише у платних версіях продукту. Безкоштовна версія Election Runner зазвичай має суттєві обмеження, наприклад, щодо максимальної кількості учасників голосування (часто до 20 осіб), що робить її практично непридатною для організацій середнього та великого розміру або для проведення масових опитувань. Це може стати значним бар'єром для освітніх закладів, неприбуткових організацій або малих спільнот, які потребують функціонального інструменту, але мають обмежений бюджет.

У табл. 1.1 представлено порівняльну характеристику оглянутих аналогів програмного продукту за основними функціями.

Таблиця 1.1

Порівняльна характеристика аналогів програмного продукту

Функціонал	GoogleForms	«Дія»	Election Runner	Система онлайн-голосувань
Простота	+	-	-	+
Доступ до повного функціоналу без доплат	+	+	-	+
Автентифікація	-	+	+	+
Безпека	-	+	+	+
Кастомізація	-	-	+	-
Доступність	+	-	+	+
Захист від шахрайства	-	-	+	+
Масштабованість	+	-	-	+

Проведений аналіз існуючих платформ для онлайн-голосувань, таких як універсальний та простий Google Forms, високобезпечний державний портал

«Дія» та спеціалізований Election Runner, висвітлив різноманітність підходів та виявив певні ніші, які може ефективно зайняти розроблювана система. На відміну від Google Forms, який, попри свою беззаперечну доступність та легкість у використанні, не пропонує надійних вбудованих механізмів контролю унікальності голосів та верифікації учасників, розроблювана система, завдяки обов'язковій верифікації електронної пошти для виконання ключових функцій, створює необхідний базовий рівень ідентифікації. Це дозволяє асоціювати кожен голос із конкретним, хоча б мінімально перевіреним, користувачем, що значно ускладнює можливість масових анонімних маніпуляцій та надає організаторам голосування вищу ступінь впевненості в достовірності отриманих даних, що є критично важливим для забезпечення прозорості та довіри у багатьох практичних сценаріях застосування.

Водночас, розроблювана система, хоч і не претендує на рівень державної верифікації та безпеки, властивий платформі «Дія», яка є вузькоспеціалізованим інструментом для офіційних опитувань громадян України та обмежена у своїй універсальності й можливості створення користувацьких ініціатив, пропонує значно більшу гнучкість та ширшу сферу застосування. Можливість самостійно створювати, налаштовувати та адмініструвати опитування, залучаючи учасників на основі верифікації електронної пошти, робить розроблюваний продукт придатним для задоволення потреб різноманітних користувачів та організацій. Система дозволяє оперативно розгортати опитування для збору зворотного зв'язку, прийняття колективних рішень у робочих групах, проведення студентських голосувань чи організації громадських обговорень без прив'язки до складних та специфічних державних механізмів автентифікації.

У порівнянні зі спеціалізованими платформами, такими як Election Runner, які часто орієнтовані на формалізовані виборчі процедури та мають значну частину функціоналу у платних версіях з обмеженнями для безкоштовного використання, розроблювана система онлайн-голосування прагне запропонувати більш універсальне, доступне та масштабоване рішення. Не концентруючись виключно на виборчих процесах, вона надає гнучкі інструменти для створення

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		15

різноманітних типів опитувань, адаптованих під потреби широкого кола користувачів. Хоча початковий рівень безпеки та механізми верифікації можуть не відповідати найвищим стандартам, необхідним для офіційних виборів національного рівня, передбачене підтвердження електронної пошти та закладена архітектура, що допускає інтеграцію з іншими системами автентифікації або впровадження додаткових перевірок, забезпечують достатній рівень контролю для більшості повсякденних завдань. Таким чином, розроблюваний продукт позиціонується як збалансоване рішення, що займає нішу між абсолютно відкритими, але менш контрольованими платформами, та високоспеціалізованими або державними системами, пропонуючи доступний, гнучкий та простий у використанні інструмент для широкого спектра завдань онлайн-голосування без значних фінансових витрат або надмірної складності, властивої вузькоспеціалізованим рішенням.

1.3 Технологічний стек та основні рішення

Для створення безпечної, масштабованої та продуктивної системи онлайн-голосування важливо обрати відповідний набір технологій, що буде відповідати основним вимогам до розроблювального продукту.

Вибір технології для розробки серверної частини системи є ключовим рішенням, що безпосередньо впливає на продуктивність, надійність, безпеку та можливості подальшого розвитку проекту. Для системи онлайн-голосування було обрано фреймворк Spring Boot, який є одним із найпопулярніших та найпотужніших рішень у світі розробки сучасних веб-додатків на платформі Java. Цей вибір ґрунтується на комплексі переваг, що роблять Spring Boot оптимальним інструментом для реалізації поставлених перед системою завдань.

Передусім, Spring Boot значно прискорює та спрощує процес розробки завдяки своєму підходу «convention over configuration» та широкому використанню механізмів автоконфігурації [6]. Це дозволяє розробникам зосередитися на бізнес-логіці додатку, а не на рутинних налаштуваннях інфраструктури. Фреймворк усуває необхідність у великій кількості шаблонного

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		16

конфігураційного коду, який часто супроводжує традиційну розробку на платформі Spring. Завдяки наявності готових стартових залежностей (starter dependencies), інтеграція та налаштування ключових компонентів, таких як веб-сервер, інструменти для роботи з базами даних чи модулі безпеки, стає значно простішою та вимагає мінімальних зусиль.

Важливою перевагою є інтеграція Spring Boot з величезною екосистемою Spring Framework. Це надає доступ до перевірених часом та індустріальними стандартами рішень для різних аспектів розробки. Зокрема, Spring Data значно спрощує взаємодію з базами даних, абстрагуючись від деталей конкретної СУБД, а Spring Security надає комплексний та гнучкий інструментарій для реалізації надійних механізмів автентифікації та авторизації користувачів [17]. Для системи онлайн-голосування, де безпека даних та контроль доступу є критично важливими, використання цих готових та добре протестованих модулів є суттєвою перевагою, що підвищує надійність системи.

Масштабованість додатку є ще одним важливим фактором, який враховувався при виборі технології, особливо з огляду на потенційне зростання кількості користувачів та одночасних голосувань у майбутньому. Spring Boot найкраще підходить для побудови як традиційних монолітних додатків, так і сучасних мікросервісних архітектур. Можливість легко розгорнути застосунок або в перспективі декомпонувати його на менші, незалежні сервіси забезпечує гнучкість та дозволяє ефективно розподіляти навантаження. Це є важливим для забезпечення стабільної та безперебійної роботи платформи онлайн-голосування під час пікових навантажень.

Клієнтська частина розроблятиметься з допомогою JavaScript бібліотеки – React, що є потужною бібліотекою для створення динамічних веб-додатків. Завдяки компонентному підходу React дозволяє розробникам легко розділяти інтерфейс на незалежні модулі, що спрощує оновлення, підтримку та розширення системи.

Висока продуктивність React забезпечується завдяки використанню віртуального DOM, який оптимізує оновлення сторінки та мінімізує затримки

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

при рендері змін [8]. Це особливо важливо для інтерактивних застосунків, таких як система голосування, де швидка обробка користувацьких дій є критичною.

Ще однією значною перевагою React є його широка екосистема та велика підтримка спільноти. Завдяки великій кількості бібліотек та інструментів, можна швидко інтегрувати новий функціонал без необхідності писати все з нуля. Гнучкість React дозволяє легко підключати бекенд-сервіси та масштабувати інтерфейс у майбутньому.

При виборі системи управління базами даних (СУБД) для системи онлайн-голосування постало ключове питання вибору між реляційним (SQL) та нереляційним (NoSQL) підходами, кожен з яких має свої специфічні переваги та недоліки, що визначають їхню придатність для конкретних завдань. Реляційні бази даних, такі як MySQL або обрана для даного проекту PostgreSQL, характеризуються зберіганням даних у структурованому вигляді – у таблицях, що складаються з рядків та стовпців, з чітко визначеною наперед схемою. Вони гарантують строгу консистентність даних завдяки вбудованим механізмам транзакцій та відповідності фундаментальним принципам ACID (Atomicity, Consistency, Isolation, Durability). Це забезпечує високу цілісність даних навіть при виконанні одночасних операцій, що є критично важливим аспектом для систем, де точність та надійність даних є пріоритетом, як у випадку з системами голосування. Крім того, стандартизована мова SQL дозволяє виконувати складні аналітичні запити для вибірки та агрегації даних з різних пов'язаних таблиць.

Нереляційні бази даних, такі як MongoDB, Cassandra або Redis, пропонують значно гнучкіші моделі даних (документні, ключ-значення, графові тощо) і зазвичай не вимагають жорсткого дотримання попередньо визначеної схеми. Вони часто оптимізовані для забезпечення горизонтального масштабування та високої доступності, що робить їх популярним вибором для обробки великих обсягів неструктурованих або напівструктурованих даних, а також для систем з дуже високим навантаженням на операції запису. Однак, така гнучкість та масштабованість часто досягаються за рахунок компромісів у строгості консистентності даних, де перевага може надаватися доступності системи.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		18

Для розроблюваної системи онлайн-голосування вимоги до цілісності, надійності та консистентності даних були визначені як першочергові та невід'ємні. Кожен голос, поданий користувачем, має бути точно та однозначно зафіксований, пов'язаний з відповідним голосуванням та, з конкретним користувачем, унеможливлючи будь-яку втрату чи дублювання критично важливої інформації. Саме тому реляційна модель даних, разом із гарантіями ACID, які надає PostgreSQL, була визнана найбільш відповідною цим фундаментальним вимогам. Структура даних системи голосування, що включає користувачів, опитування, питання, варіанти відповідей та власне голоси, характеризується чітко визначеними взаємозв'язками, які легко та ефективно моделюються за допомогою таблиць та зовнішніх ключів у реляційній базі даних. Можливість виконувати складні SQL-запити для детального аналізу результатів, підрахунку голосів за різноманітними критеріями є суттєвою функціональною перевагою. Окрім цього, реляційні бази даних, і PostgreSQL зокрема, пропонують розвинені та перевірені часом механізми безпеки та контролю доступу на рівні таблиць та окремих рядків, що є надзвичайно важливим для захисту чутливих даних голосування [9]. Хоча нереляційні бази даних можуть пропонувати певні переваги у контексті масштабування, очікувані навантаження для типової системи онлайн-голосування під силу сучасним реляційним СУБД, таким як PostgreSQL, особливо за умови належної оптимізації запитів та структури бази. Таким чином, переваги реляційної моделі даних у забезпеченні цілісності, структурованості та надійності збереження даних стали вирішальними факторами при виборі PostgreSQL як основи для бази даних цього проекту.

Безпека є ключовим аспектом будь-якої системи онлайн-голосування. Для захисту даних користувачів та результатів голосування використовується механізм автентифікації JWT (JSON Web Token). JWT дозволяє передавати зашифровану інформацію між клієнтом і сервером, уникаючи необхідності збереження сесій на сервері [18]. Це підвищує рівень безпеки та спрощує управління користувачами в розподілених системах.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		19

Для контролю доступу використовується модель RBAC (Role-Based Access Control). Вона дозволяє керувати правами користувачів, розподіляючи їх за ролями, що мінімізує ризик несанкціонованих дій. Наприклад, організатори голосувань матимуть розширені можливості управління, тоді як учасники – обмежений набір функцій для голосування.

Висновки до першого розділу

У першому розділі було проведено комплексний аналіз предметної області, пов'язаної з розробкою систем онлайн-голосування. Детально розглянуто актуальність створення безпечних, прозорих та масштабованих платформ для електронного збору голосів, що зумовлено зростаючою потребою у доступних та ефективних інструментах для прийняття колективних рішень у різних сферах суспільного та професійного життя. В ході дослідження було проаналізовано низку існуючих аналогів, зокрема такі поширені інструменти як Google Forms, державний портал «Дія» та спеціалізовану платформу Election Runner. Це порівняння дозволило виявити їхні переваги, такі як простота використання у Google Forms, високий рівень державної верифікації у «Дії» та спеціалізація на виборчих процесах у Election Runner, а також суттєві обмеження, серед яких недостатній контроль унікальності голосів, обмежена гнучкість у створенні користувацьких опитувань, або вузька спрямованість та потенційна вартість для широкого загалу.

На основі виявлених переваг та недоліків існуючих рішень було сформульовано ключові вимоги до розроблюваної системи онлайн-голосування. Пріоритетними критеріями визначено забезпечення належного рівня безпеки, включаючи верифікацію користувачів та захист від потенційних маніпуляцій, гнучкість у створенні та налаштуванні різноманітних типів голосувань, масштабованість для підтримки різної кількості учасників та опитувань, а також інтуїтивно зрозумілий користувацький інтерфейс. З урахуванням цих вимог було здійснено обґрунтований вибір технологічного стеку для реалізації проекту. Для серверної частини було обрано фреймворк Spring Boot, що забезпечує швидку

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		20

розробку, надійність та широкі можливості для інтеграції систем безпеки. Клієнтська частина буде реалізована з використанням бібліотеки React, яка дозволяє створювати динамічні та інтерактивні користувацькі інтерфейси. В якості системи управління базами даних обрано PostgreSQL, що гарантує цілісність даних та підтримує складні запити, а для механізмів автентифікації та авторизації передбачено використання JWT та моделі RBAC. Запропоновані технологічні рішення спрямовані на створення продуктивної, безпечної та гнучкої системи, здатної задовольнити сучасні потреби в організації онлайн-голосувань.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

РОЗДІЛ 2 ПРОЕКТУВАННЯ СИСТЕМИ ОНЛАЙН-ГОЛОСУВАННЯ

2.1 Визначення варіантів використання системи

Аналіз вимог до системи онлайн-голосування дозволив чітко визначити ключові ролі користувачів та основні сценарії їхньої взаємодії з платформою. На основі цього аналізу було розроблено діаграму варіантів використання на рис. 2.1, яка візуалізує функціональні можливості системи та взаємозв'язки між акторами та прецедентами, забезпечуючи уніфіковане розуміння меж та можливостей розроблюваного програмного продукту.

У системі онлайн-голосування виокремлено три основні типи акторів: Гість, Авторизований користувач та Адміністратор, причому роль Модератора є підмножиною або тісно пов'язаною з можливостями Адміністратора, як це показано на діаграмі. Гість представляє незареєстрованого користувача, чії можливості обмежені базовими функціями. Зокрема, Гість може виконувати реєстрацію в системі для отримання розширених прав, авторизуватися, якщо він вже має обліковий запис, а також переглядати загальну статистику по системі.

Авторизований користувач, після успішного входу в систему, отримує значно ширший спектр можливостей. Окрім функцій, доступних Гостю (за винятком реєстрації, оскільки він вже зареєстрований), Авторизований користувач може безпосередньо проходити голосування. Цей процес може включати перегляд результатів голосувань, у яких він брав участь або які є публічно доступними. Важливою функцією є можливість залишати коментарі до опитувань та подавати скарги на контент або користувачів, що порушують правила системи. Ключовою можливістю авторизованого користувача є створення власних голосувань, що включає визначення питань, варіантів відповідей та налаштувань опитування.

Роль Адміністратора передбачає найповніший набір повноважень для управління всією системою онлайн-голосування. Адміністратор успадковує всі можливості Авторизованого користувача та додатково отримує доступ до

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

функцій адміністрування. До таких функцій належить видалення будь-яких голосувань, незалежно від того, хто є їх автором. Критично важливою є можливість модерування ролей користувачів, що дозволяє адміністратору призначати або відкликати певні ролі, тим самим керуючи рівнями доступу до функціоналу системи. Адміністратор також має інструменти для перегляду активності користувачів, що допомагає моніторити дії в системі та виявляти потенційні порушення. Обробка скарг, поданих користувачами, є ще однією важливою функцією адміністратора, що включає розгляд скарги, прийняття рішення та інформування заявника. Адміністратор має повноваження банити користувачів, тимчасово або постійно обмежуючи їх доступ до системи, а також, у крайніх випадках, повністю видаляти дані користувачів з системи. Модератор, як окрема або споріднена роль, зазвичай має доступ до функцій видалення голосувань, обробки скарг та бану користувачів, але може не мати повноважень на модерування ролей чи повне видалення користувачів, залежно від конфігурації системи.

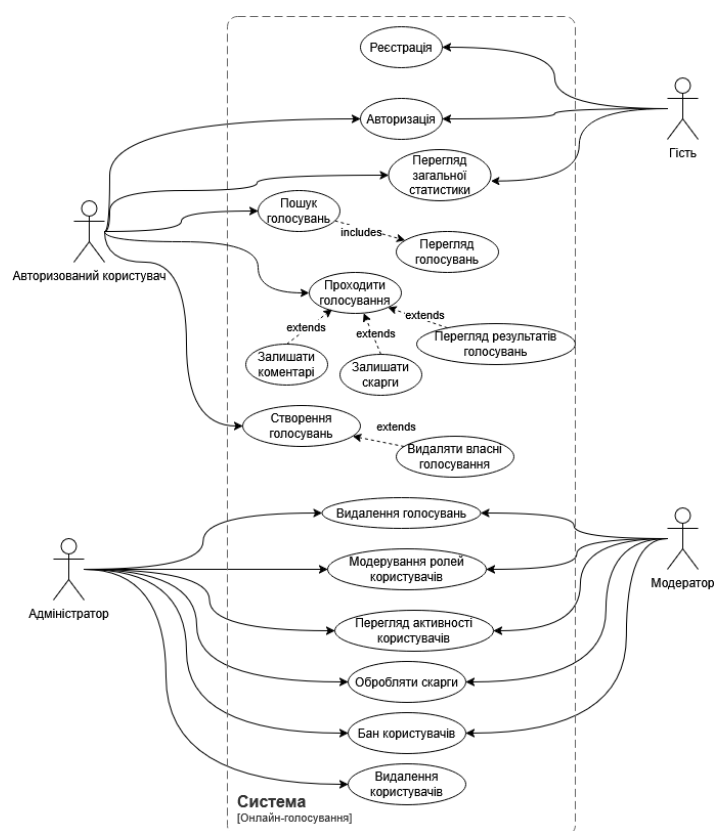


Рисунок 2.1 – Діаграма використання системи

Визначимо вимоги до розроблюваної системи.

Проектування системи онлайн-голосування базується на аналізі предметної області, вивченні існуючих рішень і визначенні загальних цілей проекту. Враховуючи обраний технологічний стек – React для клієнтської частини та Spring Boot для серверної, а також багатошарову архітектуру (Controller-Service-Repository) – були сформульовані ключові вимоги до майбутньої системи.

Архітектура проекту передбачає використання Spring Boot для серверної частини, з чітким розділенням відповідальності між контролерами, сервісами та репозиторіями. Контролери оброблятимуть HTTP-запити, сервісний шар реалізовуватиме бізнес-логіку, а репозиторії забезпечуватимуть доступ до бази даних. На боці клієнта система буде реалізована у вигляді односторінкового додатку за допомогою бібліотеки React, що дозволить дотриматися компонентного підходу до розробки користувацького інтерфейсу. Для обміну даними між клієнтом і сервером буде застосовано RESTful API, що гарантуватиме стандартизовану комунікацію між окремими частинами системи.

Особлива увага приділяється питанню автентифікації та авторизації користувачів. Для цього планується впровадження механізму на основі JWT у поєднанні з можливостями Spring Security. Система повинна забезпечувати розмежування доступу відповідно до ролей користувачів, серед яких передбачені ролі Гостя, Користувача, Модератора та Адміністратора. Контролери перевірятимуть валідність токенів та відповідність запитів правам доступу перед виконанням будь-яких операцій.

Функціонал керування голосуваннями включає проектування відповідних REST-контролерів для CRUD-операцій над голосуваннями, питаннями та варіантами відповідей. Бізнес-логіка, пов'язана зі створенням, редагуванням, публікацією та завершенням голосувань, а також обробкою відповідей користувачів, буде сконцентована у сервісному шарі. Репозиторії забезпечать ефективну взаємодію із базою даних PostgreSQL, використовуючи переваги Spring Data JPA.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

Процес голосування повинен бути максимально зручним для користувачів. Необхідно передбачити API-ендпоінти для отримання активних голосувань, перегляду питань та надсилання відповідей. Обробка відповідей буде здійснюватися на рівні сервісного шару із подальшим записом у базу даних. Важливим аспектом є впровадження механізмів захисту від повторного голосування та можливих фальсифікацій.

Користувацький інтерфейс має бути інтуїтивно зрозумілим та адаптивним до різних розмірів екранів. В межах клієнтської частини передбачено створення компонентів для відображення списку голосувань, окремих голосувань з питаннями, форм для створення й редагування опитувань, панелі адміністратора і модератора, а також сторінок зі статистикою та результатами голосування. Кожен з компонентів повинен ефективно взаємодіяти з REST API для обробки даних у реальному часі.

Обробка даних між клієнтською та серверною частинами здійснюватиметься за допомогою спеціально спроектованих Data Transfer Objects, що дозволить уникнути прямого витоку внутрішніх структур даних. Для підвищення надійності запитів буде реалізована валідація даних як на рівні контролерів, так і у сервісному шарі.

Безпека системи є одним із пріоритетних завдань проекту. На рівні Spring Boot буде застосовано стандартні практики захисту веб-додатків: механізми протидії CSRF та XSS-атакам, налаштування CORS-політик. Паролі користувачів зберігатимуться у хешованому вигляді, що унеможливить їх компрометацію у разі витоку бази даних. Окрім цього, проектування архітектури враховуватиме захист цілісності даних під час голосування.

Важливо забезпечити високу продуктивність системи. Для цього планується оптимізація запитів до бази даних PostgreSQL з метою досягнення мінімального часу відповіді (не більше однієї секунди для стандартних операцій). Клієнтська частина має бути оптимізована для швидкого завантаження і рендерингу навіть при великій кількості даних.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

Масштабованість системи передбачає можливість горизонтального розширення серверної частини на базі Spring Boot, а також масштабування бази даних PostgreSQL відповідно до зростання навантаження.

Важливим показником є також надійність та доступність сервісу. Система повинна бути розроблена таким чином, щоб забезпечити безперебійну роботу не менше ніж 98% часу. Для підвищення стійкості передбачено обробку помилок і винятків на всіх рівнях архітектури, а також механізми резервного копіювання бази даних.

Підтримуваність та тестованість системи будуть забезпечені шляхом дотримання принципів чистого коду і SOLID у процесі розробки як серверної, так і клієнтської частини. Активно використовуватиметься система контролю версій, наприклад Git. Компоненти, особливо сервіси, проектуватимуться з урахуванням можливості легкого тестування, що включає створення юніт-тестів. Чітке розділення обов'язків між шарами Controller, Service та Repository сприятиме зрозумілості і підтримуваності коду.

Клієнтська частина також повинна відповідати сучасним стандартам сумісності. Застосунок має коректно працювати у популярних веб-браузерах, зокрема Chrome, Firefox, Edge та Safari.

Усі вищезгадані вимоги сформують основу для наступних етапів розробки – детального проектування бази даних, опису алгоритмів і побудови загальної архітектури системи.

2.2 Розробка бази даних системи

Для забезпечення надійного та структурованого зберігання всієї інформації, що генерується та обробляється в системі онлайн-голосування, було прийнято рішення про використання реляційної системи управління базами даних. Після ретельного аналізу вимог проекту та врахування переваг різних СУБД, вибір було зроблено на користь PostgreSQL, відомої своєю стабільністю, розширюваністю та підтримкою складних транзакцій. Спираючись на попередньо визначені ключові сутності системи, такі як користувачі,

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

опитування, питання, варіанти відповідей, голоси та коментарі, а також на детально прописані вимоги до функціонування платформи, була розроблена комплексна логічна структура бази даних. Ця структура, візуалізована на рис. 2.2, ретельно відображає та охоплює всі основні аспекти взаємодії користувачів із системою, включаючи процеси створення нових облікових записів, автентифікації, авторизації, формування та публікації опитувань, а також безпосереднього проходження голосувань зареєстрованими учасниками. Окрім того, спроектована схема бази даних передбачає ефективні механізми для фіксації та управління комунікацією між учасниками, наприклад, через систему коментарів або подання скарг, що є невід'ємною частиною сучасної інтерактивної платформи.

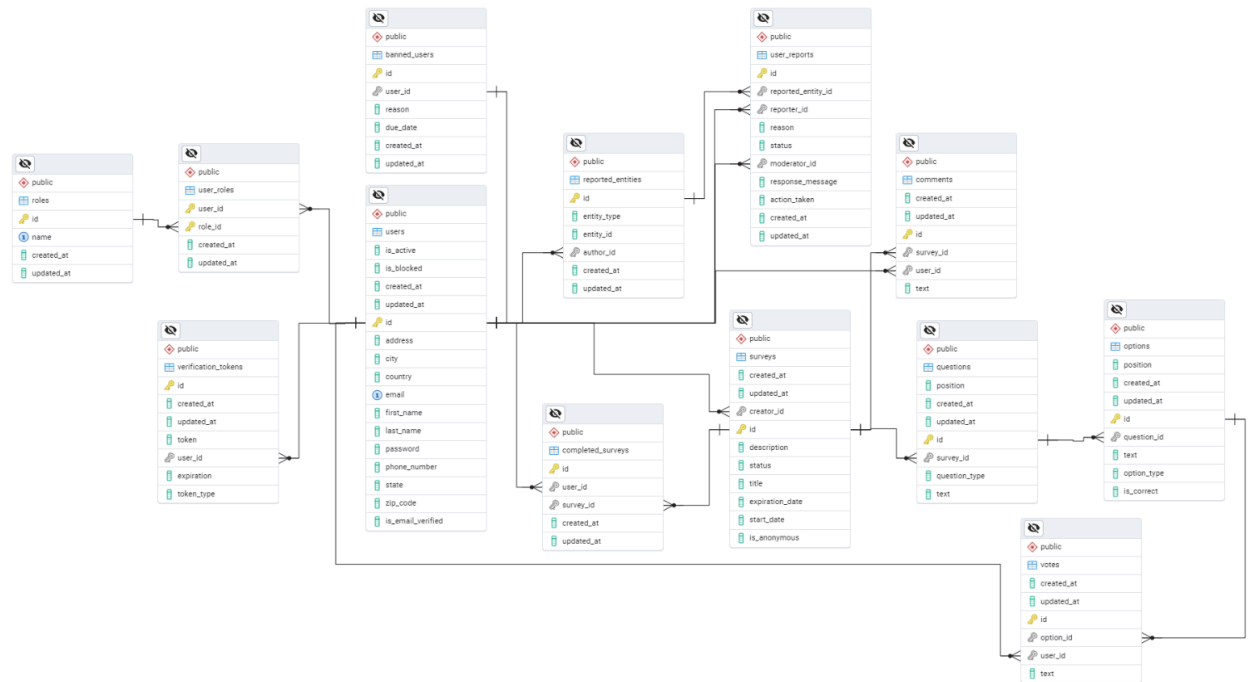


Рисунок 2.2 – Структура бази даних

Детально розглянемо таблиці бази даних.

Таблиця users (табл. 2.1) є центральною сутністю в системі, оскільки вона зберігає всю основну інформацію про зареєстрованих користувачів. Кожен запис у цій таблиці представляє унікального учасника системи, ідентифікованого за допомогою поля id типу uuid. Окрім персональних даних, таких як ім'я, прізвище та контактна інформація, таблиця містить критично важливі для функціонування

системи поля. Поле email є унікальним і використовується як логін, а password зберігає захешований пароль для автентифікації. Статусні поля, зокрема is_active, is_blocked та is_email_verified, дозволяють керувати життєвим циклом та правами доступу акаунта. Ця таблиця має численні зв'язки з іншими таблицями системи, виступаючи в ролі джерела даних про авторів опитувань (surveys), коментарів (comments), голосів (votes) та скарг (user_reports), що робить її ядром усієї моделі даних.

Таблиця 2.1

Опис таблиці Users

Поле	Тип даних	Короткий опис
id	uuid	Унікальний ідентифікатор користувача.
email	character varying(255)	Електронна пошта (унікальна).
password	character varying(255)	Захешований пароль.
first_name	character varying(255)	Ім'я.
last_name	character varying(255)	Прізвище.
phone_number	character varying(255)	Номер телефону.
address, city, state, country, zip_code	character varying(255)	Адресні дані.
is_active	boolean	Прапорець активності акаунта.
is_blocked	boolean	Прапорець блокування.
is_email_verified	boolean	Прапорець підтвердження email.
created_at	timestamp	Дата та час реєстрації.
updated_at	timestamp	Дата та час оновлення.

Для реалізації гнучкої системи розмежування доступу (Role-Based Access Control, RBAC) використовуються дві таблиці: roles та user_roles (табл. 2.2 - 2.3). Таблиця roles є простим довідником, що містить перелік усіх можливих ролей у системі (наприклад, «USER», «MODERATOR», «ADMIN»), де кожна роль має унікальну назву. Таблиця user_roles є асоціативною (або зв'язуючою) і реалізує

зв'язок «багато-до-багатьох» між користувачами та ролями. Кожен запис у цій таблиці пов'язує конкретного користувача (`user_id`) з певною роллю (`role_id`). Така архітектура дозволяє динамічно призначати користувачам одну або кілька ролей, що визначатиме їхні права та можливості в системі без необхідності змінювати основну логіку додатку.

Таблиця 2.2

Опис таблиці Roles

Поле	Тип даних	Короткий опис
id	uuid	Унікальний ідентифікатор ролі.
name	character varying(255)	Назва ролі (унікальна).
created_at	timestamp	Дата та час створення.
updated_at	timestamp	Дата та час оновлення.

Таблиця 2.3

Опис таблиці User Roles

Поле	Тип даних	Короткий опис
user_id	uuid	Ідентифікатор користувача.
role_id	uuid	Ідентифікатор ролі.
created_at	timestamp	Дата та час призначення ролі.
updated_at	timestamp	Дата та час оновлення.

Таблиця `surveys` (табл. 2.4) є центральною сутністю, що описує опитування. Кожен запис представляє унікальне опитування з його назвою, описом та налаштуваннями. Поле `creator_id` встановлює зв'язок «багато-до-одного» з таблицею `users`, чітко визначаючи автора. Важливу роль відіграють поля, що керують життєвим циклом опитування: `status` (наприклад, «чернетка», «активне», «завершене»), `start_date` та `expiration_date` визначають період його доступності. Поле `is_anonymous` є ключовим для забезпечення конфіденційності, дозволяючи приховати інформацію про респондентів. На цю таблицю

посилаються questions, comments та completed_surveys, що робить її головним об'єктом, навколо якого групується пов'язаний контент.

Таблиця 2.4

Опис таблиці Surveys

Поле	Тип даних	Короткий опис
id	uuid	Унікальний ідентифікатор опитування.
creator_id	uuid	Ідентифікатор користувача-творця.
title	character varying(255)	Назва опитування.
description	text	Детальний опис опитування.
status	character varying(255)	Статус опитування.
start_date	timestamp	Дата та час початку.
expiration_date	timestamp	Дата та час закінчення.
is_anonymous	boolean	Прапорець анонімності.
created_at	timestamp	Дата та час створення.
updated_at	timestamp	Дата та час оновлення.

Структура кожного опитування визначається ієрархією з двох таблиць: questions та options (табл. 2.5 - 2.6). Таблиця questions містить текст та тип кожного питання (наприклад, вибір одного варіанта, вибір кількох, вільна текстова відповідь). Кожне питання нерозривно пов'язане з певним опитуванням через зовнішній ключ survey_id, утворюючи зв'язок «один-до-багатьох». Поле position забезпечує правильний порядок відображення питань. У свою чергу, таблиця options зберігає можливі варіанти відповідей для кожного питання, маючи зв'язок «один-до-багатьох» з questions через question_id. Поле is_correct дозволяє використовувати опитування як тести з автоматичною перевіркою, а option_type забезпечує логіку вибору.

Таблиця 2.5

Опис таблиці Questions

Поле	Тип даних	Короткий опис
id	uuid	Унікальний ідентифікатор питання.
survey_id	uuid	Ідентифікатор опитування.
text	text	Текст питання.
question_type	character varying(255)	Тип питання.
position	integer	Порядковий номер питання.

Таблиця 2.6

Опис таблиці Options

Поле	Тип даних	Короткий опис
id	uuid	Унікальний ідентифікатор варіанту.
question_id	uuid	Ідентифікатор питання.
text	text	Текст варіанту відповіді.
position	integer	Порядковий номер варіанту.
is_correct	boolean	Прапорець правильності відповіді.

Для збору та аналізу результатів опитувань використовуються дві ключові таблиці. Таблиця votes (табл. 2.7) фіксує кожну відповідь користувача. Кожен запис у ній означає, що користувач (user_id) вибрав певний варіант (option_id) або надав текстову відповідь (text). Ця таблиця є основою для підрахунку голосів та формування статистики. Таблиця completed_surveys (табл. 2.8), у свою чергу, відіграє роль маркера, що фіксує сам факт завершення опитування користувачем. Унікальне обмеження на пару (user_id, survey_id) гарантує, що один користувач не може пройти те саме опитування двічі. Ця таблиця важлива для відстеження активності користувачів та розрахунку коефіцієнта участі в опитуваннях.

Таблиця 2.7

Опис таблиці Votes

Поле	Тип даних	Короткий опис
id	uuid	Унікальний ідентифікатор голосу.
option_id	uuid	Ідентифікатор обраного варіанту.
user_id	uuid	Ідентифікатор користувача.
text	text	Відповідь для текстових питань.

Таблиця 2.8

Опис таблиці Completed Surveys

Поле	Тип даних	Короткий опис
id	uuid	Унікальний ідентифікатор запису.
user_id	uuid	Ідентифікатор користувача.
survey_id	uuid	Ідентифікатор опитування.

Таблиця comments (табл. 2.9) реалізує функціонал обговорення опитувань, підвищуючи інтерактивність платформи. Кожен коментар прив'язаний до конкретного опитування через зовнішній ключ survey_id та до автора через user_id. Це дозволяє групувати дискусії навколо відповідних тем та ідентифікувати учасників обговорення. Текстовий вміст коментаря зберігається у полі text. Ця таблиця, як і інші види контенту, що генерується користувачами, може бути об'єктом модерації через систему скарг.

Таблиця 2.9

Опис таблиці Comments

Поле	Тип даних	Короткий опис
id	uuid	Унікальний ідентифікатор коментаря.
survey_id	uuid	Ідентифікатор опитування.
user_id	uuid	Ідентифікатор автора коментаря.
text	text	Вміст коментаря.

Для забезпечення порядку, захисту користувачів та підтримки високого рівня безпеки на платформі було спроектовано спеціалізовану підсистему модерації. Вона покликана реагувати на порушення правил поведінки та регламенту користування, а також гарантувати своєчасну обробку скарг і вжиття відповідних заходів реагування. Архітектурно підсистема реалізована у вигляді трьох взаємопов'язаних таблиць, кожна з яких виконує свою специфічну функцію в процесі модерації.

Центральною складовою є таблиця `reported_entities` (табл. 2.10), що виконує роль універсального реєстру об'єктів, на які було подано скарги. Завдяки використанню полів `entity_type` (тип об'єкта, наприклад: `USER`, `SURVEY`, `COMMENT`) та `entity_id` (ідентифікатор конкретної сутності), ця таблиця забезпечує гнучкість та масштабованість моделі, дозволяючи розширювати перелік типів об'єктів, які можуть бути об'єктом модерації, без необхідності суттєвих змін у структурі бази даних.

Інформація про самі скарги зберігається в таблиці `user_reports` (табл. 2.11). Вона містить усі ключові атрибути, необхідні для подальшого аналізу й обробки скарги: ідентифікатор користувача, який ініціював скаргу (`reporter_id`), обґрунтування або причину звернення (`reason`), поточний статус розгляду (`status`), а також результати модераторської перевірки – зокрема, ідентифікатор модератора (`moderator_id`), відповідь або коментар щодо розгляду скарги (`response_message`) і вжиті заходи (`action_taken`). Така деталізація дозволяє зберігати історію прийнятих рішень, що може бути корисно як для внутрішнього аудиту, так і для прозорості комунікації з користувачами.

У разі, коли наслідком розгляду скарги є блокування користувача, відповідна інформація фіксується в таблиці `banned_users` (табл. 2.12). У ній зберігаються дані про заблокованого користувача (`user_id`), обґрунтування рішення про блокування, а також, у випадках тимчасових обмежень, дата завершення блокування (`due_date`). Це дозволяє системі автоматично знімати обмеження після закінчення встановленого строку без участі модератора.

Таким чином, описана структура реалізує повний цикл обробки скарг – від фіксації повідомлення про порушення до прийняття рішення та застосування відповідних санкцій. Такий підхід не лише підвищує ефективність реагування на інциденти, але й сприяє підтриманню довіри користувачів до платформи, створюючи умови для безпечної та цивілізованої взаємодії в межах системи.

Таблиця 2.10

Опис таблиці User reports

Поле	Тип даних	Короткий опис
id	uuid	Унікальний ідентифікатор скарги.
reported_entity_id	uuid	Ідентифікатор сутності, на яку скаржаться.
reporter_id	uuid	Ідентифікатор скаржника.
reason	text	Причина скарги.
status	character varying(20)	Статус розгляду.
moderator_id	uuid	Ідентифікатор модератора.
action_taken	character varying(50)	Вжиті заходи.

Таблиця 2.11

Опис таблиці Reported Entites

Поле	Тип даних	Короткий опис
id	uuid	Унікальний ідентифікатор запису.
entity_type	character varying(20)	Тип сутності, на яку подано скаргу ('USER', 'SURVEY', 'COMMENT').
entity_id	uuid	Ідентифікатор конкретної сутності.

author_id	uuid	Ідентифікатор автора сутності.
created_at	timestamp	Дата та час створення.
updated_at	timestamp	Дата та час оновлення.

Таблиця 2.12

Опис таблиці Banned users

Поле	Тип даних	Короткий опис
id	uuid	Унікальний ідентифікатор запису про блокування.
user_id	uuid	Ідентифікатор заблокованого користувача.
reason	character varying(200)	Причина блокування.
due_date	timestamp	Дата та час закінчення терміну блокування.
created_at	timestamp	Дата та час створення запису.
updated_at	timestamp	Дата та час останнього оновлення запису.

Окрім таблиць, що реалізують бізнес-логіку, у схемі присутні технічні таблиці. `verification_tokens` використовується для безпекових механізмів, таких як підтвердження електронної пошти чи скидання пароля. Вона зберігає унікальний токен, прив'язаний до користувача (`user_id`), його тип та термін дії. Таблиця `flyway_schema_history` є службовою таблицею інструменту міграцій Flyway і не бере участі в логіці додатку, а лише відстежує версії схеми БД.

2.3 Проектування та реалізація алгоритмів роботи серверної частини

Для забезпечення безперебійної роботи онлайн-системи голосування розроблені алгоритми, що охоплюють реєстрацію та автентифікацію користувачів, створення та проведення голосувань, а також захист і моніторинг виконання ключових операцій.

Одним із базових елементів системи є генерація унікальних ідентифікаторів для основних сутностей. Для цієї мети обрано UUID версії 7, що базується на мікросекундних часових мітках. Такий підхід не лише забезпечує глобальну унікальність, але й покращує продуктивність індексації в базі даних за рахунок майже послідовного порядку вставки записів.

Безпека системи реалізується за допомогою інтеграції Spring Security (додаток А). На етапі налаштування механізмів безпеки було впроваджено фільтрацію запитів (додаток Б), що дозволяє розмежовувати доступ до ресурсів залежно від ролі користувача. Для управління автентифікацією застосовується підхід на основі JWT токенів. При кожній взаємодії користувача із захищеними ресурсами система розшифровує переданий токен, дістаючи з нього необхідну інформацію про акаунт користувача. Далі, за допомогою Spring Security Context, ця інформація стає доступною для використання у бізнес-логіці додатка, що забезпечує надійний контроль доступу без необхідності повторного звернення до бази даних на кожен запит. Щоб запобігти потенційним атакам типу DDOS, реалізовується алгоритм обмеження частоти запитів (rate-limit filter). Він дозволяє виставити фіксовану кількість запитів, для кожного користувача, що буде відновлюватися через деякий період часу (додаток В). Такий підхід дозволить підвищити загальну безпеку, а також попередити просадки продуктивності під час високого

Особлива увага приділяється реєстрації та авторизації користувачів. Під час реєстрації користувач надає необхідні дані, після чого система створює новий запис у базі даних із зашифрованим паролем та згенерованим UUID.

У випадку авторизації подані облікові дані перевіряються, після чого у разі успіху генерується новий JWT токен, який містить основну інформацію про користувача, включно з його роллю та ідентифікатором. Цей токен передається клієнту для подальшого використання у запитах. На клієнті токен зберігається у Local Storage та передається при кожному запиті як authorization request header.

Моніторинг важливих операцій здійснюється через впровадження аспектно-орієнтованого програмування (AOP). Було реалізовано спеціальний

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

аспект (додаток Г), який автоматично логуватиме ключові дії користувачів, включно зі спробами авторизації, створенням голосувань та голосуванням за варіанти відповідей. Такий підхід дає змогу зберігати журнал подій без втручання у бізнес-логіку основного коду, що підвищує його чистоту і спрощує подальший супровід.

Процес створення нового голосування нерозривно пов'язаний із забезпеченням того, що лише автентифіковані та авторизовані користувачі можуть виконувати цю дію. З цією метою система використовує можливості фреймворку Spring Security, який надає інструменти для управління контекстом безпеки поточного запиту [12]. Ключовим елементом у цьому процесі є метод `getCurrentUser()` (представлений у додатку Д), який відповідає за отримання інформації про поточного автентифікованого користувача.

При спробі створити голосування, система спочатку викликає цей метод або аналогічну логіку для перевірки статусу користувача. Робота методу `getCurrentUser()` починається зі звернення до `SecurityContextHolder.getContext().getAuthentication()`. `SecurityContextHolder` є центральним сховищем у Spring Security, яке зберігає `SecurityContext`. `SecurityContext`, у свою чергу, асоційований з поточним потоком виконання і містить об'єкт `Authentication`. Об'єкт `Authentication` інкапсулює інформацію про автентифіковану сутність, включаючи її ідентифікатор (`principal`), облікові дані (`credentials`) та надані повноваження (`authorities/roles`). Цей об'єкт зазвичай встановлюється раніше в ланцюжку фільтрів безпеки, наприклад, після успішної валідації JWT токена, надісланого клієнтом.

Далі, у методі `getCurrentUser()`, виконується перевірка: `if (authentication != null && authentication.getPrincipal() instanceof User)`. Перша частина умови, `authentication != null`, переконується, що об'єкт `Authentication` взагалі існує в поточному контексті безпеки. Якщо його немає, це означає, що користувач не пройшов автентифікацію. Друга частина умови, `authentication.getPrincipal() instanceof User`, перевіряє, чи є об'єкт `principal`, отриманий з `Authentication`, екземпляром очікуваного класу `User`. `Principal` представляє ідентифіковану

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

сутність, і в даній реалізації передбачається, що це об'єкт кастомного класу User, який містить детальну інформацію про користувача, таку як його унікальний ідентифікатор, ім'я, роль у системі тощо.

Якщо обидві умови задовольняються, тобто користувач автентифікований і його principal є об'єктом типу User, метод повертає цей об'єкт User, приведений до відповідного типу: `return (User) authentication.getPrincipal();`. Отриманий об'єкт User тепер може бути використаний для подальших операцій, наприклад, для присвоєння його як автора створюваного голосування, для перевірки специфічних прав на створення контенту або для логування дій користувача.

У сучасних веб-додатках, створених за допомогою фреймворку Spring Boot, контролери відіграють центральну роль у обробці вхідних HTTP-запитів та формуванні відповідних HTTP-відповідей. Вони є ключовим компонентом архітектурного патерну Model-View-Controller (MVC), діючи як посередники між користувацьким інтерфейсом (або клієнтським додатком) та бізнес-логікою системи. У контексті Spring, `@RestController` є спеціалізованою версією традиційного `@Controller`, розробленою спеціально для побудови RESTful веб-сервісів, які обмінюються даними переважно у форматах JSON або XML, а не генерують HTML-сторінки для відображення у браузері.

Основна відмінність `@RestController` від звичайного `@Controller` полягає у тому, що `@RestController` є комбінацією двох анотацій: `@Controller` та `@ResponseBody` [12]. Анотація `@Controller` позначає клас як компонент, здатний обробляти запити, що робить його доступним для механізму диспетчеризації Spring MVC. Анотація `@ResponseBody`, коли застосовується на рівні класу (як це відбувається неявно з `@RestController`) або на рівні окремого методу-обробника, вказує Spring MVC, що значення, повернуте методом, має бути безпосередньо записане у тіло HTTP-відповіді. Це означає, що Spring не намагатиметься знайти та відобразити шаблон представлення (view), а натомість використає відповідні `HttpMessageConverter` для серіалізації повернутого об'єкта у відповідний формат, зрозумілий клієнту. Таким чином, `@RestController` ідеально підходить для створення API, де відповіді є даними, а не повноцінними веб-сторінками.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

У життєвому циклі обробки запиту в Spring Boot, `@RestController` займає позицію після серії фільтрів (таких як фільтри безпеки Spring Security або кастомні фільтри) та перед безпосереднім виконанням бізнес-логіки. Коли надходить HTTP-запит, `DispatcherServlet`, центральний компонент Spring MVC, аналізує URL запиту та HTTP-метод. На основі цієї інформації та мапінгів, визначених у контролерах (за допомогою анотацій типу `@RequestMapping`, `@GetMapping`, `@PostMapping` тощо), `DispatcherServlet` знаходить відповідний метод-обробник у відповідному `@RestController`. Перед викликом цього методу Spring MVC може виконати ряд проміжних дій, таких як прив'язка параметрів запиту до аргументів методу, валідація вхідних даних, та автентифікація/авторизація користувача.

Після того, як `DispatcherServlet` визначив цільовий метод у `@RestController`, контролер перебирає на себе відповідальність за координацію подальшої обробки запиту. Важливою функцією контролера є делегування виконання основної бізнес-логіки спеціалізованим сервісним компонентам. Контролери не повинні містити складну бізнес-логіку; їхнє завдання – перетворити вхідні дані, викликати відповідні методи сервісного шару, передавши їм необхідну інформацію, та підготувати дані, отримані від сервісів, для відправки клієнту у вигляді HTTP-відповіді. У наданому прикладі коду `SurveyController` (додаток Л) яскраво демонструє цей підхід. Він анотований `@RestController` та має базовий мапінг `/api/surveys`. Усі його методи-обробники, такі як `getMySurveys`, `getSurveys`, та `exportSurvey`, приймають параметри запиту (наприклад, через `@RequestParam`, `@PathVariable`, `@ModelAttribute`), а потім викликають відповідні методи `SurveyService` (наприклад, `service.getUserSurveys(...)`, `service.getSurveys(...)`, `service.exportSurvey(...)`) для виконання основної роботи. Після отримання результату від сервісу, контролер обгортає його в `ResponseEntity` для більш тонкого контролю над HTTP-статусом та заголовками відповіді, і повертає клієнту. Таким чином, `@RestController` діє як тонкий шар, що пов'язує HTTP-інтерфейс з внутрішньою бізнес-логікою системи, забезпечуючи чітке розділення відповідальностей.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

Анотація `@PreAuthorize` є ключовим інструментом у Spring Security для реалізації декларативної безпеки на рівні методів. Ця анотація використовує потужну мову виразів Spring Expression Language (SpEL), що дозволяє визначати надзвичайно гнучкі та складні правила доступу. Коли метод анотовано `@PreAuthorize`, перевірка безпеки відбувається перед його виконанням. Якщо вираз, вказаний в анотації, обчислюється як `false`, доступ до методу буде заборонено, і система згенерує виняток `AccessDeniedException`, запобігаючи виконанню бізнес-логіки неавторизованим користувачем. Такий підхід дозволяє перенести логіку авторизації з тіла методу безпосередньо в його оголошення, що робить код більш чистим, читабельним та легким для супроводу.

У рамках Spring Expression Language, вираз `hasAnyAuthority(...)` є одним із найпоширеніших для перевірки повноважень користувача. Його призначення – перевірити, чи володіє поточний автентифікований суб'єкт хоча б одним із перерахованих у списку повноважень (authorities). Наприклад, вираз `@PreAuthorize("hasAnyAuthority('ROLE_ADMIN', 'ROLE_MODERATOR')")` означає, що доступ до методу отримають лише ті користувачі, які мають або повноваження `ROLE_ADMIN`, або `ROLE_MODERATOR`. Це забезпечує гнучкість у налаштуванні доступу, коли виконання певної операції дозволено кільком різним групам користувачів (додаток II). Важливо зазначити, що концепція «повноваження» (authority) є більш гранулярною, ніж «роль», і може представляти не тільки загальну роль користувача, але й конкретний дозвіл на дію, що дозволяє будувати більш деталізовані та безпечні системи контролю доступу [17].

Одним із ключових аспектів розробки програмного забезпечення є організація шару доступу до даних (Data Access Layer), який відповідає за взаємодію додатку з базою даних. У рамках даної роботи для реалізації цього шару було обрано технологічний стек, що поєднує специфікацію Java Persistence API (JPA), її реалізацію у вигляді фреймворку Hibernate, та абстракцію вищого рівня Spring Data JPA. Такий підхід дозволяє досягти високого рівня

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		40

продуктивності розробки, забезпечуючи при цьому надійність, гнучкість та легкість у супроводженні коду.

Основою для взаємодії з реляційними базами даних у сучасних Java-додатках є специфікація Java Persistence API. Вона виступає як стандарт, що визначає єдиний набір інтерфейсів та анотацій для реалізації об'єктно-реляційного відображення (Object-Relational Mapping, ORM). Головне призначення JPA полягає у стандартизації процесу збереження Java-об'єктів у базі даних, що забезпечує портативність коду між різними ORM-провайдерами. Використання стандартних анотацій, таких як @Entity, @Id та @Table, дозволяє декларативно описувати відповідність між класами моделі даних та таблицями бази даних, не прив'язуючись до конкретної реалізації ORM-фреймворку (додаток Ж).

Hibernate є провідною та найбільш поширеною реалізацією специфікації JPA. Цей потужний ORM-фреймворк бере на себе виконання всіх складних низькорівневих завдань, пов'язаних із персистентністю даних [7]. Hibernate аналізує JPA-анотації, визначені у сутностях додатку, та на їх основі автоматично генерує SQL-запити. Він керує життєвим циклом об'єктів, відкриттям та закриттям сесій, управлінням транзакціями, а також реалізує механізми кешування першого та другого рівнів для оптимізації продуктивності. Таким чином, Hibernate є тим «двигуном», що виконує всю реальну роботу по перетворенню об'єктів у реляційні дані та навпаки.

Практичне застосування описаної архітектури найбільш наочно демонструється через взаємодію між класами-сутностями (Entity) та інтерфейсами-репозиторіями (Repository). Шляхом успадкування користувацького інтерфейсу від JpaRepository (додаток З), Spring Data JPA автоматично активує механізм, що динамічно створює повноцінну реалізацію шару доступу до даних. Це забезпечує набір готових методів для виконання CRUD-операцій (створення, читання, оновлення, видалення) та, що важливо, дозволяє формувати запити на основі назв методів (derived queries). Такий підхід

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

усуває необхідність ручного написання SQL-запитів для більшості стандартних сценаріїв взаємодії з базою даних.

Однак, коли логіка вибірки даних стає складнішою і не може бути виражена через іменування методів, Spring Data JPA надає потужніший інструмент – мову запитів JPQL (Java Persistence Query Language). За допомогою анотації `@Query` розробник може визначити власний JPQL-запит безпосередньо над методом репозиторію. На відміну від нативного SQL, JPQL є об'єктно-орієнтованою мовою, що оперує не таблицями та стовпцями, а класами-сутностями та їхніми полями. Провайдер персистентності, в даному випадку Hibernate, відповідає за трансляцію цього об'єктно-орієнтованого запиту в оптимізований нативний SQL, враховуючи діалект конкретної бази даних. Такий механізм дозволяє виконувати складні запити, зберігаючи при цьому високий рівень абстракції та портативність коду.

У сучасних багат шарових архітектурах програмних систем, зокрема у веб-додатках, побудованих з використанням Spring Boot для серверної частини та React для клієнтської, ефективний та безпечний обмін даними між шарами має критичне значення. Для вирішення цього завдання широко застосовується патерн проектування Data Transfer Object (DTO). DTO являють собою прості об'єкти, основне призначення яких полягає у передачі даних між різними компонентами системи (додаток І), наприклад, між сервісним шаром та контролером, або між сервером та клієнтом. Вони інкапсулюють набір атрибутів, що представляють певну структуру даних, але, на відміну від сутностей моделі предметної області, DTO зазвичай не містять бізнес-логіки та не пов'язані безпосередньо з механізмами персистентності JPA. Використання DTO дозволяє відокремити структуру даних, що передається через API, від внутрішньої структури сутностей бази даних, забезпечуючи гнучкість, зменшуючи зв'язність між компонентами та підвищуючи безпеку, оскільки дозволяє вибірково експонувати лише необхідні поля, приховуючи внутрішні деталі реалізації або чутливу інформацію.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		42

Валідація даних, що надходять від клієнта у вигляді DTO, є невід'ємною частиною забезпечення коректності та безпеки системи. Бібліотека Jackson, що широко використовується у Spring Boot для серіалізації та десеріалізації JSON, інтегрується з механізмами валідації, такими як Bean Validation API [26]. Це дозволяє розробникам декларативно визначати правила валідації безпосередньо в класах DTO за допомогою анотацій (наприклад, @NotNull, @Size, @Pattern). При отриманні HTTP-запиту, Spring MVC автоматично виконує валідацію DTO (додаток Е), і у випадку виявлення невідповідностей, генерує помилку, яку можна централізовано обробити, не допускаючи некоректні дані до бізнес-логіки.

Перетворення між сутностями предметної області та об'єктами DTO є рутинною, але важливою операцією. Для автоматизації цього процесу та уникнення написання великої кількості шаблонного коду використовуються спеціалізовані бібліотеки мапінгу, однією з яких є MapStruct. MapStruct – це генератор коду, який створює реалізації маперів під час компіляції на основі інтерфейсів, анотованих спеціальними анотаціями. Це забезпечує високу продуктивність, оскільки згенерований код є простим Java-кодом без використання рефлексії під час виконання.

У представленому прикладі коду інтерфейс SurveyMapper (додаток Є) демонструє типові сценарії використання MapStruct. Метод SurveyDTO toDTO(Survey survey) призначений для перетворення сутності Survey на SurveyDTO. Анотація @Mapping(target = "creatorId", source = "creator.id") вказує на необхідність мапінгу поля id з вкладеного об'єкта creator сутності Survey на поле creatorId в SurveyDTO. Анотація @Mapping(target = "questions", ignore = false) підтверджує, що поле questions має бути оброблене мапером; якщо типи Question та QuestionDTO відрізняються, MapStruct намагатиметься знайти або згенерувати відповідний мапер для них. Метод Survey toEntity(SurveyDTO surveyDTO) виконує зворотне перетворення – з SurveyDTO на сутність Survey.

Також слід зазначити метод, що відрізняється за використанням - метод void updateSurveyFromPatch(SurveyPatchDTO patch, @MappingTarget Survey

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

survey). Він призначений для часткового оновлення існуючої сутності Survey даними з SurveyPatchDTO. Анотація @MappingTarget вказує, що параметр survey є цільовим об'єктом, який буде модифіковано. Важливою є анотація @BeanMapping(nullValuePropertyMappingStrategy = NullValuePropertyMappingStrategy.IGNORE), яка інструктує MapStruct ігнорувати null-значення полів у patch DTO під час мапінгу. Таким чином, лише ті поля сутності survey, для яких у patch передані ненульові значення, будуть оновлені, що є стандартною практикою для реалізації PATCH-операцій. Крім того, наявність окремого SurveyPatchDTO з обмеженим набором полів (title, description, expirationDate, isAnonymous) дозволяє чітко визначити, які атрибути можуть бути змінені через цей механізм оновлення. Метод SurveyInfoDTO toInfoDTO(Survey survey) з анотацією @Mapping(target = "numberOfQuestions", expression = "java(survey.getQuestions().size())") демонструє можливість використання Java-виразів для обчислення значень полів DTO, що дозволяє формувати спеціалізовані представлення даних, наприклад, для відображення зведеної інформації. Таким чином, використання DTO у поєднанні з валідацією та інструментами автоматичного мапінгу, такими як MapStruct, значно спрощує розробку, підвищує надійність та підтримуваність коду системи онлайн-голосування.

У сучасній архітектурі веб-додатків, побудованих з використанням платформи Spring Boot, важливе місце займає ефективна обробка виняткових ситуацій. Цей аспект безпосередньо впливає на надійність, передбачуваність роботи системи, а також на якість взаємодії з клієнтською стороною. Замість того, щоб використовувати численні блоки try-catch у кожному контролері, Spring MVC пропонує централізований механізм обробки винятків через спеціальні класи, анотовані @ControllerAdvice або @RestControllerAdvice. Такий підхід дозволяє відокремити логіку обробки помилок від бізнес-логіки, що, у свою чергу, покращує читабельність та підтримуваність коду.

Клас GlobalExceptionHandler (додаток Ї), який позначено анотацією @RestControllerAdvice, виступає у ролі глобального перехоплювача винятків, що

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

виникають під час обробки запитів будь-яким контролером, анотованим `@RestController`. У межах цього класу використовуються методи з анотацією `@ExceptionHandler`, які визначають, як саме слід реагувати на винятки певного типу. Після викидання винятку в межах контролера або іншого компонента обробки запиту, Spring виконує пошук відповідного обробника у всіх класах, позначених `@RestControllerAdvice`. Якщо відповідний обробник знаходиться, він викликається, і його результат, як правило, формується у вигляді об'єкта `ResponseEntity`, що й повертається клієнту.

Централізація логіки обробки помилок дозволяє уникнути дублювання коду та забезпечує уніфікованість у представленні повідомлень про помилки. Завдяки цьому клієнтська сторона отримує відповіді у стандартизованому форматі, що значно покращує зручність взаємодії з API. Крім того, внутрішні деталі реалізації, включаючи технічні стектрейси, не передаються клієнту, що позитивно впливає на безпеку системи. Інтеграція з компонентом `MessageSource` (додаток К) забезпечує підтримку інтернаціоналізації, дозволяючи повертати повідомлення відповідно до мови клієнта. Такий механізм сприяє підвищенню зручності використання додатка у багатомовному середовищі. Звільнення контролерів від обробки винятків також позитивно впливає на якість коду, дозволяючи зосередитися виключно на реалізації бізнес-логіки.

Реалізація `GlobalExceptionHandler` охоплює низку обробників для стандартних і спеціалізованих винятків. Зокрема, метод, що перехоплює `Exception`, слугує загальним обробником на випадок, якщо не знайдено більш специфічного. Він формує відповідь із кодом помилки 500 та повідомленням, яке витягується з винятку. Це забезпечує захист від непередбачених ситуацій.

Обробник для `MethodArgumentNotValidException` реагує на порушення правил валідації аргументів, анотованих `@Valid`, наприклад, під час перевірки DTO. Він формує відповідь зі статусом 400 та картою повідомлень про помилки, які генеруються відповідно до локалі користувача.

Обробка користувацького винятку `UserIsBannedException` передбачає повернення статусу 403 та формування локалізованого повідомлення, що містить

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		45

дані про термін блокування користувача та причину. Аналогічно, обробник `InvalidDataException` відповідає за ситуації, коли дані, передані клієнтом, не відповідають бізнес-правилам. У таких випадках формується відповідь з кодом 400 та локалізованим описом помилки.

У системі передбачено окремі обробники для винятків, пов'язаних із відмовою в доступі чи авторизації. Це стосується як користувацького винятку `AuthorizationDeniedException`, так і стандартного `AccessDeniedException`, який генерується Spring Security. В обох випадках клієнту повертається статус 403.

Випадки, коли жоден обробник не знайдений для певного запиту, перехоплюються обробником `NoHandlerFoundException`. Його коректна робота вимагає попереднього налаштування параметрів Spring Boot. У відповідь повертається код 404, що сигналізує про відсутність відповідного ендпоінта.

Користувацький виняток `ApiException` дозволяє задавати динамічний статус відповіді, оскільки значення `HttpStatus` витягується безпосередньо з поля об'єкта винятку. Це забезпечує гнучкість у визначенні поведінки API при різних типах помилок.

У випадку з `ExpiredJwtException`, який сигналізує про завершення терміну дії токена JWT, формується відповідь з кодом 401, що означає необхідність повторної автентифікації користувача.

У класі `GlobalExceptionHandler` передбачено допоміжний приватний метод `buildErrorResponse`. Цей метод інкапсулює логіку формування об'єкта `ErrorResponse` разом із `ResponseEntity`, забезпечуючи використання локалізованих повідомлень на основі `messageSource`. Такий підхід сприяє підтримці консистентності у форматуванні помилок, що повертаються клієнту.

2.4 Проектування та реалізація клієнтської частини

Клієнтська частина системи онлайн-голосування реалізована як сучасний односторінковий вебзастосунок (SPA), що базується на бібліотеці React. Такий підхід дозволяє забезпечити високу швидкість реакції інтерфейсу,

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

безперервність взаємодії користувача з системою, а також гнучкість у реалізації складної логіки взаємодії з сервером.

У розробці сучасних клієнтських веб-додатків, зокрема тих, що побудовані на бібліотеці React, ефективна взаємодія з серверною частиною через HTTP-запити є фундаментальною потребою. Для спрощення та стандартизації цього процесу широко використовується бібліотека Axios, яка є популярним HTTP-клієнтом, що працює як у браузері, так і в середовищі Node.js [14]. Axios надає зручний, заснований на Promise, API для виконання HTTP-запитів, пропонуючи низку переваг порівняно з нативним fetch API, таких як автоматичне перетворення даних JSON, захист від CSRF, та, що особливо важливо, можливість використання перехоплювачів (interceptors) для глобальної обробки запитів та відповідей.

Представлений фрагмент коду (додаток М) демонструє створення та конфігурацію екземпляра Axios, призначеного для всіх API-запитів у додатку. Спочатку створюється екземпляр api за допомогою axios.create(), що дозволяє визначити базову конфігурацію, яка буде застосовуватися до всіх запитів, зроблених через цей екземпляр. Параметр baseURL встановлюється зі змінної оточення import.meta.env.VITE_API_URL, що є практикою для конфігурування URL сервера залежно від середовища розгортання (розробка, тестування, продакшн). Також встановлюється стандартний заголовок Content-Type зі значенням application/json, що вказує серверу, що тіло запиту, якщо воно присутнє, буде у форматі JSON.

Ключовою особливістю Axios, продемонстрованою у коді, є використання перехоплювачів запитів (api.interceptors.request.use). Перехоплювачі запитів дозволяють модифікувати конфігурацію запиту перед тим, як він буде фактично відправлений на сервер. У даному випадку, перед кожним запитом виконується функція, яка отримує об'єкт конфігурації config. Першочергово, вона намагається отримати JWT токен з localStorage. Якщо токен знайдено, він додається до заголовків запиту як Authorization з префіксом Bearer, що є стандартним механізмом для передачі токенів автентифікації. Надзвичайно

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

важливою є також наступна дія: `config.headers["Accept-Language"] = i18n.language || "en";`. Цей рядок додає заголовок `Accept-Language`, значенням якого є поточна мова, встановлена у бібліотеці інтернаціоналізації `i18n` (наприклад, `i18next`), або ж англійська мова (`"en"`) за замовчуванням, якщо мова не визначена. Надсилання цього заголовка дозволяє серверу ідентифікувати бажану мову клієнта та повертати локалізовані дані або повідомлення про помилки, що значно покращує користувацький досвід для багатомовних додатків. Після модифікації конфігурації, вона повертається для подальшого виконання запиту.

Аналогічно до перехоплювачів запитів, у коді налаштований перехоплювач відповідей (`api.interceptors.response.use`). Цей механізм дозволяє глобально обробляти відповіді від сервера перед тим, як вони будуть передані до коду, що ініціював запит (тобто, до блоку `.then()` або `.catch()` відповідного `Promise`). Перша функція-обробник викликається для успішних відповідей (статус `2xx`). Друга функція-обробник призначена для перехоплення помилок. Якщо сталася помилка відповіді (`error.response` існує), код намагається відобразити повідомлення про помилку користувачеві за допомогою бібліотеки `toast`. Повідомлення береться з `error.response.data?.message` або, якщо воно відсутнє, використовується стандартне повідомлення «`Something went wrong!`», попередньо перекладене функцією `t` з бібліотеки інтернаціоналізації. Особлива увага приділяється помилкам зі статусом `401 (Unauthorized)`, що зазвичай вказує на недійсний або прострочений токен. У такому випадку токен видаляється з `localStorage`, і користувач перенаправляється на сторінку логіна (`navigateTo("/login")`). Якщо ж `error.response` відсутній, це може свідчити про мережеву помилку, і користувачеві відображається відповідне повідомлення. В обох випадках помилку, `Promise.reject(error)` повертається для того, щоб помилка могла бути оброблена і далі у ланцюжку `Promise`, якщо це необхідно.

Таким чином, використання налаштованого екземпляра `Axios` з перехоплювачами запитів та відповідей дозволяє інкапсулювати загальну логіку, пов'язану з HTTP-взаємодією, таку як додавання токенів автентифікації,

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		48

встановлення заголовків локалізації, а також глобальну обробку помилок та управління сесіями користувача, роблячи код React-додатку більш чистим, структурованим та легким для підтримки.

Центральним компонентом, що відповідає за структуру та навігацію додатку, є App.ts (додаток Н). Тут визначається загальна структура маршрутів за допомогою компонентів Routes та Route з react-router-dom [15]. Кожен Route пов'язує певний URL-шлях (path) з відповідним компонентом (element), який буде відображено при переході за цим шляхом. Наприклад, кореневий шлях / веде до HomePage, шлях /login – до LoginPage і так далі. Важливою особливістю є використання MainLayout як обгортки для всіх маршрутів, що дозволяє створити єдиний шаблон сторінки з навігаційними елементами, хедером, футером тощо, який буде присутній на всіх сторінках додатку. Компонент ToastContainer з бібліотеки react-toastify інтегрований для відображення сповіщень користувачеві.

Для забезпечення контролю доступу та управління станом автентифікації використовуються два провайдери контексту: AuthProvider та RBACProvider. AuthProvider (додаток О) відповідає за управління інформацією про поточного користувача, його автентифікацію та сесію. Він зберігає об'єкт користувача (UserDTO), надає функції для входу, реєстрації та виходу, а також визначає, чи є користувач автентифікованим та чи має він ролі адміністратора або модератора. Важливим аспектом є ефект useEffect всередині AuthProvider, який при завантаженні додатку намагається валідувати наявний у localStorage токен, викликаючи API для отримання профілю користувача. Це дозволяє відновити сесію користувача без необхідності повторного входу, якщо токен ще дійсний. Стан isLoading використовується для індикації процесу валідації токена, що дозволяє відображати стан завантаження на сторінках, захищених ProtectedRoute.

RBACProvider (додаток П), у свою чергу, використовує інформацію про користувача з AuthContext для надання функцій перевірки ролей та прав доступу. Він інкапсулює логіку визначення, чи має користувач певну роль (hasRole), чи

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		49

має хоча б одну з перерахованих ролей (`hasAnyRole`), а також чи може він отримати доступ до сторінки, що вимагає певних ролей (`canAccessPage`). Окремі методи, такі як `canCreateSurveys`, `canPassSurveys`, `canWriteComments`, `isAdmin`, `isModerator`, надають більш семантично зрозумілі перевірки специфічних прав. Логіка `canAccessPage` враховує, що адміністратори та модератори за замовчуванням мають доступ до сторінок, якщо не вказано інше. Використання `useMemo` для створення об'єкта `value` у `RBACProvider` оптимізує продуктивність, запобігаючи зайвим перерендерінгам компонентів, що споживають цей контекст, якщо об'єкт користувача не змінився.

Ключовим елементом системи безпеки є компонент `ProtectedRoute` (додаток Р). Він обгортає маршрути, доступ до яких має бути обмежений. `ProtectedRoute` використовує хуки `useAuth` та `useRBAC` для отримання інформації про стан автентифікації та ролі користувача. Перед відображенням дочірнього компонента (власне сторінки), `ProtectedRoute` виконує кілька перевірок. По-перше, якщо дані автентифікації ще завантажуються (`isLoading` з `AuthContext`), відображається індикатор завантаження (`CircularProgress`). По-друге, якщо маршрут вимагає автентифікації (`requireAuth` встановлено в `true`, що є значенням за замовчуванням) і користувач не автентифікований (`!isAuthenticated`), відбувається перенаправлення на сторінку `/login`. По-третє, якщо для доступу до маршруту визначено список необхідних ролей (`requiredRoles`) і користувач не має жодної з них (перевіряється через `canAccessPage` з `RBACContext`), він перенаправляється на головну сторінку `/`. Лише якщо всі перевірки пройдено успішно, дочірній компонент відображається. Такий підхід дозволяє декларативно захищати маршрути, наприклад, сторінка `/dashboard` доступна лише для користувачів з ролями `ADMIN`, `MODERATOR` або `CAN_CREATE_SURVEYS`, а адміністративні сторінки `/admin/users` та `/admin/reports` – лише для `ADMIN` та `MODERATOR`. Це забезпечує гнучкий та централізований контроль доступу до різних частин React-додатку.

Практика структурування коду, при якій логіка взаємодії з API для певних сутностей або функціональних блоків інкапсулюється в окремі файли, є широко

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		50

розповсюдженим та рекомендованим підходом у розробці сучасних веб-додатків. Такий підхід, як видно на прикладі файлу (додаток С), що містить функції login, register, verifyEmail та інші, пов'язані з автентифікацією, приносить значні переваги у плані організації, підтримки та масштабування кодової бази.

Одним із ключових позитивних аспектів такого структурування є дотримання принципу єдиної відповідальності (Single Responsibility Principle) та покращення модульності системи. Кожен такий файл, наприклад, authApi.js, surveyApi.js або commentsApi.js, концентрує в собі всю логіку, пов'язану з виконанням HTTP-запитів до конкретної групи ендпоінтів сервера. Це означає, що всі деталі, що стосуються URL-адрес, методів запитів (POST, GET тощо), формування тіла запиту та обробки специфічних відповідей для певної сутності, локалізовані в одному місці. Таке чітке розмежування відповідальностей робить код більш зрозумілим, оскільки не потрібно шукати відповідні API-виклики по всьому проекту – вони зібрані у відповідному модулі.

Інкапсуляція API-логіки у спеціалізовані модулі значно підвищує підтримуваність та спрощує внесення змін. Якщо, наприклад, змінюється URL ендпоінта для реєстрації користувачів або формат даних, що надсилаються чи отримуються, ці зміни потрібно буде внести лише в одному файлі. Це мінімізує ризик виникнення помилок у різних частинах програми, які могли б використовувати цей ендпоінт, та суттєво скорочує час, необхідний для рефакторингу чи адаптації до змін на сервері. Крім того, налагодження проблем, пов'язаних із взаємодією з API, стає простішим, оскільки точка входу для всіх запитів до певної групи ресурсів чітко визначена.

Для підвищення зручності користувацького досвіду та загальної інтуїтивності інтерфейсу при створенні та редагуванні опитувань, зокрема при налаштуванні варіантів відповідей на питання, була реалізована функціональність перетягування опцій (додаток С). Цей механізм дозволяє користувачам легко змінювати порядок варіантів відповідей шляхом простого переміщення їх за допомогою миші, що є значно швидшим та наочнішим способом, ніж, наприклад, ручне редагування порядкових номерів. Для реалізації

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

цієї можливості була використана спеціалізована бібліотека для створення інтерфейсів з підтримкою drag-and-drop, як видно з використання компонентів DragDropContext, Droppable та Draggable.

Представлений фрагмент коду демонструє, як саме ця функціональність інтегрована для списку опцій. Компонент DragDropContext виступає як обгортка для всієї області, де можливе перетягування, та відповідає за управління станом операції drag-and-drop. Його ключовий атрибут onDragEnd приймає функцію handleDragEnd, яка викликається після завершення перетягування і саме в ній реалізується логіка оновлення порядку опцій у стані компонента на основі початкової та кінцевої позиції переміщеного елемента. Всередині DragDropContext розташований компонент Droppable з унікальним ідентифікатором options-droppable, який визначає зону, куди можна «кинути» перетягуваний елемент. Атрибут isDropDisabled={disabled} дозволяє динамічно вимикати можливість перетягування, наприклад, якщо опитування вже опубліковане або користувач не має прав на редагування.

Кожен окремий варіант відповіді зі списку sortedOptions обгортається у компонент Draggable. Кожен Draggable елемент має унікальний draggableId (зазвичай це id самої опції) та index, що відповідає його поточній позиції у списку. Візуальним індикатором можливості перетягування слугує іконка DragIndicatorIcon, яка обробляється через dragProvided.dragHandleProps. Ці пропси, надані бібліотекою, забезпечують правильну поведінку елемента як «ручки» для перетягування. Стилiзація, така як зміна прозорості елемента під час перетягування (dragSnapshot.isDragging ? 0.7 : 1) та зміна курсору на «grab» при наведенні, додатково покращує візуальний зворотний зв'язок та інтуїтивність взаємодії. Таким чином, використання бібліотеки для реалізації drag-and-drop функціональності дозволило створити більш гнучкий та зручний інтерфейс для управління порядком варіантів відповідей, що позитивно впливає на загальний досвід користувача при роботі з системою створення опитувань.

Після того, як користувач завершує операцію перетягування варіанта відповіді і спрацьовує функція, передана в onDragEnd (тобто handleDragEnd),

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		52

відбувається не лише оновлення візуального порядку елементів на стороні клієнта. Внутрішня логіка цієї функції передбачає обчислення нової послідовності всіх варіантів відповідей для даного питання. Ця оновлена послідовність, що відображає новий порядок, встановлений користувачем, потім готується для відправки на сервер. Для забезпечення персистентності змін, тобто для збереження нового порядку опцій у базі даних, виконується відповідний API-запит. Цей запит може надсилати або повний оновлений список опцій з їх новими порядковими номерами, або лише інформацію про ті опції, чия позиція змінилася. Аналогічно до того, як у функції `handleOptionChange` зміни окремої опції (наприклад, її тексту або статусу `isCorrect`) відправляються на сервер через `patchOption` для часткового оновлення, зміна порядку опцій також синхронізується з бекендом, гарантуючи, що при наступному завантаженні питання варіанти відповідей будуть відображені у встановленому користувачем порядку.

Висновки до другого розділу

У другому розділі було здійснено комплексне проектування системи онлайн-голосування, що базується на попередньо визначених функціональних, нефункціональних вимогах та вимогах безпеки. На серверній стороні спроектовано багатошарову архітектуру з використанням фреймворку Spring Boot, що забезпечує чітке розмежування відповідальності між контролерами, сервісами та репозиторіями. Ключовим елементом взаємодії є RESTful API, захищений механізмами автентифікації та авторизації на основі JWT токенів та Spring Security, доповнений фільтрами обмеження частоти запитів та аспектно-орієнтованим логуванням. Для персистентності даних розроблено детальну реляційну схему бази даних PostgreSQL, яка відображає сутності користувачів, опитувань, голосів, коментарів та механізми модерації. Алгоритми роботи серверної частини, включаючи обробку запитів, бізнес-логіку та взаємодію з базою даних через Spring Data JPA, були спроектовані з акцентом на ефективність, безпеку та масштабованість.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		53

Проектування клієнтської частини системи онлайн-голосування було спрямоване на створення інтерактивного односторінкового застосунку (SPA) на базі бібліотеки React. Розроблено компонентну архітектуру, механізми управління станом за допомогою AuthContext та RBACContext для автентифікації та розмежування прав доступу, а також систему навігації з використанням react-router-dom та захищених маршрутів. Взаємодія з серверним API реалізована через конфігурований екземпляр Axios, оснащений перехоплювачами для управління токенами автентифікації, передачі параметрів локалізації та централізованої обробки відповідей і помилок. Особливу увагу було приділено покращенню користувацького досвіду, зокрема, шляхом впровадження функціоналу перетягування для впорядкування варіантів відповідей, та структуруванню клієнтських API-сервісів для забезпечення модульності. Спроектовані варіанти використання системи візуалізовано за допомогою діаграм, що демонструють взаємодію користувачів з основними функціями. Загалом, представлені проектні рішення формують цілісну та обґрунтовану архітектуру, готову до подальшої реалізації програмного продукту.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ОНЛАЙН-ГОЛОСУВАННЯ

3.1 Порядок встановлення та налаштування параметрів системи

Розгортання та конфігурація розроблюваного застосунку для системи онлайн-голосування є багатоетапним процесом, що охоплює послідовне налаштування декількох ключових взаємопов'язаних компонентів для забезпечення його повноцінної функціональності. Представлена на діаграмі архітектура системи (рис. 3.1) демонструє класичний підхід до побудови сучасних веб-застосунків, заснований на принципі розподілених сервісів. Вона включає сервер баз даних, відповідальний за зберігання інформації, сервер додатків (бек-енд), що реалізує основну бізнес-логіку, та клієнтську частину (фронт-енд), яка виконується безпосередньо у веб-браузері кінцевого користувача. Такий архітектурний поділ сприяє підвищенню гнучкості, масштабованості системи та забезпечує чітке розмежування відповідальностей між її функціональними рівнями.

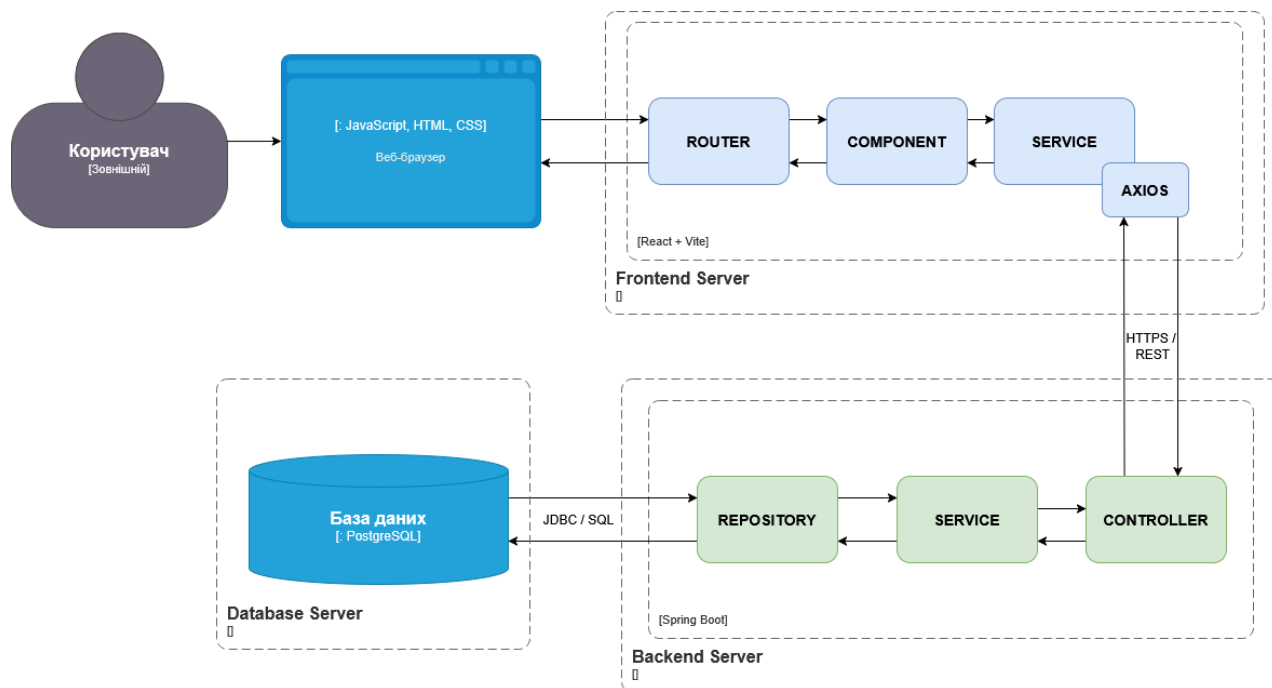


Рисунок 3.1 – Діаграма розгортання

Першочерговим етапом у процесі розгортання є підготовка сервера баз даних, на якому базуватиметься вся система зберігання даних застосунку. У

якості такої системи обрано PostgreSQL, як зазначено на діаграмі. Цей етап включає встановлення самої системи управління базами даних, виконання її базової конфігурації, створення виділеної бази даних для потреб системи онлайн-голосування, а також налаштування облікового запису користувача з відповідними правами доступу. Надалі, параметри підключення до цієї бази даних, такі як адреса сервера, назва бази, логін та пароль користувача, будуть використані для конфігурації серверної частини застосунку.

Наступним кроком є встановлення та конфігурація серверної частини застосунку, розробленої на технологічній платформі Spring Boot. Для цього необхідно забезпечити наявність встановленого середовища виконання Java (JDK) та відповідних інструментів для збирання проекту, наприклад, Maven. Після успішного збирання, бек-енд розгортається як окремий серверний процес. Ключовим моментом конфігурації цього компонента є вказання параметрів підключення до попередньо налаштованого сервера баз даних PostgreSQL, що здійснюється через JDBC/SQL. Окрім цього, налаштовуються параметри безпеки, мережеві порти для обробки запитів від клієнтської частини, рівні логування та інші специфічні властивості, що зазвичай визначаються у файлах конфігурації або через змінні середовища. Архітектурно, бек-енд, згідно з діаграмою, включає рівень репозиторіїв (Repository) для прямої взаємодії з базою даних, сервісний рівень (Service) для інкапсуляції бізнес-логіки та рівень контролерів (Controller) для обробки вхідних HTTP-запитів.

Після успішного налаштування та запуску серверної частини, приступають до розгортання клієнтського застосунку, створеного з використанням бібліотеки React та інструментарію Vite. Цей процес вимагає наявності середовища Node.js та відповідного менеджера пакетів (npm або yarn) для завантаження та встановлення всіх необхідних залежностей проекту. Конфігурація фронт-енду передбачає, перш за все, вказання URL-адреси розгорнутого бек-енд сервера, до якого клієнтська частина буде надсилати API-запити за допомогою бібліотеки Axios. Фронт-енд, що виконується у веб-браузері користувача, структурується за допомогою системи маршрутизації (Router) для навігації між сторінками, набору

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		56

візуальних компонентів (Component) та клієнтських сервісів (Service) для управління станом та комунікації з сервером, як це відображено на схемі.

Завершальним етапом налаштування є перевірка коректної взаємодії між усіма компонентами системи, починаючи від користувача, який взаємодіє з веб-браузером, що відображає контент на основі JavaScript, HTML та CSS. Клієнтська частина, у свою чергу, здійснює обмін даними з бек-енд сервером через захищене HTTPS-з'єднання за допомогою REST API. Конфігурація Axios на фронт-енді та контролерів на бек-енді забезпечує належну обробку цих запитів та відповідей. Для коректної роботи у розподілених середовищах може також знадобитися налаштування політик CORS на серверній стороні. Успішне завершення всіх цих кроків встановлення та налаштування дозволяє забезпечити стабільну та ефективну роботу розробленої системи онлайн-голосування.

3.2 Структура інтерфейсу користувача

Для наочної демонстрації функціональних можливостей та користувацького інтерфейсу розробленої системи онлайн-голосування, нижче наведено детальний опис ключових сторінок застосунку, проілюстрованих відповідними рисунками. Кожна сторінка спроектована з урахуванням принципів інтуїтивності та зручності взаємодії, забезпечуючи ефективне виконання користувачем поставлених завдань.

На рис. 3.2 представлена сторінка авторизації користувача, яка є першочерговим етапом для доступу до захищених функцій системи онлайн-голосування. Дана сторінка реалізує стандартний механізм автентифікації, вимагаючи від користувача введення своїх облікових даних, а саме логіна та пароля, у відповідні поля форми. Візуальне оформлення сторінки є лаконічним та функціональним, зосереджуючи увагу користувача на процесі входу. У випадку успішного підтвердження введених даних, система перенаправляє користувача на відповідну сторінку з розширеними можливостями, найчастіше на головний дашборд або сторінку зі списком доступних опитувань, тим самим відкриваючи доступ до повного спектру функціоналу, передбаченого його

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		57

роллю. Цей етап є критично важливим для забезпечення безпеки системи та персоніфікації взаємодії з нею.

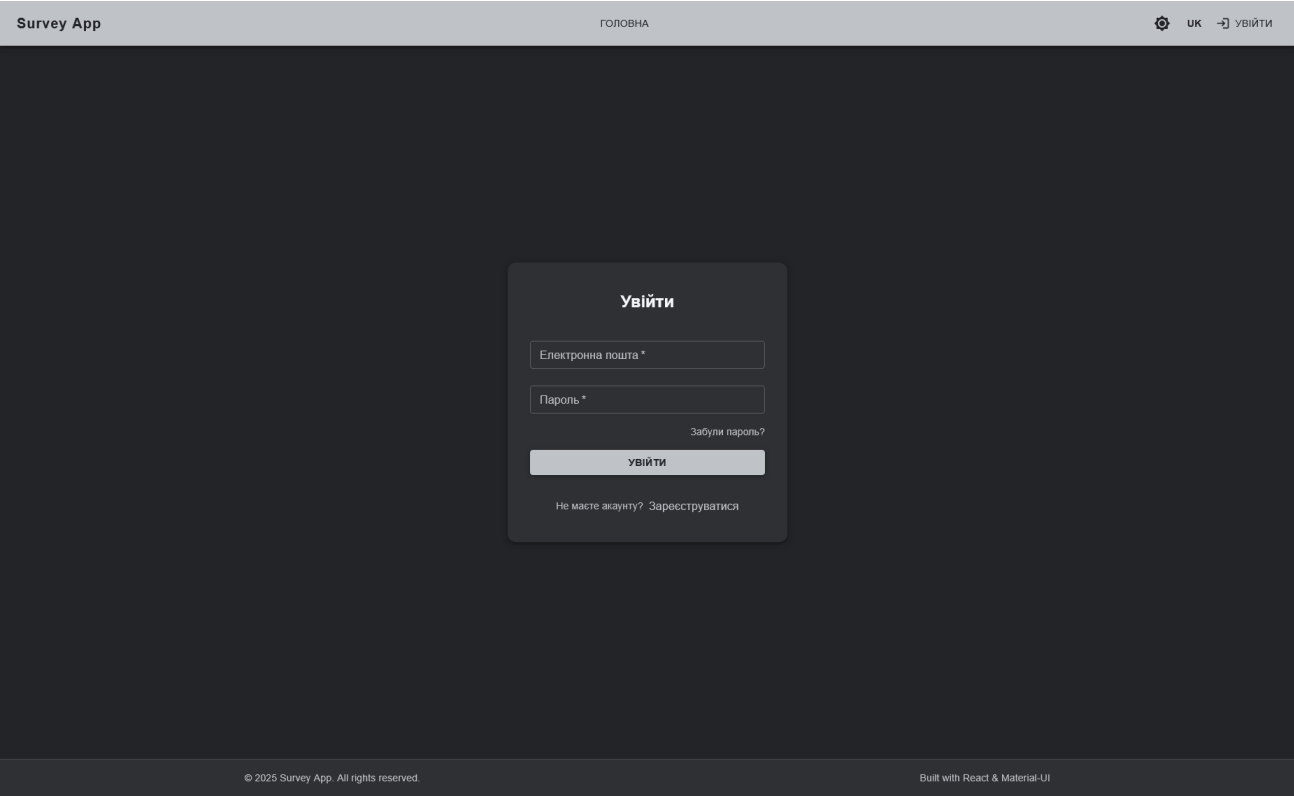


Рисунок 3.2 – Сторінка авторизації користувача

Рис. 3.3 наочно ілюструє ключову сторінку системи, що відображає вичерпний перелік усіх опитувань, доступних конкретному користувачеві в рамках його прав та привілеїв. Ця сторінка була спроектована таким чином, щоб слугувати центральним інформаційним та навігаційним вузлом, що забезпечує швидкий та зручний доступ до взаємодії з різноманітними голосуваннями, які проводяться у системі. На цій сторінці користувачеві надається можливість побачити як ті опитування, що на даний момент знаходяться в активній фазі збору голосів, так і ті, термін проведення яких вже завершився, і для яких, відповідно, доступний перегляд детальної агрегованої статистики та аналітичних даних щодо результатів волевиявлення. Для забезпечення ефективної роботи користувача з потенційно великою кількістю опитувань та для полегшення процесу пошуку необхідної інформації, на сторінці було ретельно реалізовано багатогранний функціонал фільтрації. Цей механізм дозволяє користувачеві гнучко відсіювати та сортувати опитування за різноманітними критеріями,

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

такими як поточний статус опитування (наприклад, «активне», «завершене», «чернетка»), дата його створення, автор опитування, або ж за іншими специфічними атрибутами, що можуть бути релевантними для конкретного користувацького сценарію. Таким чином, завдяки продуманій організації інформації та наявності інструментів фільтрації, користувач отримує можливість ефективно знаходити та швидко обирати саме ті опитування, які йому необхідні для безпосередньої участі, глибокого аналізу результатів або для виконання адміністративних дій, пов'язаних з управлінням власними чи довіреними йому голосуваннями.

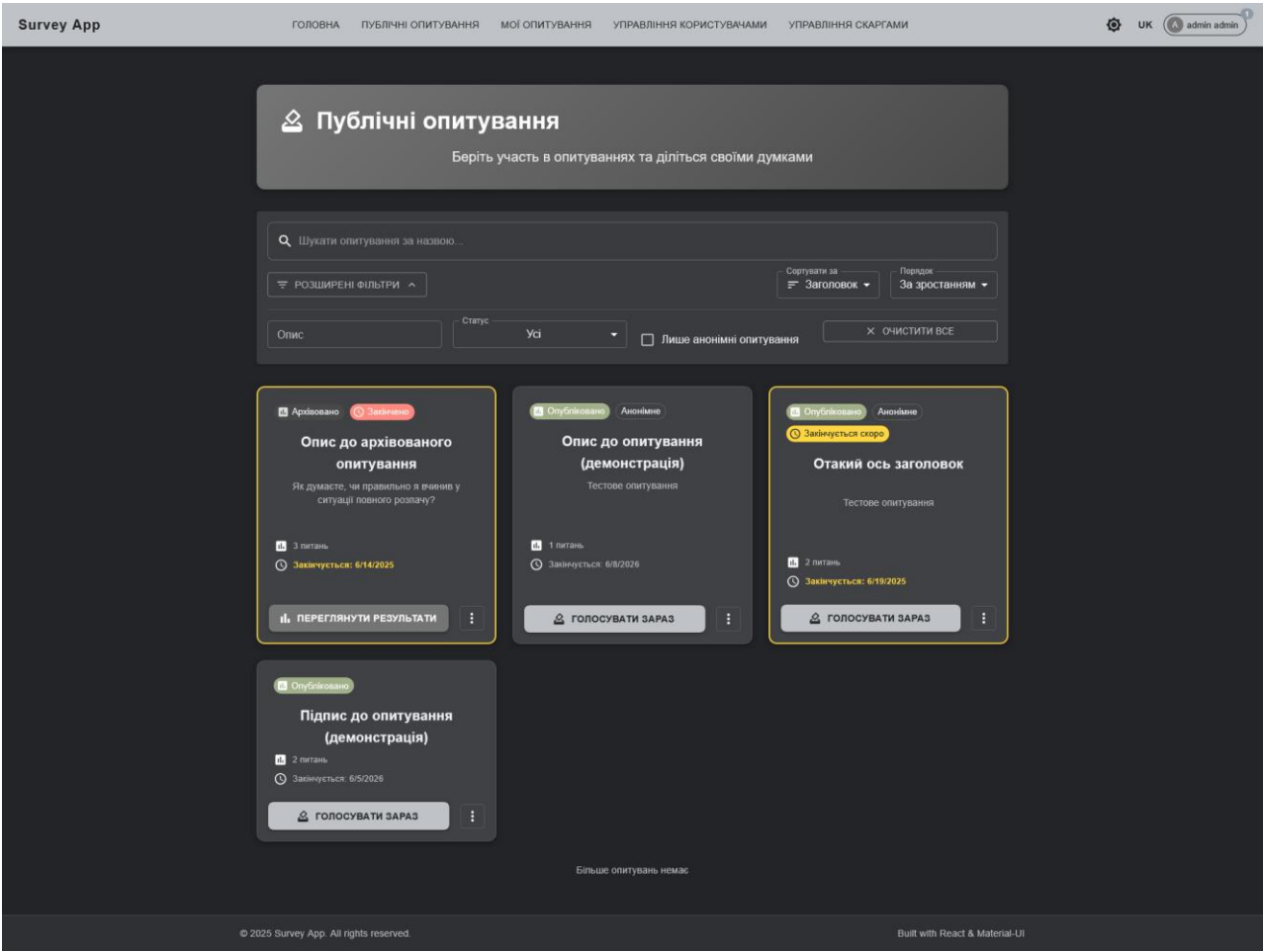


Рисунок 3.3 – Сторінка з усіма опитуваннями

Сторінка створення та управління власними опитуваннями, зображена на рис. 3.4, має чітко структурований двосекційний інтерфейс. Ліва частина сторінки призначена для відображення списку всіх голосувань, автором яких є поточний користувач. Це надає зручний огляд створених ним опитувань та

швидкий доступ до них. Важливою функцією є можливість редагування тих голосувань, які ще не були опубліковані, що дозволяє користувачеві вносити необхідні корективи перед їх запуском. Окрім того, для забезпечення гнучкості обміну даними та можливості архівування чи перенесення опитувань, на сторінці передбачені елементи для імпорту та експорту голосувань у стандартизованому форматі JSON, що значно розширює можливості роботи з даними поза межами системи.

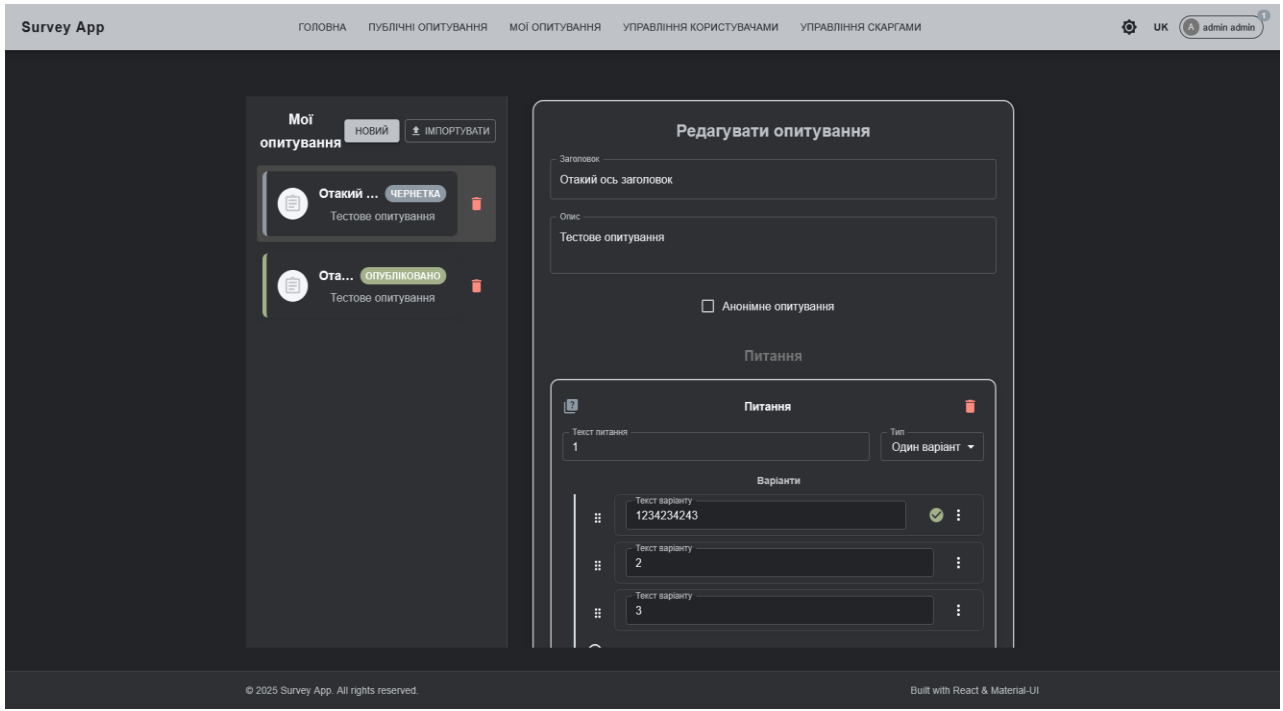


Рисунок 3.4 – Сторінка створення опитувань

Процес безпосередньої участі в опитуванні представлений на сторінці, ілюстрованій рис. 3.5. Для того, щоб залишити свої відповіді, користувач повинен перейти на сторінку конкретного опитування. Інтерфейс цієї сторінки надає користувачеві всю необхідну інформацію для комфортного проходження голосування, включаючи загальну кількість питань та динамічний підрахунок тих питань, на які вже було надано відповідь. Такий підхід дозволяє користувачеві відстежувати свій прогрес. Після надання відповідей на всі поставлені питання, система пропонує користувачеві завершити опитування та, зазвичай, надає можливість перейти на сторінку з переглядом агрегованої

статистики або результатів даного голосування, якщо це передбачено його налаштуваннями.

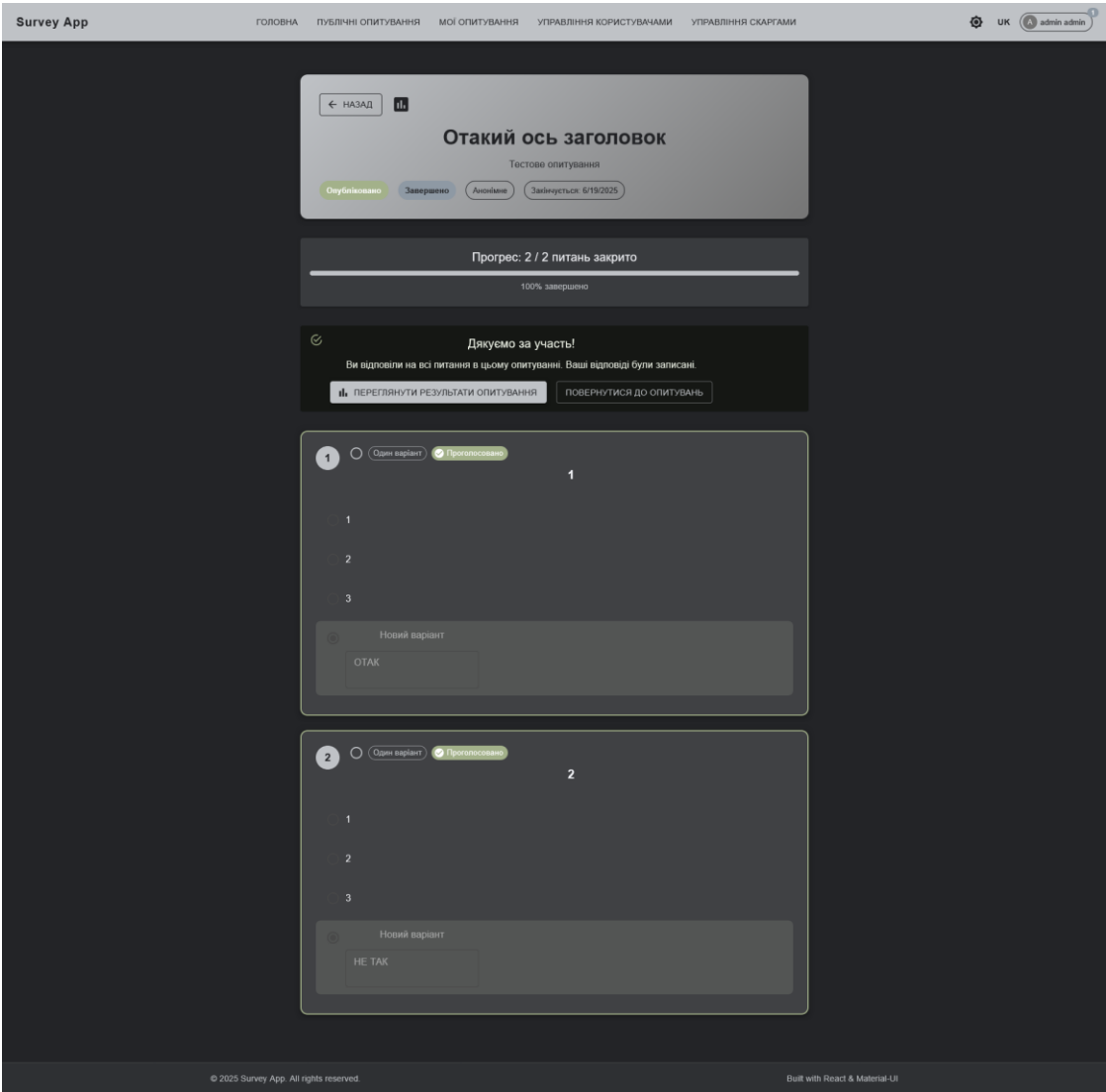


Рисунок 3.5 – Сторінка проходження опитування

На рис. 3.6 зображено сторінку перегляду результатів та статистики опитування, яка надає користувачам детальну інформацію щодо підсумків проведеного голосування. На цій сторінці візуалізуються ключові метрики, такі як загальна кількість користувачів, що взяли участь в опитуванні, сукупна кількість голосів, врахованих системою, а також відсоток завершення опитування. Останній показник відображає частку учасників, які пройшли опитування до кінця та успішно надіслали свої результати. Для глибокого аналізу, користувач має можливість переглядати розподіл голосів за кожним варіантом відповіді, представлений у відсотковому співвідношенні. З метою

підвищення наочності та полегшення сприйняття даних, на сторінці активно використовуються графічні елементи, зокрема кругові та стовпчасті діаграми. Додатково, залежно від налаштувань анонімності опитування, може бути доступна інформація про те, хто саме з учасників проголосував за той чи інший варіант відповіді.

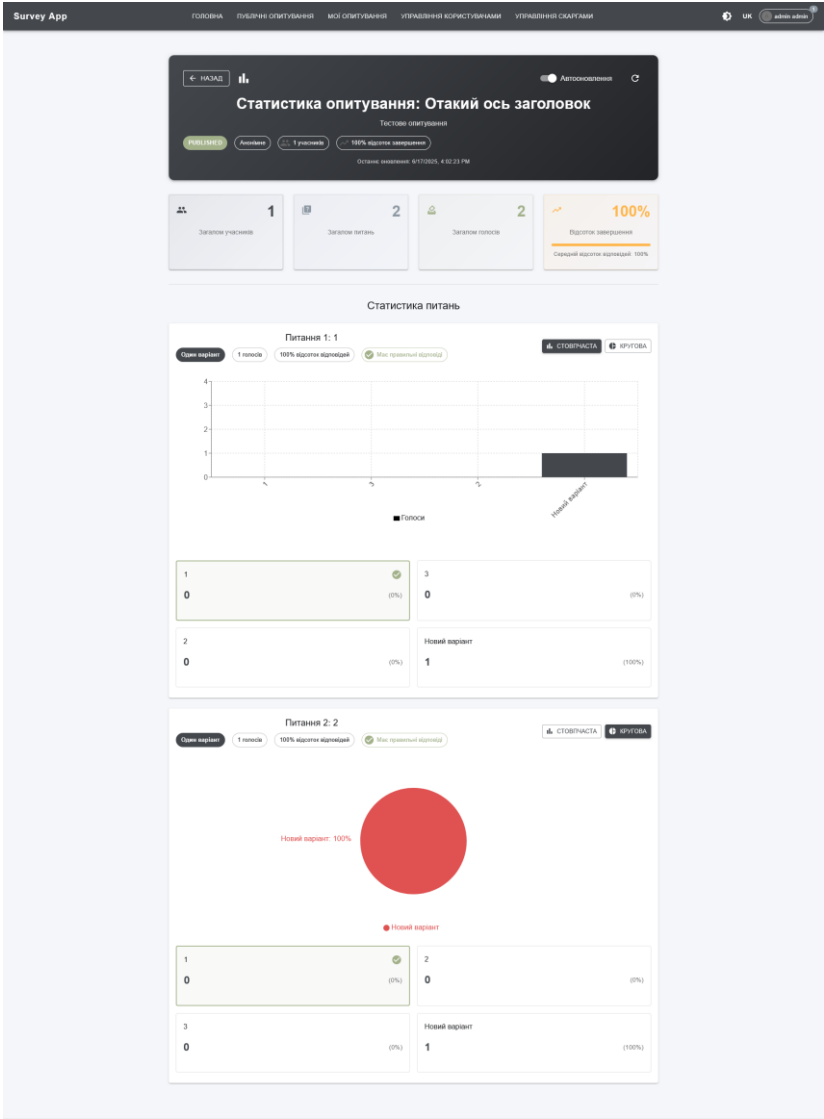


Рисунок 3.6 – Сторінка перегляду результатів

Головна сторінка застосунку, проілюстрована на рис. 3.7, виконує роль інформаційного центру та навігаційного хабу. Вона надає користувачеві агреговану статистичну інформацію щодо активності в системі онлайн-голосування в цілому. Окрім загальних показників, на головній сторінці зазвичай відображаються списки найпопулярніших опитувань, що визначаються за

кількістю учасників або голосів, а також перелік найновіших голосувань, нещодавно доданих до системи. Такий підхід дозволяє користувачам швидко ознайомитися з актуальними та найбільш обговорюваними опитуваннями, сприяючи їх залученню до процесу голосування та забезпечуючи зручну точку входу для подальшої взаємодії з функціоналом системи.

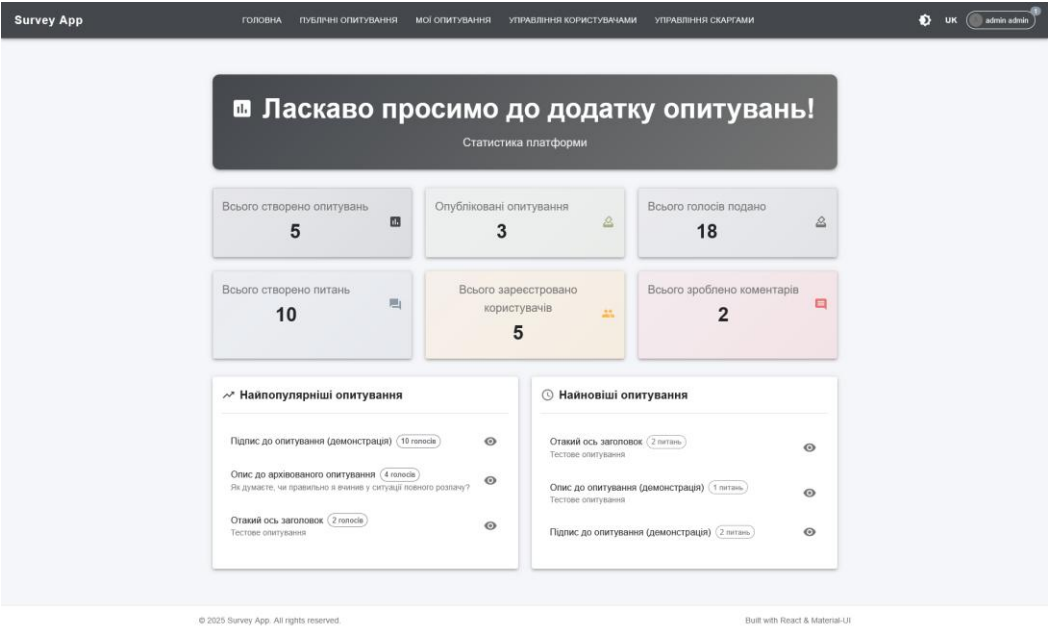


Рисунок 3.7 – Головна сторінка

3.3 Порядок роботи користувача з системою

Послідовність дій користувача при взаємодії з розробленою системою онлайн-голосування є логічною та інтуїтивно зрозумілою, починаючи від створення облікового запису і завершуючи активною участю у голосуваннях та їх адмініструванні, залежно від наданих прав. Нижче детально описано основні етапи роботи користувача з системою, що проілюстровані відповідними рисунками.

Для того, щоб розпочати роботу з системою, новий користувач повинен спочатку створити особистий обліковий запис. Процес реєстрації ініціюється через спеціальну форму на рис. 3.8. У цій формі користувачеві необхідно заповнити обов'язкові поля, такі як ім'я, прізвище, адреса електронної пошти, яка буде використовуватися як логін, та пароль для доступу до системи. Додатково, передбачені необов'язкові поля для введення номера телефону та адреси

проживання, що можуть використовуватися для розширеної ідентифікації або комунікації в майбутньому. Заповнення цієї форми є першим кроком до отримання доступу до функціоналу системи.

Рисунок 3.8 – Форма для реєстрації користувача

Після успішного завершення процедури реєстрації, система автоматично перенаправляє користувача на сторінку його особистого профілю на рис. 3.9. На цій сторінці користувач отримує можливість переглядати та редагувати раніше введену інформацію про свій акаунт, а також відстежувати певну статистику, пов'язану з його активністю в системі на рис. 3.10. Важливим аспектом є механізм верифікації електронної пошти. Для користувачів, які ще не підтвердили свою поштову скриньку, на сторінці профілю буде відображатися відповідне повідомлення з нагадуванням про необхідність виконати цю дію для активації повного функціоналу облікового запису та підвищення рівня його безпеки.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		64

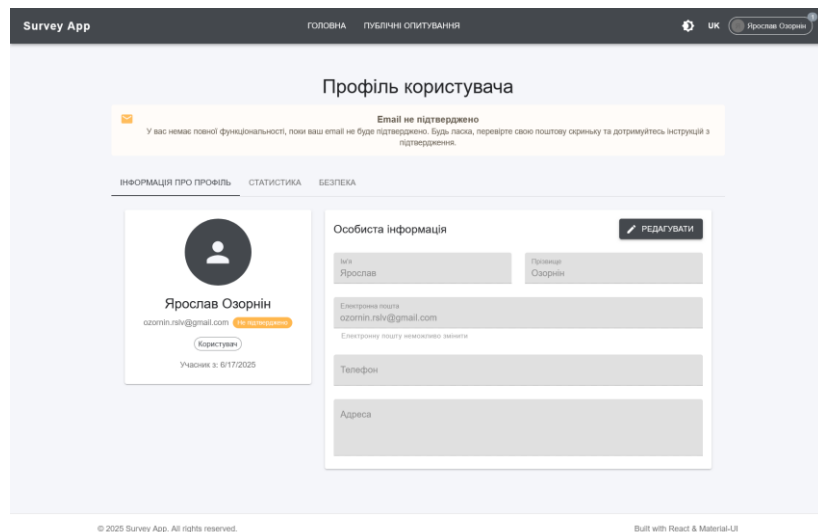


Рисунок 3.9 – Сторінка користувача

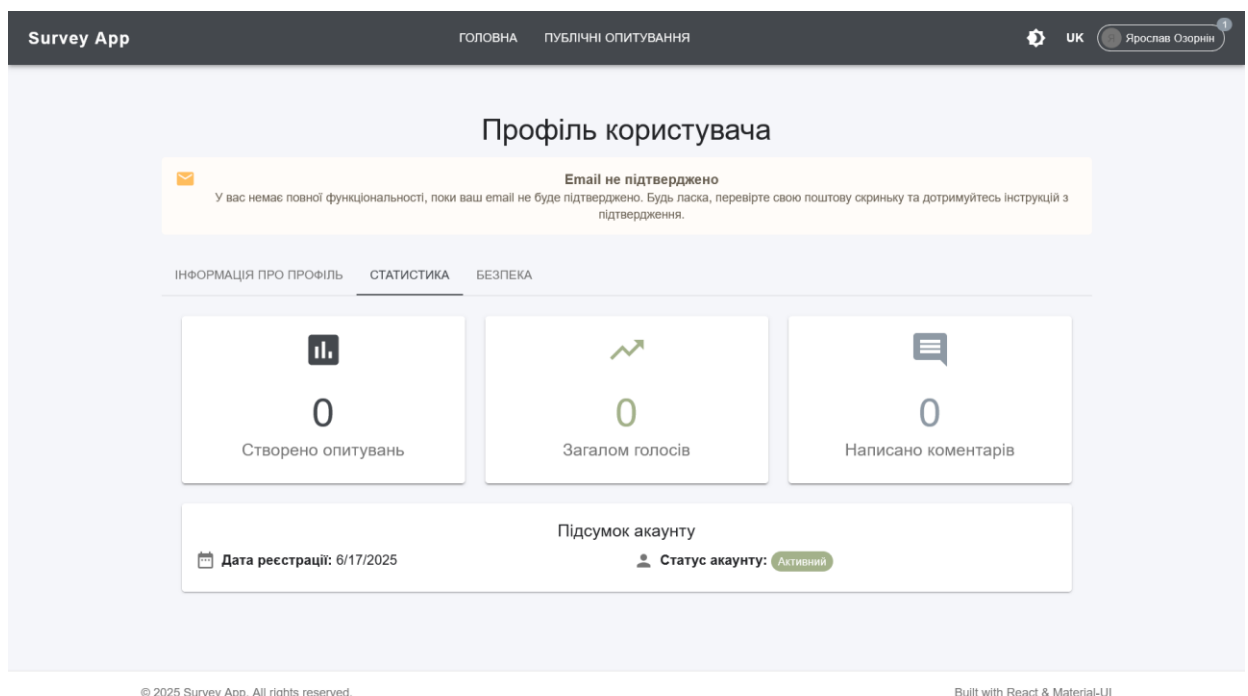


Рисунок 3.10 – Статистика користувача

Процес верифікації електронної пошти передбачає надсилання системою автоматичного повідомлення на вказану при реєстрації адресу. Цей лист містить унікальне посилання, перехід за яким веде на сторінку підтвердження. Якщо наданий у посиланні токен верифікації є дійсним та коректним, система підтверджує електронну пошту, і користувач бачить повідомлення про успішне завершення процедури на рис. 3.11. Успішна верифікація електронної пошти є важливою умовою для виконання багатьох ключових дій у системі.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		
						65

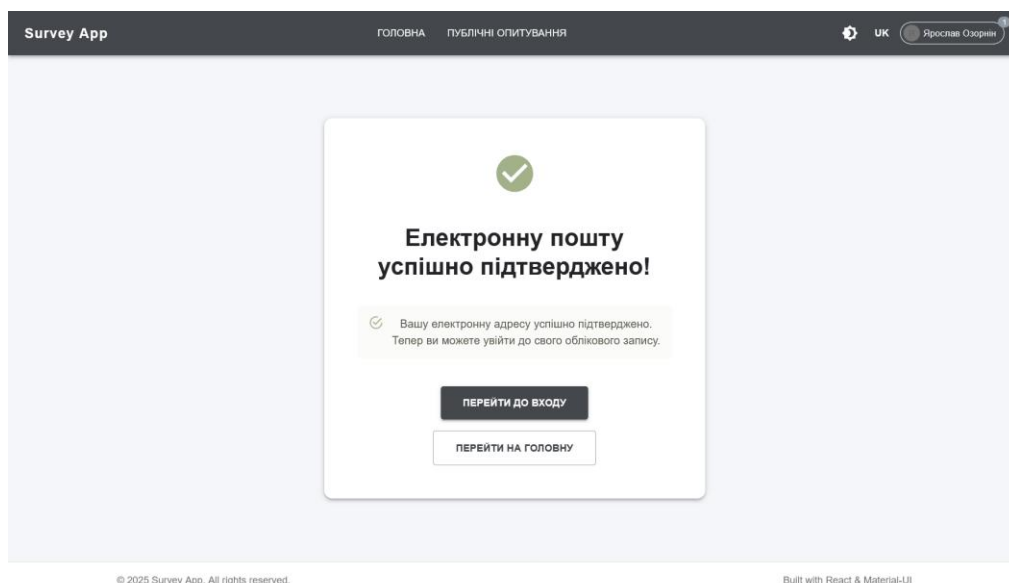


Рисунок 3.11 – Успішно верифікована електронна пошта

Функціональні можливості, доступні користувачеві, визначаються його роллю в системі. За замовчуванням, новостворений та верифікований користувач не має прав модератора або адміністратора, тому його взаємодія з системою обмежується функціоналом, представленим у попередніх пунктах на рис. 3.2 – 3.7, тобто переглядом та участю у голосуваннях, а також створенням власних опитувань. Ключовою можливістю для звичайного користувача є участь у доступних голосуваннях. Кожен голос, відданий користувачем, зберігається централізовано в базі даних на сервері. Такий підхід забезпечує цілісність даних та покращує загальний досвід користування, оскільки система не залежить від локальних даних на клієнтському пристрої. Це дозволяє користувачам продовжувати або завершувати проходження опитувань на різних пристроях або з різних веб-браузерів, зберігаючи свій прогрес. Якщо користувач надав відповіді на всі питання опитування та підтверджує своє бажання його завершити, як показано на діалоговому вікні на рис. 3.12, його вибір фіксується, і подальше редагування відповідей для даного опитування стає неможливим, а саме голосування позначається як пройдене для цього користувача. Після успішного завершення опитування, користувач отримує можливість перейти на сторінку з його результатами для перегляду агрегованої статистики на рис. 3.13.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		
						66

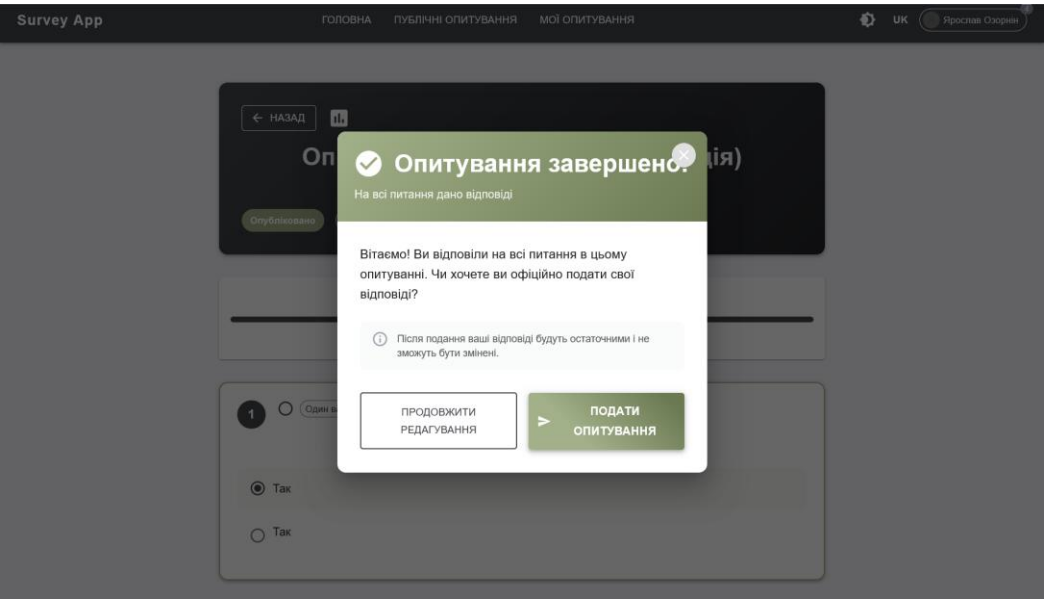


Рисунок 3.12 – Пропозиція закінчити опитування, або продовжити редагувати.

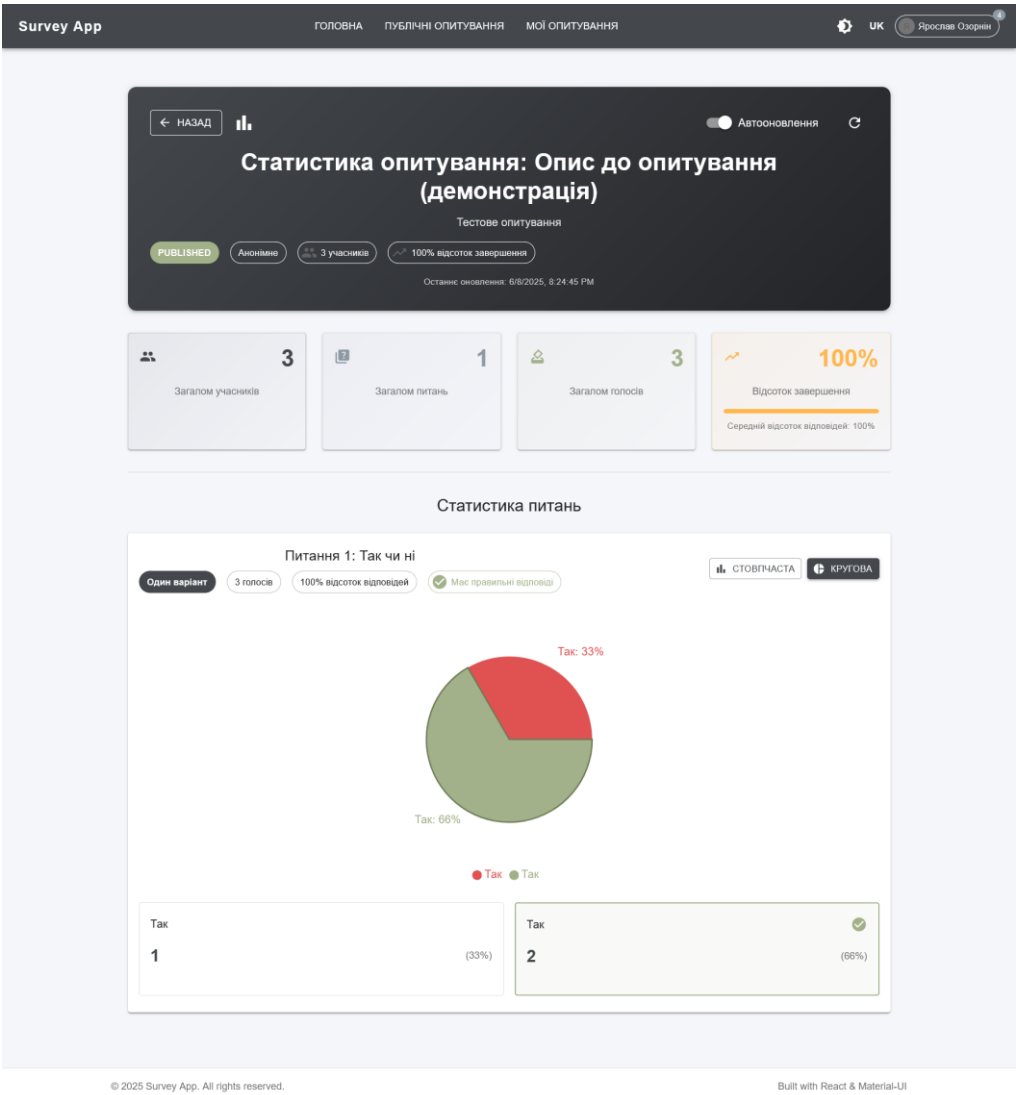


Рисунок 3.13 – Сторінка з результатами опитування

Якщо користувач бажає створити власне опитування, він повинен перейти на відповідну сторінку управління власними опитуваннями, звідки надається ця функціональність. Система пропонує два основні шляхи для створення нового голосування: імпорт вже існуючого опитування, попередньо збереженого у форматі JSON, або створення нового опитування вручну. У випадку ручного створення, користувачеві необхідно заповнити початкові дані для збереження базової інформації в системі. Обов'язковим для заповнення є назва опитування, тоді як поле опису є опціональним на рис. 3.14. Після введення цих даних та збереження, створюється новий запис про опитування на рис. 3.15.

Рисунок 3.14 – Діалог створення опитування

Рисунок 3.15 – Створене опитування

Наступним кроком після створення каркасу опитування є додавання до нього питань та відповідних варіантів відповідей на рис. 3.16. Для покращення взаємодії користувача з інтерфейсом та спрощення процесу структурування опитування, реалізовано механізм перетягування блоків питань та варіантів відповідей в межах їхніх батьківських об'єктів. Це дозволяє легко та інтуїтивно змінювати послідовність як самих питань в опитуванні, так і варіантів відповідей у межах конкретного питання. Кожне питання може бути одного з двох основних типів: з можливістю вибору лише однієї правильної відповіді або з можливістю вибору кількох правильних відповідей. Система контролює логіку вибору: якщо питання передбачає одну правильну відповідь, користувач-творець не зможе позначити два або більше варіанти як правильні. Відповідно, якщо питання допускає кілька правильних відповідей, така можливість надається. Додатково, для кожного питання передбачена опція додавання відкритої відповіді, що дозволяє респондентам вводити власний текстовий варіант, якщо запропоновані варіанти їх не влаштовують. Важливо зазначити, що варіант з відкритою відповіддю може бути лише один на кожне окреме питання.

Після того, як опитування було налаштоване, з усіма необхідними питаннями, продуманими варіантами відповідей та визначеними параметрами, користувач-автор переходить до ключового етапу – його публікації. Ця дія переводить опитування зі статусу "чернетка" в активний стан, роблячи його видимим та доступним для проходження всіма користувачами системи, які мають відповідні права доступу. Ці права можуть визначатися як загальними ролями користувачів у системі, так і специфічними налаштуваннями видимості, встановленими автором для конкретного опитування, що дозволяє гнучко керувати цільовою аудиторією. Важливо зазначити, що при цьому сам автор опитування, незважаючи на свою роль творця, також зберігає можливість взяти у ньому участь на загальних підставах, як і будь-який інший користувач. Це може бути корисним не тільки для фінального тестування коректності роботи всіх механізмів з точки зору респондента, але й для того, щоб автор міг висловити власну думку чи позицію з порушених в опитуванні питань.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		69

Редагувати опитування

Заголовок

Моє перше опитування

Опис

Перевіряю як працює моє перше опитування

☐ Анонімне опитування

Питання

Питання

Текст питання

Перше тестове питання

Тип

Один варіант

Варіанти

Текст варіанту

Перший варіант відповіді

Текст варіанту

Другий варіант відповіді

Відкрите питання

Відкрите питання

+

Додати варіант

ДОДАТИ ПИТАННЯ

ОПУБЛІКУВАТИ ОПИТУВАННЯ

ЕКСПОРТУВАТИ ОПИТУВАННЯ

Рисунок 3.16 – Редагування опитування

Опублікувати опитування

Ви збираєтеся опублікувати: **Отакий ось заголовок**

Дата закінчення опитування (необов'язково)

19.06.2025 00:00

📅

Залиште порожнім для безтермінового опитування

СКАСУВАТИ

ОПУБЛІКУВАТИ

Рисунок 3.17 – Вибір дати закінчення опитування при публікації

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		70

Окрім функціоналу, доступного звичайним користувачам, система онлайн-голосування передбачає розширені можливості адміністрування, які надаються користувачам з ролями адміністратора та модератора. Для подальшого опису цих функцій буде розглядатися взаємодія з системою з облікового запису адміністратора, який володіє найповнішим набором прав. Однією з ключових адміністративних функцій є управління користувачами системи, для чого призначена спеціальна сторінка на рис. 3.18. Ця сторінка надає адміністратору інструменти для ефективного пошуку та фільтрації користувачів за різними критеріями, відображає коротку статистику по кожному користувачеві, а також показує поточний статус його акаунта, зокрема, чи є він активним або заблокованим.

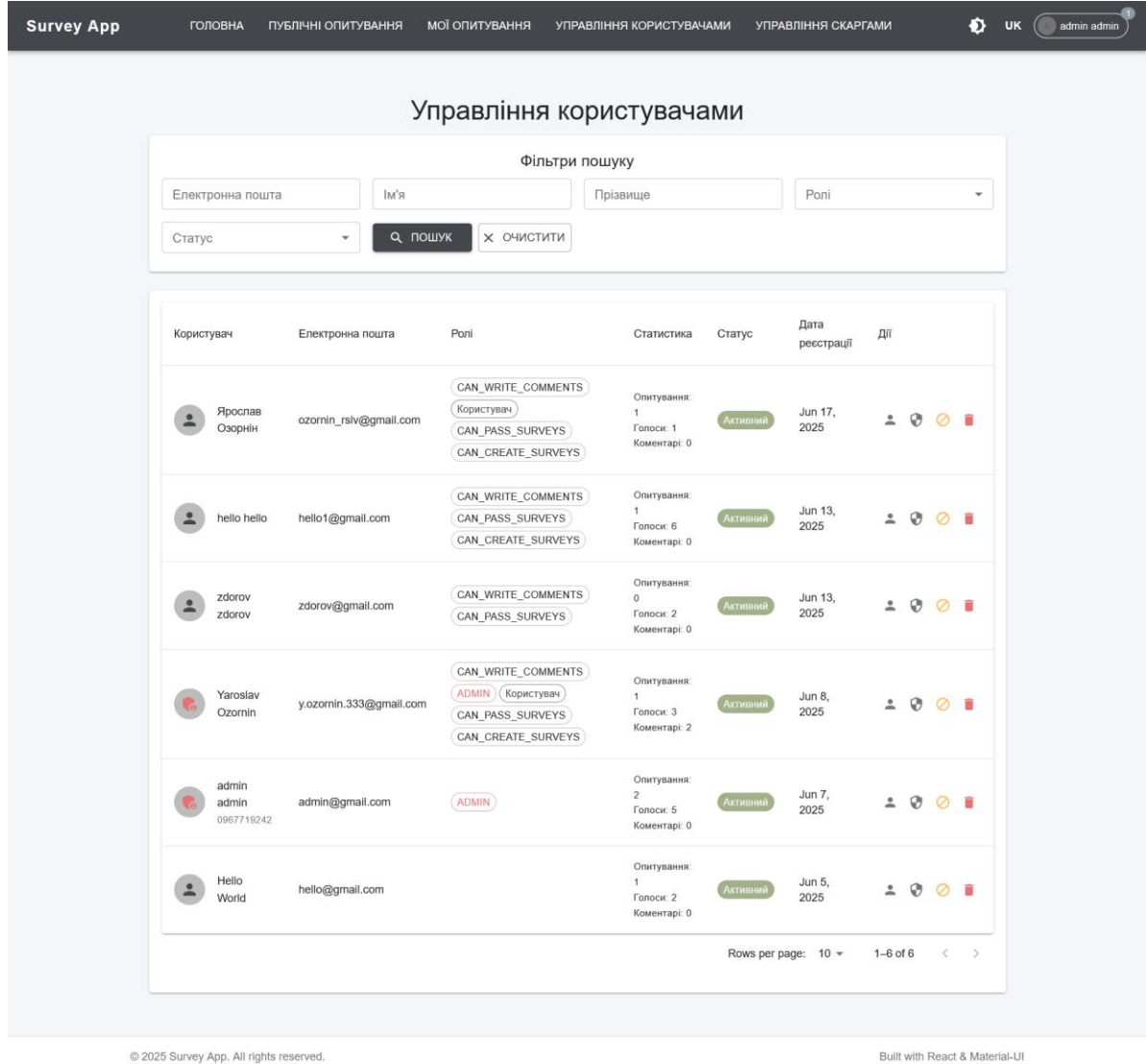


Рисунок 3.18 – Сторінка керування акаунтами користувачів

Для отримання більш детальної інформації про конкретного користувача, адміністратор може перейти на сторінку повної статистики акаунта, натиснувши відповідну кнопку. Як показано на рис. 3.19, тут доступні загальні дані, які користувач вводив під час реєстрації, перелік призначених йому ролей, агрегована статистика його активності в опитуваннях, а також дата реєстрації. Окрема вкладка «Опитування» на рис. 3.20 дозволяє переглянути список усіх опитувань, створених та опублікованих даним користувачем, що полегшує моніторинг контенту в системі.

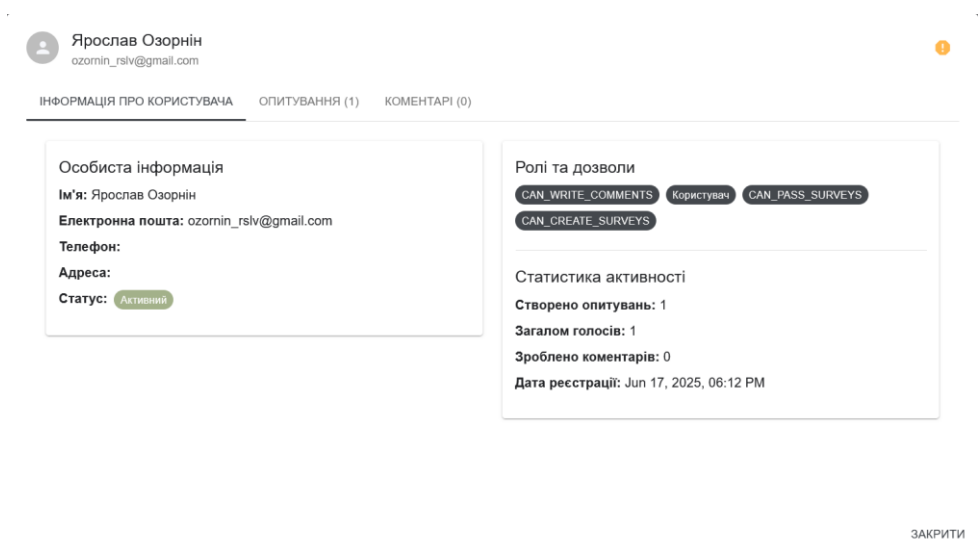


Рисунок 3.19 – Статистика користувача

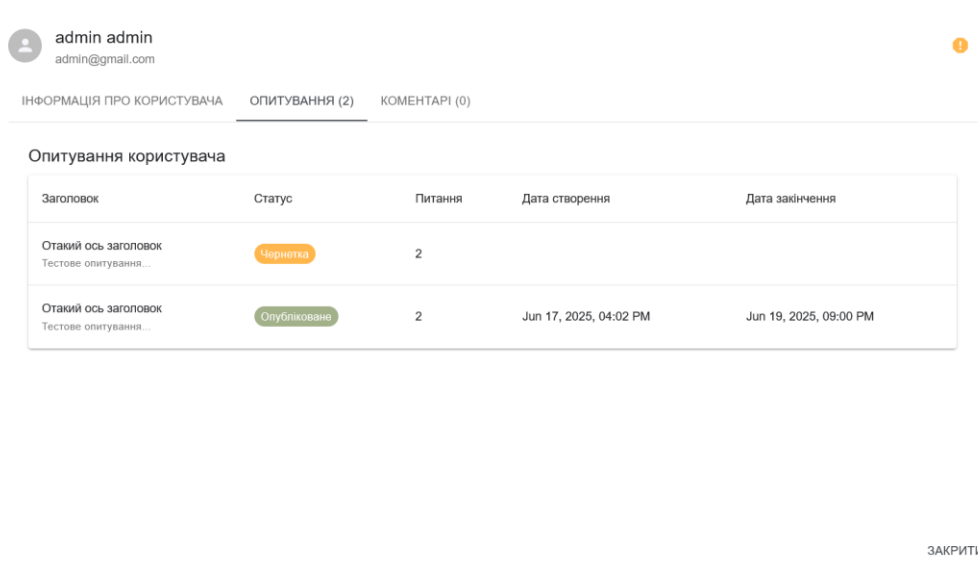


Рисунок 3.20 – Опитування користувача

Важливою можливістю адміністратора є управління ролями користувачів. Через спеціальний редактор ролей на рис. 3.21 адміністратор може динамічно видавати або відкликати певні ролі у будь-якого користувача системи. Призначення або позбавлення певних ролей безпосередньо впливає на доступ користувача до окремих функціональних можливостей системи, забезпечуючи гнучкий контроль над правами та дозволами.

Електронна пошта | Ім'я | Прізвище | Ролі

Статус

Користувач

Керування ролями для Ярослав Озорнін
ozornin_rslv@gmail.com

Керування ролями та дозволами користувача. Зміни набувають чинності негайно.

Поточні ролі

CAN_WRITE_COMMENTS	Може писати коментарі до опитувань	
Користувач	Базові привілеї користувача	
CAN_PASS_SURVEYS	Може брати участь в опитуваннях	
CAN_CREATE_SURVEYS	Може створювати та керувати опитуваннями	

Додати нову роль

Оберіть роль

- ADMIN
Повний доступ до системи та управління користувачами
- MODERATOR
Модерація контенту та управління опитуваннями

+ ДОДАТИ РОЛЬ

ЗАКРИТИ

Рисунок 3.21 – Редактор ролей

Адміністратори також мають повноваження блокувати користувачів, які порушують правила системи. Для здійснення блокування необхідно вказати тривалість блокування у днях та надати чітку причину такого рішення на рис. 3.22. Ці поля є обов'язковими для заповнення, оскільки система автоматично надсилає заблокованому користувачеві повідомлення на його електронну пошту, інформуючи про факт блокування, його причину та дату закінчення дії обмежень. Найбільш радикальною дією, доступною виключно адміністраторам

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		73

(але не модераторам), є повне видалення даних про користувача. Ця операція призводить до безповоротного очищення всієї інформації, пов'язаної з акаунтом користувача, з бази даних системи.

Заблокувати користувача

 Ви збираєтесь заблокувати цього користувача. Він втратить доступ до свого акаунту.

Користувач: Ярослав Озорнін (ozornin_rslv@gmail.com)

Тривалість блокування

☒ Тимчасове блокування

☐ Постійне блокування

Тривалість

1 тиждень

Причина *

Введіть причину блокування цього користувача...

Причина обов'язкова

СКАСУВАТИ

ЗАБЛОКУВАТИ КОРИСТУВАЧА

Рисунок 3.22 – Блокування користувача

Ще однією важливою сторінкою, доступною адміністраторам та модераторам, є сторінка управління скаргами на рис. 3.23. Система передбачає можливість для користувачів надсилати скарги на різні типи сутностей у системі, зокрема на опитування, коментарі або інших користувачів. На сторінці управління скаргами, вони агрегуються за сутностями, на які було подано скаргу. Адміністратор може вибрати конкретну сутність, щоб переглянути детальний список скарг, залишених користувачами, включаючи їхні описи та причини на рис. 3.24.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		74

Процес обробки скарги вимагає від адміністратора або модератора прийняття рішення на рис. 3.25. Необхідно вибрати один із запропонованих системою вжитих заходів (наприклад, видалення контенту, попередження користувача, блокування), встановити остаточний статус скарги (наприклад, «вирішено», «відхилено») та написати відповідь користувачеві, який подав скаргу. Заповнення цих полів є обов'язковим, оскільки ця інформація використовується не лише для внутрішнього обліку та модерації, але й для автоматичного формування та надсилання листа на електронну пошту скажника. У цьому листі користувач інформується про те, що його скарга була розглянута, та про конкретні дії, які були вжиті за її результатами. Такий механізм забезпечує прозорість процесу модерації та підтримує зворотний зв'язок з користувачами.

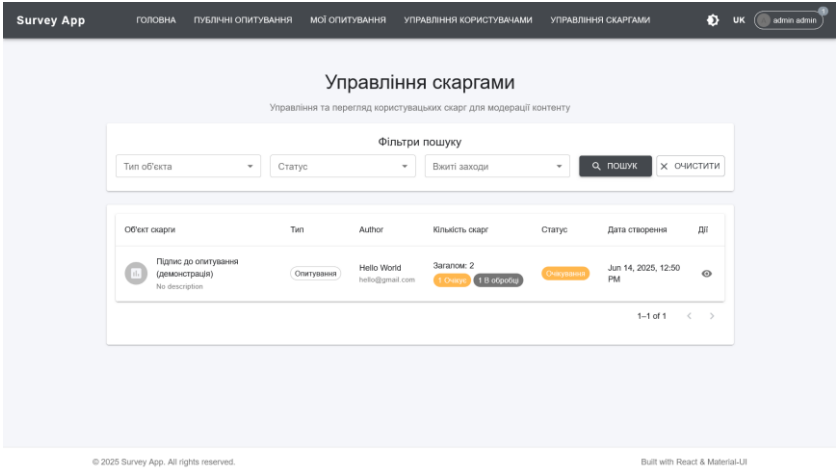


Рисунок 3.23 – Сторінка скарг

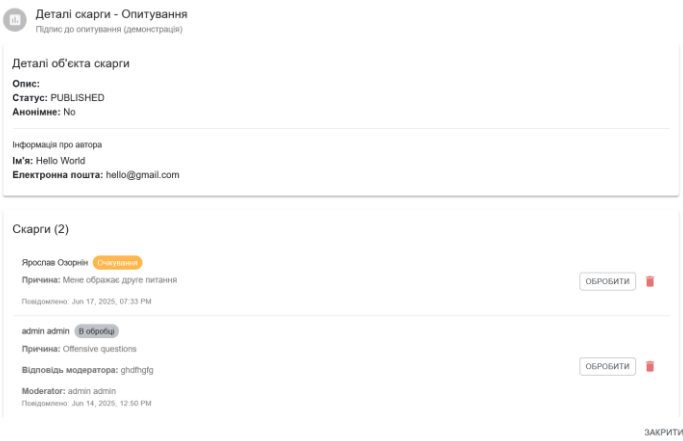


Рисунок 3.24 – Перегляд скарги на конкретну сутність

Обробити скаргу

Обробка скарги від Ярослав Озорнін

Статус

Вирішено

Вжиті заходи

Контент видалено

Повідомлення у відповідь

Опитування видалено

ОНОВИТИ СКАРГУ

СКАСУВАТИ

ЗАКРИТИ

Рисунок 3.25 – Обробка скарги від користувача

Адміністратор повинен обробити скаргу, вибрати один із перелічених вжитих заходів, вибрати статус скарги та написати відповідь користувачу. Все це необхідно не лише для загального модерування, а ще й для того, щоб відправити користувачу лист на пошту, де буде повідомлено, що його скарга була оброблена та було прийнято конкретні дії.

Висновки до третього розділу

У третьому розділі було детально розглянуто архітектуру та порядок роботи користувача з розробленою системою онлайн-голосування. Наведено опис процесу встановлення та налаштування параметрів застосунку, що включає підготовку сервера баз даних PostgreSQL, розгортання серверної частини на Spring Boot та клієнтської частини на React. Представлена архітектурна діаграма наочно продемонструвала взаємодію між цими ключовими компонентами, підкреслюючи розподілений характер системи та використання сучасних технологій для забезпечення її функціонування, таких як JDBC/SQL для взаємодії з базою даних, REST API для комунікації між клієнтом та сервером, а також бібліотеки Axios для управління HTTP-запитами на стороні клієнта.

Детальний опис ключових сторінок користувацького інтерфейсу, підкріплений відповідними рисунками, дозволив наочно продемонструвати функціональні можливості системи. Розглянуто такі важливі сторінки, як сторінка авторизації, сторінка зі списком усіх опитувань з можливостями

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		76

фільтрації, сторінка створення та управління власними опитуваннями, що включає імпорту/експорту у форматі JSON, а також сторінки безпосереднього проходження опитування та перегляду його детальної статистики з використанням графічних елементів. Кожна сторінка спроектована з урахуванням принципів інтуїтивності та зручності, забезпечуючи ефективну взаємодію користувача з системою.

Окрему увагу було приділено послідовності дій користувача при роботі з системою, починаючи від процесу реєстрації та верифікації електронної пошти, до активної участі у голосуваннях, створення власних опитувань з можливістю перетягування питань та варіантів відповідей, та завершуючи адміністративними функціями. Розглянуто розширений функціонал, доступний користувачам з ролями адміністратора та модератора, зокрема управління акаунтами користувачів, призначення ролей, блокування порушників та обробка скарг. Описані механізми взаємодії та функціональні можливості демонструють прагнення створити гнучку, керовану та зручну для користувачів систему онлайн-голосування, що відповідає сучасним вимогам до подібних програмних продуктів.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		77

ВИСНОВКИ

У даній кваліфікаційній роботі було успішно вирішено завдання розробки системи онлайн-голосування, що відповідає сучасним вимогам до безпеки, функціональності та зручності використання. Проведений на початковому етапі аналіз предметної області та існуючих аналогічних рішень, таких як Google Forms, портал «Дія» та Election Runner, дозволив чітко окреслити актуальні потреби ринку та сформулювати ключові вимоги до проектованої системи. Було встановлено, що, незважаючи на наявність різноманітних інструментів, існує потреба у збалансованому рішенні, яке б поєднувало достатній рівень безпеки та контролю з гнучкістю у створенні та адмініструванні опитувань, не обмежуючись вузькою спеціалізацією або складними державними механізмами верифікації. Виходячи з цього, основними цілями розробки стали забезпечення верифікації користувачів через електронну пошту, реалізація системи контролю доступу на основі ролей, надання інструментів для створення різноманітних типів питань, а також інтуїтивно зрозумілий інтерфейс як для учасників голосувань, так і для їх організаторів. Для досягнення цих цілей було обґрунтовано вибір сучасного технологічного стеку, що включає фреймворк Spring Boot для розробки надійної та масштабованої серверної частини, бібліотеку React для створення динамічного односторінкового клієнтського застосунку, систему управління базами даних PostgreSQL для забезпечення цілісності та ефективного зберігання даних, а також технології JWT для безпечної автентифікації та RBAC для розмежування прав доступу.

На етапі проектування системи було розроблено детальну архітектуру, що відповідає принципам багат шаровості та розподіленої обробки. Серверна частина, реалізована на Spring Boot, побудована за класичною моделлю Controller-Service-Repository, що забезпечує чітке розділення відповідальностей: контролери обробляють HTTP-запити, сервісний шар інкапсулює бізнес-логіку, а репозиторії відповідають за взаємодію з базою даних через Spring Data JPA. Взаємодія між клієнтом та сервером здійснюється за допомогою RESTful API,

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		78

захищеного механізмами автентифікації на основі JWT токенів та авторизації через Spring Security. Для підвищення безпеки та стабільності системи було інтегровано фільтри обмеження частоти запитів (rate-limit) та реалізовано аспектно-орієнтоване логування ключових операцій. Для персистентності даних розроблено детальну реляційну схему бази даних PostgreSQL, яка відображає сутності користувачів, опитувань, питань, варіантів відповідей, голосів, коментарів, а також включає таблиці для реалізації механізмів управління ролями та модерації скарг. Клієнтська частина системи, розроблена на React, є сучасним односторінковим веб-застосунком (SPA), що забезпечує високу швидкість реакції інтерфейсу та безперервність взаємодії. Було реалізовано компонентну архітектуру, використано механізми управління станом за допомогою AuthContext та RBACContext для автентифікації користувачів та розмежування прав доступу. Навігація в додатку реалізована з використанням бібліотеки react-router-dom із застосуванням захищених маршрутів (ProtectedRoute) для обмеження доступу до певних сторінок залежно від ролі та статусу автентифікації користувача. Взаємодія з серверним API здійснюється через конфігурований екземпляр бібліотеки Axios, оснащений перехоплювачами для автоматичного додавання токенів автентифікації, передачі параметрів локалізації та централізованої обробки відповідей і помилок.

У рамках третього розділу було детально описано порядок встановлення та налаштування параметрів застосунку, а також представлено ключові сторінки користувацького інтерфейсу та сценарії взаємодії користувача з системою. Було продемонстровано процес реєстрації нового користувача, включаючи верифікацію електронної пошти, процедуру авторизації, а також основні сторінки для перегляду та участі в опитуваннях, створення та управління власними голосуваннями. Особливу увагу приділено інтуїтивності інтерфейсу, зокрема реалізації функціоналу перетягування (drag-and-drop) для впорядкування питань та варіантів відповідей, що значно покращує користувацький досвід. Також було детально розглянуто розширений функціонал, доступний користувачам з ролями адміністратора та модератора,

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		79

такий як управління акаунтами інших користувачів, призначення ролей, блокування порушників та система обробки скарг на контент або дії користувачів. Описані механізми та візуальні представлення сторінок підтверджують, що розроблений інтерфейс є логічним, послідовним та спрямованим на ефективне виконання користувачами своїх завдань у системі.

Таким чином, розроблена система онлайн-голосування успішно поєднує в собі вимоги до безпеки, гнучкості та зручності використання, пропонуючи функціональне рішення для широкого кола завдань. Використання сучасних технологій та продумана архітектура забезпечують надійну основу для подальшого розвитку та масштабування системи. Реалізовані механізми верифікації, автентифікації, авторизації та адміністрування дозволяють створювати контрольоване середовище для проведення опитувань, мінімізуючи ризики недобросовісних дій та забезпечуючи довіру до отриманих результатів. Система надає користувачам інтуїтивно зрозумілі інструменти як для участі в голосуваннях, так і для їх створення та управління, що робить її доступною для широкого кола користувачів незалежно від їхнього технічного рівня.

Створений програмний продукт є готовим до практичного застосування та має потенціал для подальшого вдосконалення, наприклад, шляхом інтеграції з іншими платформами, розширення аналітичних можливостей, впровадження більш складних механізмів захисту від фальсифікацій або розробки мобільного додатку.

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		80

ДЖЕРЕЛА ІНФОРМАЦІЇ

1. Озорнін Я.В., Локтікова Т.М., Порівняльний аналіз сучасних систем онлайн-голосувань. Тези доповідей VII Всеукраїнської науково-технічної конференції «Комп'ютерні технології: інновації, проблеми, рішення», 02-03 грудня 2024 року. Житомир: «Житомирська політехніка», 2024. С. 49-50
2. Озорнін Я.В., Локтікова Т.М. Реалізація автентифікації та авторизації у вебзастосунку голосування з використанням JWT і Spring Security. Тези доповідей Всеукраїнської науково-практичної on-line конференції здобувачів вищої освіти і молодих учених, присвяченої Дню науки, 12-17 травня 2025 року. Житомир: «Житомирська політехніка», 2025. С. 54-55
3. Google Forms [Електронний ресурс] – Режим доступу до ресурсу: <https://workspace.google.com/products/forms/>
4. Дія опитування [Електронний ресурс] – Режим доступу до ресурсу: <https://center.diia.gov.ua/questionnaire>
5. Election Runner [Електронний ресурс] – Режим доступу до ресурсу: <https://electionrunner.com/>
6. Документація Spring Framework [Електронний ресурс] – Режим доступу до ресурсу: <https://spring.io/projects/spring-framework>
7. Документація Hibernate ORM [Електронний ресурс] – Режим доступу до ресурсу: <https://hibernate.org/orm/documentation/>
8. Документація React [Електронний ресурс] – Режим доступу до ресурсу: <https://react.dev/>
9. Документація PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postgresql.org/docs/>
10. Fowler M. Richardson Maturity Model: steps toward the glory of REST [Електронний ресурс] – Режим доступу до ресурсу: <https://martinfowler.com/articles/richardsonMaturityModel.html>
11. Domain-Driven Design for Web Development [Електронний ресурс] – Режим доступу до ресурсу: <https://dev.to/ddd-and-web-development>

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		81

12. Spring Boot Reference Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>
13. MDN Web Docs: JavaScript [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
14. Axios HTTP client Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://axios-http.com/docs/intro>
15. React Router: Declarative routing for React.js [Електронний ресурс] – Режим доступу до ресурсу: <https://reactrouter.com/>
16. MapStruct Reference Guide [Електронний ресурс] – Режим доступу до ресурсу: <https://mapstruct.org/documentation/stable/reference/html/>
17. Spring Security Reference Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.spring.io/spring-security/reference/index.html>
18. Introduction to JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу: <https://jwt.io/introduction>
19. OWASP Top Ten Project [Електронний ресурс] – Режим доступу до ресурсу: <https://owasp.org/www-project-top-ten/>
20. Vite Official Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://vitejs.dev/guide/>
21. Baeldung. Introduction to Spring Data JPA [Електронний ресурс] – Режим доступу до ресурсу: <https://www.baeldung.com/spring-data-jpa-intro>
22. DigitalOcean Community. How To Set Up a React Project with Vite [Електронний ресурс] – Режим доступу до ресурсу: <https://www.digitalocean.com/community/tutorials/how-to-set-up-a-react-project-with-vite>
23. LogRocket Blog. A complete guide to React Context API [Електронний ресурс] – Режим доступу до ресурсу: <https://blog.logrocket.com/a-complete-guide-to-react-context-api/>
24. The Official TypeScript Handbook [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org/docs/handbook/intro.html>

25. Java™ Platform, Standard Edition & Java Development Kit Version 17 API Specification [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.oracle.com/en/java/javase/17/docs/api/index.html>
26. Jackson Deserialization with ObjectMapper (Official Guide) [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/FasterXML/jackson-docs/wiki/JacksonDeserialization>
27. FlywayDB Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://flywaydb.org/documentation/>
28. Material-UI (MUI) Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://mui.com/material-ui/getting-started/>
29. React-toastify Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://fkhadra.github.io/react-toastify/introduction>
30. React Beautiful DnD Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/atlassian/react-beautiful-dnd>
31. i18next Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://www.i18next.com/overview/getting-started>
32. GitHub: Official Website [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com>
33. Clean Architecture by Robert C. Martin (Uncle Bob) - A Summary [Електронний ресурс] – Режим доступу до ресурсу: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>

ДОДАТКИ

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		84

```

@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
@EnableMethodSecurity(prePostEnabled = true)
public class SecurityConfig {

    @Value("${cors.allowed.origins}")
    private List<String> allowedOrigins;

    private final JwtAuthenticationFilter jwtAuthFilter;
    private final AuthenticationProvider
authenticationProvider;
    private final RateLimitFilter rateFilter;

    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity
http) throws Exception {
        http.csrf(csrf -> csrf.disable()).cors(cors ->
cors.configurationSource(corsConfigurationSource()))
        .authorizeHttpRequests(
            auth ->
auth.requestMatchers("/api/auth/**", "/error",
"/api/statistics/total").permitAll().anyRequest().authenticate
d())
        .sessionManagement(session ->
session.sessionCreationPolicy(SessionCreationPolicy.STATELESS)
)

        .authenticationProvider(authenticationProvider)
        .addFilterBefore(jwtAuthFilter,
UsernamePasswordAuthenticationFilter.class)
        .addFilterAfter(rateFilter,
JwtAuthenticationFilter.class);
        //.exceptionHandling(exception ->
exception.authenticationEntryPoint(new
CustomAuthenticationEntryPoint()));

        return http.build();
    }

    @Bean
    CorsConfigurationSource corsConfigurationSource() {

```

		Озорнін Я.В.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		85

```

        CorsConfiguration configuration = new
CorsConfiguration();
        configuration.setAllowCredentials(true);
        configuration.setAllowedOrigins(allowedOrigins);
        configuration.addAllowedHeader("*");
        configuration.setMaxAge(3600L);
        configuration.setAllowedMethods(Arrays.asList("GET",
"POST", "PUT", "DELETE", "PATCH"));
        UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**",
configuration);
        return source;

    }

}

```

Додаток Б

```

@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends
OncePerRequestFilter {

    private final JwtService jwtService;
    private final UserDetailsService userDetailsService;
    private final MessageSource messageSource;
    private final LocaleResolver localeResolver;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletRequestResponse response, FilterChain
filterChain)
        throws ServletException, IOException {

        if (request.getServletPath().contains("/auth")) {
            filterChain.doFilter(request, response);
            return;
        }

        final String authHeader =
request.getHeader("Authorization");

```

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		86

```

        final String jwt;
        final String userEmail;

        if (authHeader == null ||
!authHeader.startsWith("Bearer ")) {
            filterChain.doFilter(request, response);
            return;
        }

        jwt = authHeader.substring(7);
        Locale locale =
localeResolver.resolveLocale(request);

        try {
            userEmail = jwtService.extractUsername(jwt);
        } catch (ExpiredJwtException ex) {
            setErrorResponse(response,
HttpServletResponse.SC_UNAUTHORIZED,
                messageSource.getMessage("jwt.expired", null,
locale));
            return;
        } catch (Exception ex) {
            setErrorResponse(response,
HttpServletResponse.SC_UNAUTHORIZED,
                messageSource.getMessage("jwt.invalid", null,
locale));
            return;
        }

        if (userEmail != null &&
SecurityContextHolder.getContext().getAuthentication() ==
null) {
            UserDetails userDetails =
this.userDetailsService.loadUserByUsername(userEmail);
            if (jwtService.isTokenValid(jwt, userDetails) &&
userDetails.isAccountNonLocked()) {
                UsernamePasswordAuthenticationToken
authToken =
                    new
UsernamePasswordAuthenticationToken(userDetails, null,
userDetails.getAuthorities());
                authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
            }
        }
    }
}

```

```

        SecurityContextHolder.getContext().setAuthentication(auth
Token);
    }
}
filterChain.doFilter(request, response);
}

private void setErrorResponse(HttpServletResponse
response, int status, String message) throws IOException {
    response.setStatus(status);
    response.setCharacterEncoding("UTF-8");
    response.setContentType("application/json");
    response.getWriter().write("{\"message\": \"" +
message + "\"}");
}
}

```

Додаток В

@Component

@RequiredArgsConstructor

public class RateLimitFilter extends OncePerRequestFilter {

```

    private final Map<String, Bucket> buckets = new
ConcurrentHashMap<>();

```

```

    private final MessageSource messageSource;
    private final LocaleResolver localeResolver;
    private final Logger logger =

```

```

LoggerFactory.getLogger(RateLimitFilter.class);

```

```

    private Bucket resolveBucket(String userEmail) {
        return buckets.computeIfAbsent(userEmail, id ->
            Bucket.builder()
                .addLimit(Bandwidth.classic(20,
Refill.greedy(20, Duration.ofSeconds(10))))
                .build()
    );
}

```

@Override

		Озорнін Я.В.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		88

```

        protected void doFilterInternal(HttpServletRequest request,
        HttpServletResponse response, FilterChain
        filterChain)
            throws ServletException, IOException {

            Authentication authentication =
            SecurityContextHolder.getContext().getAuthentication();

            if (authentication != null &&
            authentication.isAuthenticated()) {
                String userId = authentication.getName();

                Bucket bucket = resolveBucket(userId);
                if (!bucket.tryConsume(1)) {

                    Locale locale =
                    localeResolver.resolveLocale(request);
                    logger.warn("Rate limit exceeded for user:
                    {}", userId);
                    setErrorResponse(response,
                    HttpStatus.TOO_MANY_REQUESTS.value(),
                    messageSource.getMessage("rate.limit.exceeded", null,
                    locale));
                    return;
                }
            }

            filterChain.doFilter(request, response);
        }

        private void setErrorResponse(HttpServletResponse
        response, int status, String message) throws IOException {
            response.setStatus(status);
            response.setCharacterEncoding("UTF-8");
            response.setContentType("application/json");
            response.getWriter().write("{\"message\": \"" +
            message + "\"}");
        }
    }

```

```

@Aspect
@Component
public class LoggingAspect {

    private static final Logger LOGGER =
LoggerFactory.getLogger(LoggingAspect.class);
    private final ObjectMapper objectMapper;

    public LoggingAspect(ObjectMapper objectMapper) {
        this.objectMapper = objectMapper;

        this.objectMapper.disable(SerializationFeature.WRITE_DATE
S_AS_TIMESTAMPS);
        this.objectMapper.findAndRegisterModules();
    }

    //@Value("${logging.service.enabled:true}")
    private boolean serviceLoggingEnabled = true;

    //@Value("${logging.controller.enabled:true}")
    private boolean controllerLoggingEnabled = true;

    @Around("@within(org.springframework.stereotype.Service)"
)
    public Object logAround(ProceedingJoinPoint joinPoint)
throws Throwable {

        if (!serviceLoggingEnabled) {
            return joinPoint.proceed();
        }

        LOGGER.info("Method called: " +
joinPoint.getSignature().getName());

        Object result;

        try {
            result = joinPoint.proceed();
        } catch (Throwable throwable) {
            LOGGER.error("Exception in method: " +
joinPoint.getSignature().getName());

```

		Озорнін Я.В.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Локтікова Т.М.				90
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        LOGGER.error("With arguments: " +
Arrays.toString(joinPoint.getArgs()), throwable);
        throw throwable;
    }
    LOGGER.info("Method executed: " +
joinPoint.getSignature().getName() + " with result: " +
result);
    return result;
}

@Around("@within(org.springframework.web.bind.annotation.
RestController)")
public Object logAroundController(ProceedingJoinPoint
joinPoint) throws Throwable {

    if (!controllerLoggingEnabled) {
        return joinPoint.proceed();
    }

    HttpServletRequest request =
((ServletRequestAttributes)
RequestContextHolder.currentRequestAttributes())
.getRequest();

    String requestId = UUID.randomUUID().toString();
    long startTime = System.currentTimeMillis();

    String methodName =
joinPoint.getSignature().getName();
    String className =
joinPoint.getTarget().getClass().getSimpleName();

    LOGGER.info("[{}] Request: {} {} invoked {}::{}",
requestId, request.getMethod(), request.getRequestURI(),
className, methodName);

    Object[] args = joinPoint.getArgs();
    if (args != null && args.length > 0) {
        for (Object arg : args) {
            if (arg != null) {
                if (arg instanceof ServletRequest ||
arg instanceof ServletResponse) {
                    continue;

```

		Озорнін Я.В.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		91

```

        }
        try {
            LOGGER.info("[{}] Request
Argument: {}", requestId,
objectMapper.writeValueAsString(arg));
        } catch (Exception e) {
            LOGGER.info("[{}] Request
Argument: [Could not serialize - {}]", requestId,
            arg.getClass().getName());
        }
    }
}

Object result;
try {
    result = joinPoint.proceed();
} catch (Exception e) {
    LOGGER.error("[{}] Exception: {} - {}",
requestId, e.getClass().getName(), e.getMessage());
    throw e;
}

long duration = System.currentTimeMillis() -
startTime;
try {
    if (result != null) {
        LOGGER.info("[{}] Response from {}::{} ({}
ms): {}", requestId, className, methodName, duration,
        objectMapper.writeValueAsString(result));
    } else {
        LOGGER.info("[{}] Response from {}::{} ({}
ms): null", requestId, className, methodName, duration);
    }
} catch (Exception e) {
    LOGGER.info(e.getMessage());
}

return result;
}
}

```

		Озорнін Я.В.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		92

```

public User getCurrentUser() {
    Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
    if (authentication != null &&
authentication.getPrincipal() instanceof User) {
        return (User) authentication.getPrincipal();
    }
    throw new SecurityException("User not authenticated
or invalid user type");
}

```

```

@Entity
@Table(name = "surveys")
@Data
@NoArgsConstructor
@AllArgsConstructor
@EqualsAndHashCode(callSuper = true)
@Builder
public class Survey extends Auditable implements Reportable {

    @Id
    @GeneratedValue(generator = "uuid")
    @UUIDv7
    @Column(columnDefinition = "UUID", updatable = false,
nullable = false)
    private UUID id;

    @Column(nullable = false)
    private String title;

    @Column(columnDefinition = "TEXT")
    private String description;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "creator_id")
    @JsonBackReference
    private User creator;

    @Enumerated(EnumType.STRING)

```

```

@Column(nullable = false)
private SurveyStatus status;

@Column(name = "expiration_date")
private LocalDateTime expirationDate;

@Column(name = "is_anonymous", nullable = false)
private Boolean isAnonymous;

@Column(name = "start_date", nullable = false)
private LocalDateTime startDate;

@OneToMany(mappedBy = "survey", cascade = CascadeType.ALL,
orphanRemoval = true)
@JsonManagedReference
private List<Question> questions = new ArrayList<>();

@OneToMany(mappedBy = "survey", cascade = CascadeType.ALL,
orphanRemoval = true)
@JsonManagedReference
private List<Comment> comments = new ArrayList<>();

@OneToMany(mappedBy = "survey", cascade = CascadeType.ALL,
orphanRemoval = true)
@JsonManagedReference
private List<CompletedSurvey> completedSurveys = new
ArrayList<>();

public enum SurveyStatus {
    DRAFT, PUBLISHED, ARCHIVED
}

@Override
public UUID getAuthorId() {
    return creator != null ? creator.getId() : null;
}

public void addQuestion(Question question) {
    questions.add(question);
    question.setSurvey(this);
}

public void removeQuestion(Question question) {
    questions.remove(question);
}

```

		Озорнін Я.В.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		94

```

        question.setSurvey(null);
    }

    public void addComment(Comment comment) {
        comments.add(comment);
        comment.setSurvey(this);
    }

    public void removeComment(Comment comment) {
        comments.remove(comment);
        comment.setSurvey(null);
    }
}

```

Додаток 3

```

@Repository
public interface SurveyRepository extends
JpaRepository<Survey, UUID>, BaseRepository<Survey> {

    @Query("""
        SELECT s FROM Survey s
        WHERE s.creator.id = :userId and (s.status =
'PUBLISHED' or s.status = 'ARCHIVED')
        """)
    Page<Survey> findByCreatorId(UUID userId, Pageable
pageable);

    Page<Survey> findAllByCreatorId(UUID userId, Pageable
pageable);

    void deleteById(UUID id);

    Page<Survey> findAllByStatus(SurveyStatus published,
PageRequest of);

    @Query("""
        SELECT COUNT(DISTINCT v.option.question.id)
        FROM Vote v
        WHERE v.user.id = :userId AND
v.option.question.survey.id = :surveyId
        """)

```

		Озорнін Я.В.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		95

```

    long countQuestionsAnswered(@Param("userId") UUID userId,
    @Param("surveyId") UUID surveyId);

    @Query("""
        SELECT COUNT(q.id)
        FROM Question q
        WHERE q.survey.id = :surveyId
        """)
    long countQuestionsInSurvey(@Param("surveyId") UUID
    surveyId);

    @Query("""
        Select s from Survey s
        where s.status = 'PUBLISHED' and
    s.expirationDate < current_timestamp
        """)
    List<Survey> findAllExpiredPublishedSurveys();

    @Query("""
        SELECT COUNT(s) FROM Survey s
        WHERE s.status = :status
        """)
    Long countByStatus(Survey.SurveyStatus status);

    @Query("""
        SELECT new
    com.ozzy.surveyapi.project.dto.SurveyVoteCount(q.survey,COUNT(
    v))
        FROM Vote v
        JOIN v.option o
        JOIN o.question q
        GROUP BY q.survey
        ORDER BY COUNT(v) DESC
        """)
    List<SurveyVoteCount>
    findMostPopularSurveysByVotes(Pageable pageable);

    @Query("""
        SELECT s
        FROM Survey s
        WHERE s.status = 'PUBLISHED'
        ORDER BY s.createdAt DESC
        """)
    List<Survey> findRecentSurveys(Pageable pageable);

```

		Озорнін Я.В.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		96

```

        @Query("""
            SELECT COUNT(s) from Survey s where s.creator.id
            = :id
            """)
        Long countById(UUID id);
    }

```

Додаток II

```

@Override
    @PreAuthorize("hasAnyAuthority('ROLE_CAN_CREATE_SURVEYS',
    'ROLE_ADMIN')")
    public SurveyDTO createSurvey(SurveyDTO surveyDTO) {
        Survey survey = surveyMapper.toEntity(surveyDTO);
        User creator =
        userRepository.getReferenceById(ownershipService.getCurrentUse
        rId());
        survey.setStatus(Survey.SurveyStatus.DRAFT);
        survey.setCreator(creator);
        return
        surveyMapper.toDTO(surveyRepository.save(survey));
    }

```

Додаток I

```

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class SurveyDTO implements Reportable {

    private UUID id;
    private String title;
    private String description;
    private UUID creatorId;
    private SurveyStatus status;
    private LocalDateTime expirationDate;
    private Boolean isAnonymous;
    private LocalDateTime startDate;
    private QuestionDTO[] questions;
    private boolean isCompleted;

```

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Локтікова Т.М.				97
Змн.	Арк.	№ докум.	Підпис	Дата		

```

@Override
public UUID getAuthorId() {
    return creatorId;
}
}

```

Додаток Е

```

@ALLArgsConstructor
@NoArgsConstructor
@Getter
@Setter
@Builder
public class SurveyImportDTO {

    @NotBlank(message = "survey.title.not.blank")
    private String title;

    @NotBlank(message = "survey.description.not.blank")
    private String description;

    @Future(message = "survey.expirationDate.future")
    private LocalDateTime expirationDate;

    @NotNull(message = "survey.is.Anonymous.not.blank")
    private Boolean isAnonymous;

    @Valid
    private List<QuestionImportDTO> questions;
}

```

Додаток Є

```

@Mapper(componentModel = "spring")
public interface SurveyMapper {

    @Mapping(target = "creatorId", source = "creator.id")
    @Mapping(target = "questions", ignore = false)
    SurveyDTO toDTO(Survey survey);

    Survey toEntity(SurveyDTO surveyDTO);
}

```

		Озорнін Я.В.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Локтікова Т.М.				98
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    @BeanMapping(nullValuePropertyMappingStrategy =
        NullValuePropertyMappingStrategy.IGNORE)
    void updateSurveyFromPatch(SurveyPatchDTO patch,
        @MappingTarget Survey survey);

    @Mapping(target = "numberOfQuestions", expression =
        "java(survey.getQuestions().size())")
    SurveyInfoDTO toInfoDTO(Survey survey);

}

```

Додаток І

```

@RestControllerAdvice
@RequiredArgsConstructor
public class GlobalExceptionHandler {

    private final MessageSource messageSource;

    @ExceptionHandler(Exception.class)
    public ResponseEntity<ErrorResponse>
    handleGenericException(Exception ex, Locale locale) {
        return buildErrorResponse(ex.getMessage(),
            HttpStatus.INTERNAL_SERVER_ERROR, null);
    }

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<Map<String, Map<String, String>>>
    handleValidationErrors(MethodArgumentNotValidException ex,
        Locale locale) {
        Map<String, Map<String, String>> response = new
        HashMap<>();
        Map<String, String> errors = new HashMap<>();

        ex.getBindingResult().getFieldErrors().forEach(error
        -> {
            String field = error.getField();
            String message = messageSource.getMessage(error,
            locale);
            errors.put(field, message);
        });
        response.put("errors", errors);
    }
}

```

		Озорнін Я.В.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Локтікова Т.М.				99
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        return ResponseEntity.badRequest().body(response);
    }

    @ExceptionHandler(UserIsBannedException.class)
    public ResponseEntity<ErrorResponse>
    handleUserIsBannedException(UserIsBannedException ex, Locale
    locale) {
        return buildErrorResponse(ex.getMessage(),
    HttpStatus.FORBIDDEN, locale,

        ex.getDueDate().format(DateTimeFormatter.ofPattern("YYYY-
    MM-dd")), ex.getReason());
    }

    @ExceptionHandler(InvalidDataException.class)
    public ResponseEntity<ErrorResponse>
    handleInvalidDataException(InvalidDataException ex, Locale
    locale) {
        return buildErrorResponse(ex.getMessage(),
    HttpStatus.BAD_REQUEST, locale, ex.getArguments());
    }

    @ExceptionHandler(AuthorizationDeniedException.class)
    @ResponseStatus(HttpStatus.FORBIDDEN)
    public ResponseEntity<ErrorResponse>
    handleAuthorizationDenied(AuthorizationDeniedException e,
    Locale locale) {
        System.out.println("AuthorizationDeniedException
    triggered");
        return buildErrorResponse("dont.have.permission",
    HttpStatus.FORBIDDEN, locale);
    }

    @ExceptionHandler(AccessDeniedException.class)
    public ResponseEntity<ErrorResponse>
    handleAccessDenied(Exception e) {
        System.out.println("AccessDeniedException
    triggered");
        return buildErrorResponse("forbidden",
    HttpStatus.FORBIDDEN, null);
    }

    @ExceptionHandler(NoHandlerFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)

```

		Озорнін Я.В.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Локтікова Т.М.				100
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        public ResponseEntity<ErrorResponse>
handleNoHandlerFound(NoHandlerFoundException e, Locale locale)
{
    return buildErrorResponse("Resource not found",
HttpStatus.NOT_FOUND, locale);
}

    @ExceptionHandler(ApiException.class)
    public ResponseEntity<ErrorResponse>
handleBaseException(ApiException ex, Locale locale) {
    return buildErrorResponse(ex.getMessage(),
HttpStatus.valueOf(ex.getStatus()), locale);
}

    @ExceptionHandler(ExpiredJwtException.class)
    public ResponseEntity<ErrorResponse>
handleExpiredJwtException(ExpiredJwtException ex, Locale
locale) {
    return buildErrorResponse("Token expired",
HttpStatus.UNAUTHORIZED, locale);
}

    private ResponseEntity<ErrorResponse>
buildErrorResponse(String message, HttpStatusCode status,
Locale locale,
        Object... args) {
    String localed = messageSource.getMessage(message,
args, locale);
    return ResponseEntity.status(status).body(new
ErrorResponse(localed));
}
}

```

Додаток К

```

@Configuration
public class LocaleConfiguration {

    @Bean
    public MessageSource messageSource() {
        ResourceBundleMessageSource source = new
ResourceBundleMessageSource();
        source.setBasename("messages");
    }
}

```

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Локтікова Т.М.				101
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        source.setDefaultEncoding("UTF-8");
        source.setUseCodeAsDefaultMessage(true);
        return source;
    }

    @Bean
    public LocaleResolver localeResolver() {
        AcceptHeaderLocaleResolver resolver = new
AcceptHeaderLocaleResolver();
        resolver.setDefaultLocale(Locale.ENGLISH);
        return resolver;
    }

    @Bean
    public LocalValidatorFactoryBean getValidator() {
        LocalValidatorFactoryBean factoryBean = new
LocalValidatorFactoryBean();

        factoryBean.setValidationMessageSource(messageSource());
        return factoryBean;
    }
}

```

Додаток Л

```

@RequestMapping("/api/surveys")
@RequiredArgsConstructor
public class SurveyController {

    private final SurveyService service;
    private final SecurityUtils securityUtils;

    @GetMapping("/my")
    public ResponseEntity<Page<SurveyDTO>> getMySurveys(
        @RequestParam(defaultValue = "0") int page,
        @RequestParam(defaultValue = "10") int size) {
        Page<SurveyDTO> surveys =
service.getUserSurveys(PageRequest.of(page, size));
        return ResponseEntity.ok(surveys);
    }

    @GetMapping()

```

		Озорнін Я.В.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Локтікова Т.М.				102
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        public ResponseEntity<Page<SurveyInfoDTO>> getSurveys(
            @RequestParam(defaultValue = "0") int page,
            @RequestParam(defaultValue = "10") int size,
            @RequestParam(defaultValue = "title") String
sortBy,
            @RequestParam(defaultValue = "asc") String
sortOrder,
            @ModelAttribute SurveyFilter filter) {

            Page<SurveyInfoDTO> surveys =
service.getSurveys(page, size, sortBy, sortOrder, filter);
            return ResponseEntity.ok(surveys);
        }

        @GetMapping("/{id}/export")
        public ResponseEntity<SurveyImportDTO>
exportSurvey(@PathVariable UUID id) {
            SurveyImportDTO survey = service.exportSurvey(id);
            return ResponseEntity.ok(survey);
        }

        @PostMapping("/import")
        public ResponseEntity<SurveyDTO>
importSurvey(@RequestBody @Valid SurveyImportDTO
surveyImportDTO) {
            SurveyDTO importedSurvey =
service.importSurvey(surveyImportDTO);
            return
ResponseEntity.status(HttpStatus.CREATED).body(importedSurvey)
;
        }

        @GetMapping("/public")
        public ResponseEntity<Page<SurveyInfoDTO>>
getPublicSurveys(
            @RequestParam(defaultValue = "0") int page,
            @RequestParam(defaultValue = "10") int size) {
            return
ResponseEntity.ok(service.getPublicSurveys(PageRequest.of(page
, size)));
        }

        @GetMapping("/{id}")

```

```

        public ResponseEntity<SurveyDTO>
getSurveyById(@PathVariable UUID id) {
    SurveyDTO survey = service.getSurveyById(id);
    return ResponseEntity.ok(survey);
}

@PostMapping("/{id}/publish")
public ResponseEntity<SurveyDTO>
publishSurvey(@PathVariable UUID id, @RequestBody
SurveyPublishRequest data) {
    return ResponseEntity.status(HttpStatus.CREATED)
        .body(service.publishSurvey(id, data));
}

@PostMapping("/{id}/submit")
public ResponseEntity<SurveyDTO>
submitSurvey(@PathVariable UUID id) {
    service.submitSurvey(id);
    return ResponseEntity.created(null).build();
}

@PostMapping()
public ResponseEntity<SurveyDTO>
createSurvey(@RequestBody SurveyDTO surveyDTO) {
    SurveyDTO createdSurvey =
service.createSurvey(surveyDTO);
    return
ResponseEntity.status(HttpStatus.CREATED).body(createdSurvey);
}

@PatchMapping("/{id}")
public ResponseEntity<SurveyDTO>
updateSurvey(@PathVariable String id, @RequestBody
SurveyPatchDTO surveyDTO) {
    SurveyDTO updatedSurvey =
service.updateSurvey(UUID.fromString(id), surveyDTO);
    return ResponseEntity.ok(updatedSurvey);
}

@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteSurvey(@PathVariable
String id) {
    service.deleteSurvey(UUID.fromString(id));
    return ResponseEntity.noContent().build();
}

```

```

    }

}

```

Додаток М

```

const api = axios.create({
  baseURL: import.meta.env.VITE_API_URL,
  headers: {
    "Content-Type": "application/json",
  },
});

api.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem("token");
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    config.headers["Accept-Language"] = i18n.language || "en";
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);

api.interceptors.response.use(
  (response) => {
    console.log("Response:", response);
    return response;
  },
  (error) => {
    if (error.response) {
      toast.error(
        `Error: ${error.response.data?.message || t("Something went wrong!")}`
      );

      if (error.response.status === 401) {
        localStorage.removeItem("token");
        navigateTo("/login");
      }
    } else {
      toast.error("Network error. Please try again later.");
    }
    return Promise.reject(error);
  }
);

```

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		105

```
export default api;
```

Додаток Н

```
function App() {
  const navigate = useNavigate();

  React.useEffect(() => {
    setNavigate(navigate);
  }, [navigate]);

  return (
    <AuthProvider>
      <RBACProvider>
        <MainLayout>
          <Routes>
            <Route path="/" element={<HomePage />} />
            <Route path="/register" element={<RegisterPage />} />
            <Route path="/login" element={<LoginPage />} />
            <Route path="/forgot-password" element={<ForgotPasswordPage />} />
            <Route path="/verify-email" element={<EmailVerificationPage />} />
            <Route path="/reset-password" element={<PasswordResetPage />} />
            <Route
              path="/dashboard"
              element={
                <ProtectedRoute
                  requiredRoles={[
                    Roles.ADMIN,
                    Roles.MODERATOR,
                    Roles.CAN_CREATE_SURVEYS,
                  ]}
                >
                  <MySurveysPage />
                </ProtectedRoute>
              }
            />
            <Route
              path="/surveys"
              element={
                <ProtectedRoute
                  requiredRoles={[
                    Roles.ADMIN,
                    Roles.MODERATOR,
                    Roles.USER,
                    Roles.CAN_PASS_SURVEYS,
                  ]}
                >
                  <PublicSurveysPage />
                </ProtectedRoute>
              }
            />
          </Routes>
        </MainLayout>
      </RBACProvider>
    </AuthProvider>
  );
}
```

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ІІЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		106

```

/>
<Route
  path="/surveys/:id/vote"
  element={
    <ProtectedRoute
      requiredRoles={[
        Roles.ADMIN,
        Roles.MODERATOR,
        Roles.USER,
        Roles.CAN_PASS_SURVEYS,
      ]}
    >
      <SurveyVotingPage />
    </ProtectedRoute>
  }
/>
<Route
  path="/surveys/:id/statistics"
  element={
    <ProtectedRoute
      requiredRoles={[
        Roles.ADMIN,
        Roles.MODERATOR,
        Roles.USER,
        Roles.CAN_PASS_SURVEYS,
      ]}
    >
      <SurveyStatisticsPage />
    </ProtectedRoute>
  }
/>
<Route
  path="/admin/users"
  element={
    <ProtectedRoute requiredRoles={[Roles.ADMIN, Roles.MODERATOR]}}>
      <UserManagementPage />
    </ProtectedRoute>
  }
/>
<Route
  path="/admin/reports"
  element={
    <ProtectedRoute requiredRoles={[Roles.ADMIN, Roles.MODERATOR]}}>
      <ReportsManagementPage />
    </ProtectedRoute>
  }
/>
<Route
  path="/profile"
  element={

```

```

        <ProtectedRoute
            requiredRoles={[Roles.ADMIN, Roles.MODERATOR, Roles.USER]}
        >
            <UserProfilePage />
        </ProtectedRoute>
    }
    />
</Routes>
</MainLayout>
<ToastContainer position="top-right" autoClose={3000} />
</RBACProvider>
</AuthProvider>
);
}

export default App;

```

Додаток О

```

interface AuthContextType {
    user: UserDTO | null;
    setUser: React.Dispatch<React.SetStateAction<UserDTO | null>>;
    login: (req: AuthRequest) => Promise<void>;
    register: (req: RegisterRequest) => Promise<void>;
    logout: () => void;
    isAuthenticated: boolean;
    isAdmin: boolean;
    isModerator: boolean;
    isLoading: boolean;
}

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export const AuthProvider: React.FC<{ children: React.ReactNode }> = ({
    children,
}) => {
    const [user, setUser] = useState<UserDTO | null>(null);
    const [isLoading, setIsLoading] = useState(true);

    useEffect(() => {
        const validateTokenAndGetUser = async () => {
            const token = localStorage.getItem("token");

            if (!token) {
                setIsLoading(false);
                return;
            }

            try {
                const userProfile = await userProfileApi.getCurrentUserProfile();

```

		Озорнін Я.В.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Локтікова Т.М.				108
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        if (userProfile.user) {
            setUser(userProfile.user);
            console.log("Token is valid, user authenticated:", userProfile.user);
        }
    } catch (error) {
        console.log("Token validation failed:", error);
        localStorage.removeItem("token");
        localStorage.removeItem("user");
        setUser(null);
    } finally {
        setIsLoading(false);
    }
};

validateTokenAndGetUser();
}, []);
const login = async (req: AuthRequest) => {
    const data = await apiLogin(req);
    setUser(data.user);
};

const register = async (req: RegisterRequest) => {
    const data = await apiRegister(req);
    setUser(data.user);
};

const logout = () => {
    setUser(null);
    localStorage.removeItem("token");
    localStorage.removeItem("user");
};

const isAdmin = user?.roles.includes(Roles.ADMIN) || false;
const isModerator = user?.roles.includes(Roles.MODERATOR) || false;

return (
    <AuthContext.Provider
        value={{
            user,
            setUser,
            login,
            register,
            logout,
            isAuthenticated: user !== null,
            isAdmin,
            isModerator,
            isLoading,
        }}
    >

```

```

        {children}
      </AuthContext.Provider>
    );
  };

function useAuth() {
  const context = useContext(AuthContext);
  if (!context) {
    throw new Error("useAuth must be used within an AuthProvider");
  }
  return context;
}
export { useAuth };

```

Додаток П

```

interface RBACContextType {
  hasRole: (role: Roles) => boolean;
  hasAnyRole: (roles: Roles[]) => boolean;
  canAccessPage: (requiredRoles: Roles[]) => boolean;
  canCreateSurveys: () => boolean;
  canPassSurveys: () => boolean;
  canWriteComments: () => boolean;
  isAdmin: () => boolean;
  isModerator: () => boolean;
}

const RBACContext = createContext<RBACContextType | undefined>(undefined);

export const RBACProvider: React.FC<{ children: React.ReactNode }> = ({
  children,
}) => {
  const { user } = useAuth();
  const hasRole = (role: Roles): boolean => {
    if (!user || !user.roles) return false;
    return user.roles.includes(role);
  };

  const hasAnyRole = (roles: Roles[]): boolean => {
    if (!user || !user.roles) return false;
    return roles.some((role) => user.roles.includes(role));
  };

  const canAccessPage = (requiredRoles: Roles[]): boolean => {
    if (!user || !user.roles) return false;

    if (
      user.roles.includes(Roles.ADMIN) ||
      user.roles.includes(Roles.MODERATOR)
    )

```

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		110

```

    ) {
      return true;
    }

    return requiredRoles.some((role) => user.roles.includes(role));
  };

  const canCreateSurveys = (): boolean => {
    if (!user) return false;
    return hasAnyRole([Roles.ADMIN, Roles.MODERATOR, Roles.CAN_CREATE_SURVEYS]);
  };

  const canPassSurveys = (): boolean => {
    if (!user) return false;
    return hasAnyRole([
      Roles.ADMIN,
      Roles.MODERATOR,
      Roles.USER,
      Roles.CAN_PASS_SURVEYS,
    ]);
  };

  const canWriteComments = (): boolean => {
    if (!user) return false;
    return hasAnyRole([Roles.ADMIN, Roles.MODERATOR, Roles.CAN_WRITE_COMMENTS]);
  };

  const isAdmin = (): boolean => {
    return hasRole(Roles.ADMIN);
  };

  const isModerator = (): boolean => {
    return hasRole(Roles.MODERATOR);
  };

  const value = useMemo(
    () => ({
      hasRole,
      hasAnyRole,
      canAccessPage,
      canCreateSurveys,
      canPassSurveys,
      canWriteComments,
      isAdmin,
      isModerator,
    }),
    [user]
  );

  return <RBACContext.Provider value={value}>{children}</RBACContext.Provider>;

```

		Озорнін Я.В.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Локтікова Т.М.				111
Змн.	Арк.	№ докум.	Підпис	Дата		

```

};

function useRBAC() {
  const context = useContext(RBACContext);
  if (!context) {
    throw new Error("useRBAC must be used within an RBACProvider");
  }
  return context;
}
export { useRBAC };

```

Додаток Р

```

interface ProtectedRouteProps {
  children: React.ReactNode;
  requiredRoles?: Roles[];
  requireAuth?: boolean;
}

const ProtectedRoute: React.FC<ProtectedRouteProps> = ({
  children,
  requiredRoles = [],
  requireAuth = true,
}) => {
  const { isAuthenticated, isLoading } = useAuth();
  const { canAccessPage } = useRBAC();

  if (isLoading) {
    return (
      <Box
        display="flex"
        justifyContent="center"
        alignItems="center"
        minHeight="200px"
      >
        <CircularProgress />
      </Box>
    );
  }

  if (requireAuth && !isAuthenticated) {
    console.log("User is not authenticated, redirecting to login");
    return <Navigate to="/login" replace />;
  }

  if (requiredRoles.length > 0 && !canAccessPage(requiredRoles)) {
    return <Navigate to="/" replace />;
  }

  return <>{children}</>;

```

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Локтікова Т.М.				112
Змн.	Арк.	№ докум.	Підпис	Дата		

```
};
```

```
export default ProtectedRoute;
```

Додаток С

```
import { AuthRequest, RegisterRequest } from "../types/payloads";  
import api from "../axiosInstance";
```

```
export const login = async (req: AuthRequest) => {  
  try {  
    const response = await api.post("/auth/login", req);  
    if (response.data.token) {  
      storeToken(response.data.token);  
    }  
    return response.data;  
  } catch (error) {  
    console.error("Login error:", error);  
    throw error;  
  }  
};
```

```
export const register = async (req: RegisterRequest) => {  
  try {  
    const response = await api.post("/auth/register", req);  
    if (response.data.token) {  
      storeToken(response.data.token);  
    }  
    return response.data;  
  } catch (error) {  
    console.error("Register error:", error);  
    throw error;  
  }  
};
```

```
export const verifyEmail = async (token: string) => {  
  try {  
    const response = await api.post(`/auth/verify-email?token=${token}`);  
    return response.data;  
  } catch (error) {  
    console.error("Email verification error:", error);  
    throw error;  
  }  
};
```

```
export const resetPassword = async (token: string, newPassword: string) => {  
  try {  
    const response = await api.post("/auth/reset-password", {  
      token,  
      newPassword,  
    });  
  }  
};
```

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		113

```

    });
    return response.data;
  } catch (error) {
    console.error("Password reset error:", error);
    throw error;
  }
};

export const forgotPassword = async (email: string) => {
  try {
    const response = await api.post("/auth/forgot-password", {
      email,
    });
    return response.data;
  } catch (error) {
    console.error("Forgot password error:", error);
    throw error;
  }
};

const storeToken = (token: string) => {
  localStorage.setItem("token", token);
  api.defaults.headers.common["Authorization"] = `Bearer ${token}`;
};

```

Додаток С

```

const handleOptionChange = async (id: string, updatedOption: OptionDTO) => {
  const changedFields = calculateChangedFields(
    options.find((o) => o.id === id),
    updatedOption
  );
  try {
    const serverOption = await patchOption(id, changedFields);
    setOptions((prev) =>
      prev.map((o) => {
        if (o.id === id) return { ...o, ...serverOption };
        return editQuestion.questionType === "SINGLE_CHOICE"
          ? { ...o, isCorrect: false }
          : o;
      })
    );
  } catch {}
};

<DragDropContext onDragEnd={disabled ? () => {} : handleDragEnd}>
  <Draggable draggableId="options-droppable" isDropDisabled={disabled}>
    {(provided) => (
      <Box
        ref={provided.innerRef}

```

		Озорнін Я.В.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Локтікова Т.М.				114
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    {...provided.droppableProps}
    sx={{
      pl: 2,
      borderLeft: `3px solid ${theme.palette.info.light}`,
      mb: 1,
    }}
  >
  {sortedOptions.length > 0 ? (
    sortedOptions.map((option, idx) => (
      <Draggable
        key={option.id}
        draggableId={option.id}
        index={idx}
        isDragDisabled={disabled}
      >
        {(dragProvided, dragSnapshot) => (
          <div
            ref={dragProvided.innerRef}
            {...dragProvided.draggableProps}
            style={{
              ...dragProvided.draggableProps.style,
              opacity: dragSnapshot.isDragging ? 0.7 : 1,
            }}
          >
            <Box sx={{ display: "flex", alignItems: "center" }}>
              {" "}
              <Box
                {...(disabled
                  ? {}
                  : dragProvided.dragHandleProps)}
                sx={{
                  display: "flex",
                  alignItems: "center",
                  justifyContent: "center",
                  width: 32,
                  height: 32,
                  cursor: disabled ? "default" : "grab",
                  color: disabled
                    ? theme.palette.action.disabled
                    : theme.palette.action.active,
                  "&:hover": disabled
                    ? {}
                    : {
                      color: theme.palette.primary.main,
                      backgroundColor:
                        theme.palette.action.hover,
                    },
                  borderRadius: 1,
                  mr: 1,
                }}
              <

```

		Озорнін Я.В.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Локтікова Т.М.				115
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    >
      <DragIndicatorIcon fontSize="small" />
    </Box>{" "}
  <Box sx={{ flex: 1 }}>
    <OptionItem
      option={option}
      onChange={(updated) =>
        handleOptionChange(option.id, updated)
      }
      onDelete={handleDeleteOption}
      autoSave
      disabled={disabled}
    />
  </Box>
</Box>
</div>
  )}
</Draggable>

```

		Озорнін Я.В.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Локтікова Т.М.				116
Змн.	Арк.	№ докум.	Підпис	Дата		