Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Кастєрова Вячеслава Олександровича*
(ПІБ)

академічної групи *122-20-2*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка інформаційної системи для створення публічних опитувань на основі технологічного стеку ASP.NET Core, VUE.JS, MS SQL*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Ширін А.Л.*			
розділів:				
спеціальний	*доц. Ширін А.Л.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2024

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » _____ 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-20-2* *Кастєрова Вячеслава Олександровича*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка інформаційної ситеми для створення публічних опитувань на основі технологічного стеку ASP.NET Core, VUE.JS, MS SQL*

затверджена наказом ректора НТУ «ДП» від *23.05.2024* № *469-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***01.06.2024 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***06.06.2024 р.***

Завдання видав _____ *доц. Ширін А.Л.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Кастєров В.О.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*

Термін подання кваліфікаційної роботи до ЕК: *24.06.2024 р.*

ЗМІСТ

РЕФЕРАТ

Пояснювальна записка: 73 с., 58 рис., 3 табл., 3 дод., 25 джерел.

Об'єкт розробки: веборієнтована інформаційна платформа для створення публічних опитувань.

Мета кваліфікаційної роботи: розробка інформаційної платформи для створення публічних опитувань на основі технологічного стеку Vue.js та ASP.NET.

У вступі розглядається можливість створення платформи для вільного створення опитувань та подальшого аналізу результатів цих опитувань, а також переваги використання централізованої платформи для створення опитувань.

У першому розділі описуються в подробицях нюанси проведення публічних опитувань з тими платформами та засобами, що вже існують, їх аналіз та порівняння з запропонованою ідеєю.

У другому розділі описана технологічна частина вебплатформи: архітектурні підходи, використані алгоритми шифрування та їх особливості, сутності з якими буде працювати система, патерни програмування, підходи до реалізації авторизації та аутентифікації користувачів, шляхи реалізації цензурної складової на платформі.

В економічному розділі проведено розрахунки вартості розробки даної платформи, включно з виплатами розробникам, розгортанням платформи на серверах та її підтриманням протягом декількох місяців. Також проведено розрахунки часових меж у яких платформа має бути розроблена з урахуванням базових можливостей.

Практичне значення роботи полягає у розробці вебплатформи, де люди (особливо студенти з НТУ "Дніпровська політехніка") вільно зможуть ділитися питаннями, які їх хвилюють та отримувати відповіді від різних вікових, фахових, освітніх, соціальних категорій користувачів, після чого аналізувати відповіді.

Актуальність платформи обумовлюється тим, що після створення опитувань, створених за допомогою різних безкоштовних інструментів, дуже складно знайти групи людей, які могли б відповідати бажаній категорії респондентів. Також, в деяких випадках складно в цілому знайти людей, які готові б були стати респондентами та відповісти на декілька питань.

Список ключових слів: ВЕБПЛАТФОРМА, ОПИТУВАННЯ, РЕСПОНДЕНТИ, C#, VUE.JS, СТУДЕНТИ, МЕРЕЖА.

ABSTRACT

Explanatory note: 73 p., 58 figures, 3 tables, 3 appendixes, 25 sources.

Object of development: web-oriented information platform for creating public polls.

The purpose of the qualification work: development of an information platform for creating public polls based on the Vue.js and ASP.NET technology stack.

The introduction discusses the possibility of creating a platform for free creation of polls and subsequent analysis of the results of these polls, as well as the advantages of using a centralized platform for creating polls.

The first section describes in detail the nuances of conducting public polls with existing platforms and tools, analyzing them and comparing them with the proposed idea.

The second section describes the technological part of the web platform: architectural approaches, encryption algorithms used and their features, entities the system will work with, programming patterns, approaches to implementing user authorization and authentication, ways to implement the censorship component on the platform.

The economic section calculates the cost of developing this platform, including payments to developers, deployment of the platform on servers, and its maintenance for several months. We also calculated the timeframe within which the platform should be developed, taking into account the basic capabilities.

The practical significance of the work is to develop a web platform where people (especially students from NTU "Dnipro Polytechnic") can freely share their concerns and receive

The relevance of the platform is due to the fact that after creating surveys using various free tools, it is very difficult to find groups of people who could match the desired category of respondents. Also, in some cases, it is difficult to find people in general who would be willing to become respondents and answer several questions.

List of keywords: WEB PLATFORM, SURVEY, RESPONDENTS, C#, VUE.JS, STUDENTS, NETWORK.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

JS – Java Script.

HTML – HyperText Markup Language.

CSS – Cascading Style Sheets.

API – Application Program Interface.

РБД – Реляційна База Даних.

SQL – Structured Query Language.

LINQ – Language Integrated Query.

EF – Entity Framework.

DI – Dependency Injection.

CQRS – The Command and Query Responsibility Segregation.

CORS – Cross-Origin Resource Sharing.

JSON – JS Object Notation.

JWT – Json Web Token.

ВСТУП

Соціальне опитування – це список запитань, спрямованих на отримання конкретних даних від певної групи людей. На даний момент в нашому суспільстві існує велика кількість різних платформ для створення опитувань, багато компаній займаються розробкою подібних рішень, але головною проблемою залишається розповсюдження цих опитувань серед людей. Наприклад компанія Google має прекрасний інструментарій для створення опитувань та їх подальшого аналізу – Microsoft Forms. Соціальні опитувальники використовуються для різних цілей [1]:

- Збір даних. Вони дають змогу збирати інформацію про різні аспекти суспільного життя, думки, вподобання, рівень задоволеності, поведінку тощо. Ці дані допомагають зрозуміти суспільні тренди, проблеми та потреби.
- Планування політики. Політики використовують результати опитувань для виявлення вподобань виборців та орієнтування своїх програм і кампаній.
- Оцінка програм і проєктів. Опитування можуть допомогти в оцінці ефективності соціальних програм і проєктів, а також визначенні того, як вони впливають на цільові групи.
- Дослідження. Опитування використовуються для дослідження соціальних явищ, процесів і взаємодій між людьми.
- Прогнозування. На основі результатів опитувань можна робити прогнози щодо майбутніх тенденцій і подій у суспільстві.
- Контроль якості. Опитування можуть допомогти в контролі якості послуг, продуктів або політичних рішень, надаючи зворотний зв'язок від громадської думки.

Загалом, соціальні опитування відіграють важливу роль у збиранні інформації про думки та вподобання суспільства, а також у допомозі ухвалення рішень у політиці, бізнесі та інших сферах діяльності.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

В інтернеті існує багато безкоштовних рішень, що мають функціонал більш спрямований на розробку зовнішнього вигляду опитувань або аналіз результатів опитувань, але не існує рішення, яке б задовільнило головну потребу тих користувачів, що створюють опитування, а саме зручність розповсюдження. Розглянемо рішення від компанії Google. Створення самого опитування не займе багато часу, але після цього користувачу треба розповсюдити це опитування, щоб інші люди могли відповісти на поставлені запитання. Без сторонніх сервісів, це не можливо. Беручи до уваги, що кореспондент має знайти соціальні групи людей, де буде багато потенційних респондентів, дуже розповсюдженим рішенням є використання месенджерів та соціальних платформ для поширення створених опитувань, наприклад Telegram, Facebook, Instagram тощо. Варто зазначити, що деякі з перелічених соціальних платформ мають інтегрований функціонал для створення опитувань, приклад наведено на рисунку (рис. 1.1 – рис. 1.2), але навіть ці рішення не надають задовільного функціонала розповсюдження. В месенджері Telegram наприклад, опитування спрямовані на локальні групи користувачів, які можуть бути об’єднані тематично. В соціальному месенджері Facebook функціонал дещо ширший і дозволяє створювати опитування для аудиторії, що підписана на вас, але все ще не кожна людина з усіх ваших підписників стане активним респондентом, адже платформа більше спрямована на спілкування, ніж на аналіз та розповсюдження опитувань (рис. 1.3).

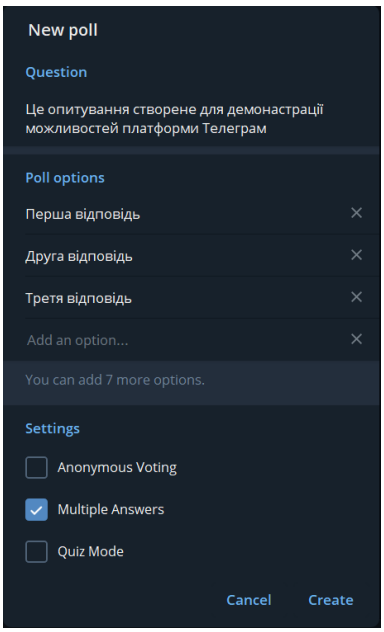


Рис. 1.1. Форма для створення опитувань в Telegram

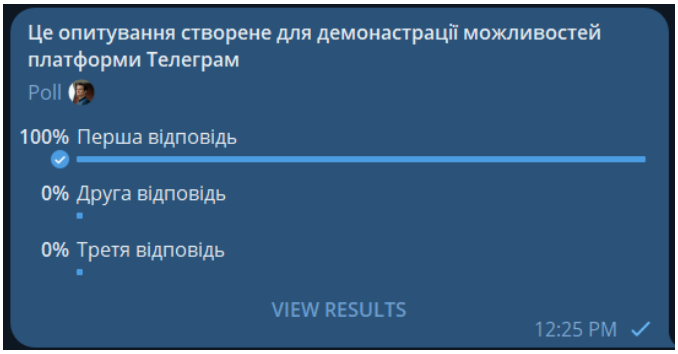


Рис. 1.2. Форма відображення відповідей на опитування в Telegram

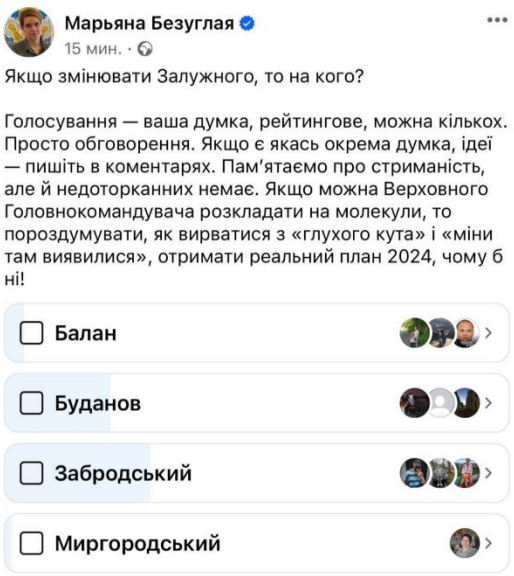


Рис. 1.3. Форма відображення відповідей на опитування в Facebook

1.2. Призначення розробки та галузь застосування

Результатом аналізу поточного стану платформ, що задовольняють потреби користувачів в розповсюдженні, створені та аналізі опитувань стає такий висновок: інтернет користувачі, які хочуть стати кореспондентами, не мають повноцінної платформи, де була б зручна система пошуку респондентів, створення опитувань та їх аналізу. В якості об'єкту розробки розглядається незалежна вебплатформа, основна ідея якої створення, аналіз та вільне розповсюдження опитувань серед різних соціальних груп користувачів.

Дана платформа має містити функціонал для створення опитувань, редагування опитувань, перегляду опитувань.

В результаті плану, буде створена платформа, де люди зможуть вільно публікувати та поширювати опитування. Основними користувачами передбачаються люди віком від шістнадцяти до п’ятдесяти років, які мають доступ до інтернету. Основними соціальними прошарками, які будуть використовувати платформу на перших етапах життя платформи мають стати студенти, в тому числі студенти НТУ «Дніпровська Політехніка», викладачі вищих учбових закладів, школярі тощо. Варто зазначити, що платформа має дати змогу не тільки аналізувати серйозні соціологічні опитування, а й просто весело проводити свій вільний час. Опитування не обов’язково завжди будуть мати в своїй суті важливі соціальні теми та питання з їх приводу, оскільки платформа вільна для використання, передбачається, що здебільшого там можуть бути присутні опитування на звичайні побутові теми, теми рекламного характеру та теми, які виникають через бажання кореспондентів дізнатися думку респондентів з приводу питань особистого характеру. Звісно весь процес має відбуватись відповідно до правил використання платформи. Вирішено дати назву платформі «Survey», що в перекладі з англійської мови – опитування. Далі в роботі замість «вебплатформа» може бути використана саме її назва.

1.3. Підстави для розробки

Підставами для розробки та виконання кваліфікаційної роботи) є:

− освітня програма 122 Комп’ютерні науки;

− навчальний план та графік навчального процесу;

− наказ ректора Національного технічного університету «Дніпровська політехніка» № 469-с від 23.05.2024 р;

− завдання на кваліфікаційну роботу на тему «Розробка платформи для створення публічних опитувань з використанням технології Vue.js та ASP.NET».

1.4. Постановка завдання

Завдання: розробити серверну та клієнтську частину вебплатформи для створення, редагування та аналізу публічних опитувань. Створити веб сторінки реєстрації; аутентифікації; створення, редагування та перегляду опитування; перегляду всіх опитувань, менеджменту всіх створених опитувань користувача.

Функціонал:

У табл. 1.1 наведені сутності користувачів, які будуть використовуватись в системі «Survey».

Таблиця 1.1

Сутності користувачів, які взаємодіють з платформою

Тип користувача	Доступний функціонал
Анонімний	Перегляд опитувань, експорт опитувань, аналіз опитувань, реєстрація
Звичайний	Авторизація та аутентифікація, створення опитувань, редагування опитувань, перегляд опитувань, експорт опитувань, аналіз опитувань, створення скарги
Адміністратор	Видалення опитувань, блокування користувачів, перегляд загальної особистої інформації

У табл. 1.2 наведені сутності, якими буде керувати система під час своєї роботи. Також на основі цих об’єктів буде створена бібліотека класів під час подальших етапів розробки вебплатформи «Survey».

Таблиця 1.2

Сутності які будуть використовуватись у системі

Сутність	Властивості
Користувач	Ім’я, пошта, дата народження, гендер, витрати на місяць, додаткова інформація в текстовому вигляді, сімейний стан, аватар у форматі картинки, країна проживання, чи активний користувач
Опитування	Назва, опис, чи можливо проголосувати за декілька відповідей, чи можливо переголосувати, чи приватне опитування, автор
Опція опитування	Назва, зв’язок з опитуванням до якого відноситься
Голос опитування	Зв’язок з опцією опитування, зв’язок з голосом користувача
Скарга	Зв’язок з користувачем що створив скаргу, зв’язок з опитуванням на яке поскаржились, коментар, загальна причина скарги
Медіа матеріали	Назва, тип медіа матеріалу, користувач якому належить
Робота користувача	Назва, опис
Освіта користувача	Назва, опис, ступінь освіти
Хобі користувача	Назва, опис
Країна користувача	Назва, код країни

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Вебплатформа «Survey» повинна бути реалізована у вигляді клієнтської та серверної частини. Серверна частина має бути реалізована мовою C# з використанням ASP.NET Core [2], а для клієнтської частини має використовуватися VUE.JS [3] та мова програмування JS. Веб сторінки будуть використовувати стандартний HTML та CSS для стилізації. Взаємодія клієнтської та серверної частин має відбуватися за допомогою HTTP запитів (POST, PUT, PATCH, DELETE, GET). Буде використана бібліотека «axios» на стороні клієнтської частини для створення запитів до сервера. Під час розробки системи, ми будемо керуватися архітектурним стилем REST [4], щоб в подальшому всі частини нашої платформи легко обмінювались даними за схожими шаблонами. Серверна частина повинна бути реалізована за Clean architecture [5], що дозволить в подальшому розширювати функціонал нашого застосунку та зробить зв’язок між нашими даними, та реалізацією взаємодії з ними слабо зв’язаним та більш гнучким. На серверному рівні для взаємодії запитів від клієнтської частини між бізнес сервісами, буде використано патерн MediatR [6], таким чином команди від клієнтської частини, будуть слабо зв’язані з їх кінцевою реалізацією, що сприяє якісному та структурованому розширенню системи у подальшому. У якості РБД буде використано MS SQL, для взаємодії сервера та РБД використати EF, що дозволить взаємодіяти з нашою базою як з класом на серверному рівні. Для роботи з EF [7] використати LINQ [8]. Буде реалізовано патерн repository (репозиторій) [9] для взаємодії сервісів з EF, щоб відокремити та слабо зв’язати роботу з даними, прив’язавши все до абстракції.

1.5.2. Вимоги до інформаційної безпеки

Варто зазначити, що платформа має передбачити верифікацію користувачів під час реєстрації, платформа має реалізовувати функціонал для збору основної особистої інформації респондентів за для подальшого її використання під часу збору результатів опитування, також має бути реалізована можливість користувачів заборонити використовувати їх особисту інформацію під час збору результатів опитування, та аналізу результатів опитувань. Користувач буде мати змогу зареєструватися з мінімальною кількістю потрібної інформації (пароль, пошта, ім’я та дата народження). Пароль має бути збережений у захешованому вигляді з використанням криптографічної солі, за для запобіганню перебору захешованих паролів у випадку, якщо база потрапить до рук зловмисників.

Важливо відмітити, що платформа має реалізовувати повноцінну систему аутентифікації та авторизації користувачів, за для того, щоб під час активної фази опитування респондентів анонімні користувачі не мали змоги проголосувати, що забезпечить впевненість в реальності респондентів, тобто що відповіді дають реальні користувачі.

Для реалізації функціоналу авторизації та аутентифікації буде використано JWT [10] та Bearer auth [11]. Даний підхід дасть змогу аутентифікувати користувача, що робить запит з клієнтської частини за HTTP заголовком, в який буде поміщено JWT. Сам же JWT буде отримуватися під час входу в кабінет користувача, сервер буде відправляти його клієнтській частині, а клієнтська частина буде його зберігати в локальному середовищі браузера, за для подальшого використання.

Оскільки лише верифіковані та аутентифіковані користувачі будуть мати змогу створювати опитування та голосувати в інших опитувань, має бути передбачена можливість створення скарги щодо створеного контенту, якщо наприклад опитування містить неприпустимий контент, який не має знаходитись

у вільному доступі на подібній відкритій платформі. Подібні скарги мають зберігатись окремо, і доступ до них має отримувати лише адміністратор.

Після аналізу скарги, адміністратор буде мати можливість видалити опитування, тимчасово приховати його, або навіть забанити користувача, який порушив правила використання платформи.

Оскільки платформа буде відкритою для перегляду та аналізу даних для всіх користувачів, в тому числі для анонімних, треба додати функціонал пошуку опитувань та їх детальний перегляд. Варто зазначити, що для авторизованих користувачів особливо важливою складовою платформи має стати менеджмент власних опитувань, тому треба розробити зручну систему пошуку, редагування та видалення тих опитувань, які вже були створені. Для верифікації нових користувачів найпростішим та найефективнішим способом є використання їх особистої пошти, відповідно, користувачі, що верифікували свою пошту під час реєстрації, у випадку порушення правил створення опитувань, будуть не просто видалені з платформи, а заблоковані за поштою, що унеможливить нову реєстрацію з використанням заблокованої пошти.

Варто зауважити, що весь процес розробки, всі зміни, які будуть вноситись у проєкт, будуть також збережені у публічний Github репозиторій, де можна буде відстежити цикл розробки платформи «Survey». Цей підхід дасть змогу відстежувати зміни, що були внесені під час розробки платформи, а також дасть змогу зацікавленим користувачам передивитись відкритий код платформи та впевнитись в безпечності сервісу, або навіть приєднатись до розробки платформи в майбутньому. Варто зазначити, що на платформі Github буде розміщено два публічних репозиторії, де перший репозиторій – це збережений код для клієнтської частини вебдодатку, а другий – це серверна частина веб додатку. Для того щоб запустити проєкт локально, треба працювати з відповідними версіями клієнтської та серверної частин, адже розробка цих двох частин буде вестись паралельно.

1.5.3. Вимоги до складу та параметрів технічних засобів

На платформу «Survey» можна буде зайти з будь-якого браузера та будь-якого пристрою. У ході виконання технічного завдання буде досягнута адаптивність нашого сайту, що дозволить налаштувати вигляд веб сторінок окремо під кожний пристрій: телефон, планшет, комп'ютер, ноутбук тощо. Варто зазначити, що оскільки мова йде про створення вебплатформи, то і працювати вона буде лише у браузері, тому окрім наявності вищеперерахованих пристроїв, користувач має встановити один з перелічених браузерів, для того щоб мати доступ в інтернет середовище та зокрема до платформи «Survey»:

- Google Chrome;
- Apple Safari;
- Mozilla Firefox;
- Microsoft Edge;
- Samsung Internet Browser;
- Opera;
- UC Browser;
- Internet Explorer;
- Brave;
- Yandex Browser.

Варто зазначити, що не всі браузери можна встановити на операційну систему Windows, Android, IOS чи MacOS, тому рекомендовано ознайомитись з можливими варіантами саме для вашого типу пристрою та операційної системи. Між різними браузерами можуть спостерігатись відмінності під час роботи з веб застосунком, але основний функціонал буде працювати відповідно вказаним вище вимогам.

1.5.4. Вимоги до інформаційної та програмної сумісності

Для технічного тестування рекомендовано користуватися застосунками з наступними технічними характеристиками:

– Версія браузера Chrome 81.0.4044.20 або вище: Використання оновленої версії браузера допоможе забезпечити сумісність та оптимальну продуктивність при тестуванні.

– Підтримка 32- або 64-розрядного процесора та операційної системи: Застосунки повинні бути сумісні з обома типами процесорів та операційних систем для максимального охоплення користувачів.

– Операційна система Windows 10 або 11: Використання останніх версій операційної системи допоможе уникнути проблем з сумісністю та забезпечить стабільну роботу застосунків.

– Наявність маніпулятора-міші/тачпаду та клавіатури: Це необхідно для забезпечення повноцінного взаємодії з застосунком та проведення різних видів тестувань.

– Оперативна пам'ять об'ємом 4 ГБ або вище: Достатня кількість оперативної пам'яті допоможе забезпечити швидку та ефективну роботу застосунків під час тестування.

– Процесор із частотою 2 ГГц або вище: Потужний процесор забезпечить швидку обробку даних та відмінну продуктивність під час тестування.

– Жорсткий диск HDD чи SSD об'ємом 20 ГБ або вище: Достатній обсяг дискового простору дозволить зберігати дані тестування та забезпечить зручний доступ до них.

Для використання застосунку в браузері телефона, рекомендовано наступні технічні параметри:

– Версія браузера Chrome для IOS 124.0.6367.111, для Android 124.0.6367.113: Використання оновленої версії браузера забезпечить сумісність та оптимальну продуктивність застосунку на мобільних пристроях.

– Розмір екрана 2436 пікселей у висоту та 1125 пікселей у ширину або 6,06 дюймів: Застосунок повинен бути оптимізований під розмір екрана мобільного пристрою для зручного користування.

– Операційна система IOS: 15.4.1 або Android 13.

– Наявність робочого сенсорного екрану: Це необхідно для забезпечення взаємодії з застосунком на сенсорних пристроях.

– Оперативна пам'ять об'ємом 4 ГБ: Достатня кількість оперативної пам'яті забезпечить швидку реакцію застосунку та плавну роботу на мобільних пристроях.

– Процесор із частотою 2 ГГц або вище.

– Жорсткий диск HDD чи SSD об'ємом 16 ГБ або вище.

Для використання застосунку в браузері планшета, рекомендовано наступні технічні параметри:

– Версія браузера Chrome для IOS 135.0.6497.100, для Android 135.0.6497.105;

– Розмір екрана 2560 пікселей у висоту та 1600 пікселів у ширину або 10.5 дюймів;

– Операційна система IOS: 16.2 або Android 14;

– Наявність робочого сенсорного екрану;

– Оперативна пам'ять об'ємом 6 ГБ;

– Процесор із частотою 2.5 ГГц або вище;

– Жорсткий диск SSD об'ємом 32 ГБ або вище.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

У ході виконання кваліфікаційної роботи буде створено відкриту вебплатформу для створення, аналізу та редагування опитувань, користувачами, що зареєструвалися на платформі. Метою створення цієї платформи є надання можливості інтернет користувачам ставати кореспондентами та отримувати доступ до розповсюдження власних опитувань серед респондентів, тобто інших користувачів сайту. Передбачено, що користувачі, які не є зареєстрованими на нашій платформі все одно будуть мати змогу переглядати та аналізувати опитування, але створювати опитування чи голосувати в них вони не зможуть. Також залогінені користувачі будуть мати змогу поскаржитися на інші опитування, якщо їх контент не буде відповідати правилам наповнення опитувань на нашій платформі. Подібні скарги зможе обробляти адміністратор, та відповідно до них блокувати користувачів, або видаляти опитування.

2.2. Опис застосованих математичних методів

Під час виконання кваліфікаційної роботи будуть операції над множинами під час взаємодії з таблицями нашої РБД. Незважаючи на реалізацію цих операцій за допомогою вбудованих конструкцій .Net, все одно всі вони будуть опиратися на такі базові дії над множинами як перетин (рис 2.1.), об’єднання (рис 2.2) та різниця (рис. 2.3). На основі цих базових принципів будується зв’язки між таблицями у РБД. По суті ці операції є базою будь-якої РБД. На рисунках 2.1. – 2.3. відображені ці дії з множинами на прикладі двох множин А і Б, де заштрихований вміст множин – це множина, яка є результатом виконання вказаної операції.

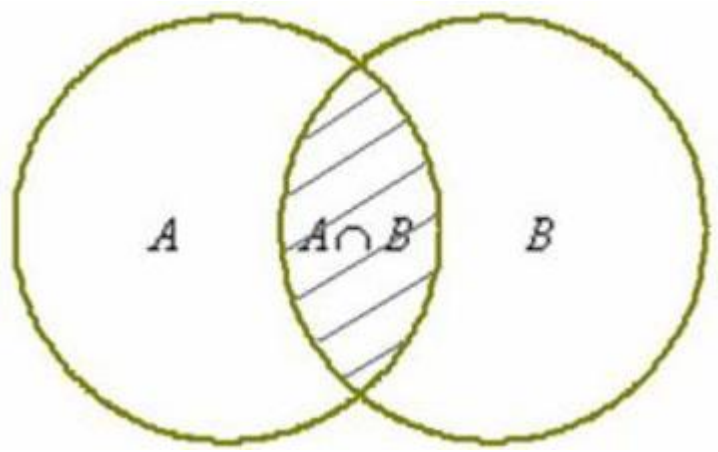


Рис. 2.1. Перетин множин. Діаграма Венна

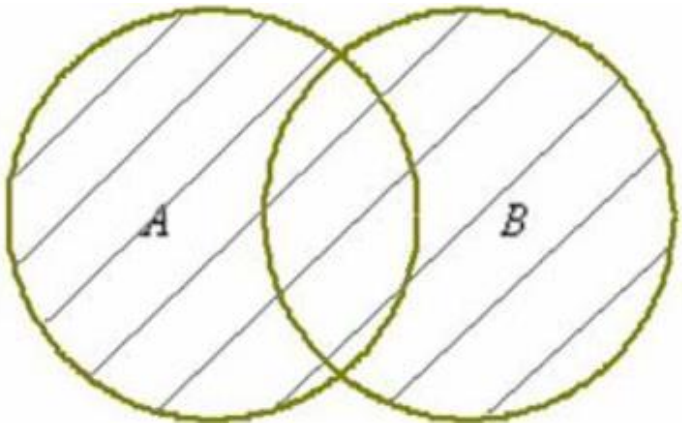


Рис. 2.2. Об’єднання множин. Діаграма Венна

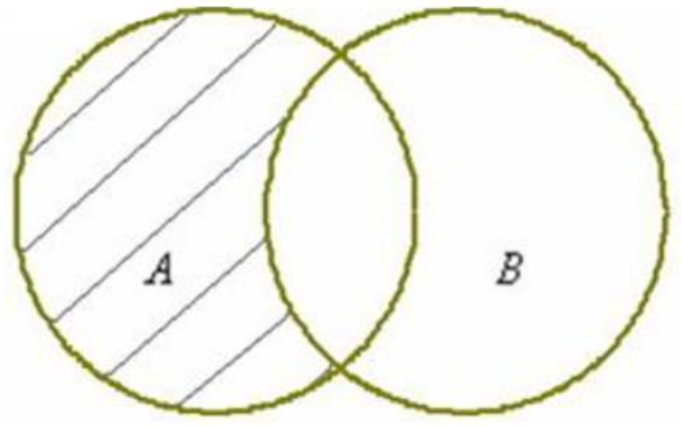


Рис. 2.3. Різниця множин. Діаграма Венна

2.3. Опис використаних технологій та мов програмування

Наш веб додаток складається з серверної частини та клієнтської. Серверна частина реалізована з використанням ASP.NET Core, та представляє собою RESTful [12] сервіс. CORS налаштування повинні бути сконфігуровані таким чином, щоб до API нашого ресурсу міг дістатися будь-який користувач, тобто не тільки клієнтська частина зможе робити запити до нашого сервера. Таким чином наше API буде відкритим до майбутніх потенційних інтеграцій. Для написання серверної частини буде використовуватися мова програмування C#. Клієнтська частина буде працювати на VUE.JS. Для полегшення розробки дизайну нашої платформи, було прийнято рішення звернутися до безкоштовної готової бібліотеки з вже описаними базовими компонентами застосунку, такими як таблиці, поля вводу даних, діаграми тощо. З поміж доступних варіантів після ретельного тестування було обрано макет для адміністративної панелі «Sakai» [13]. Основною перевагою використання готових стилів те макетів є збереження приблизно половини часу на розробку, адже в готових макетах завжди передбачений потрібний функціонал, якого достатньо для створення базового сайту. Таким чином в «Sakai» вже присутній готовий функціонал для роботи з діаграмами (рис 2.4.), макети форм користувача для реєстрації та аутентифікації (рис 2.5.), а також описана логіка відображення таблиць (рис 2.6.).

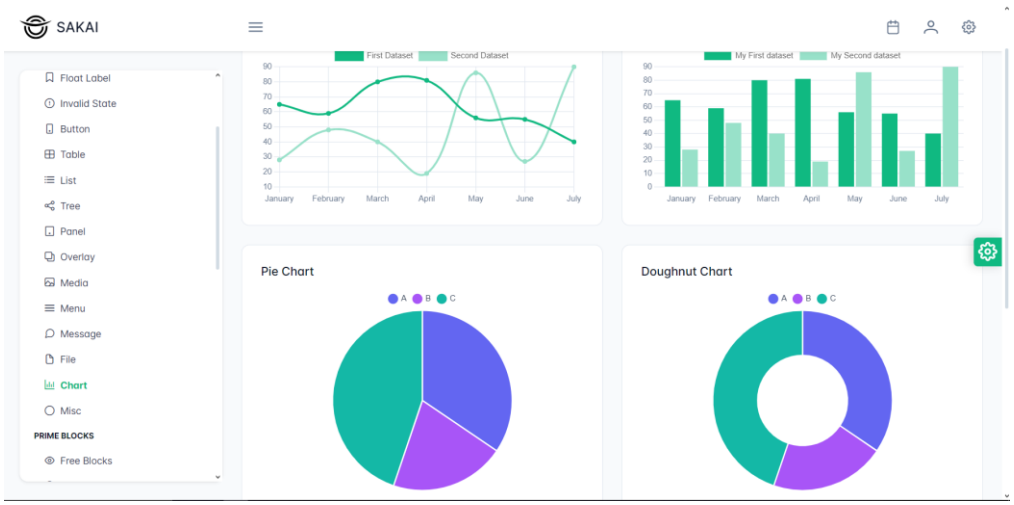


Рис. 2.4. Приклади використання діаграм в макеті «Sakai»

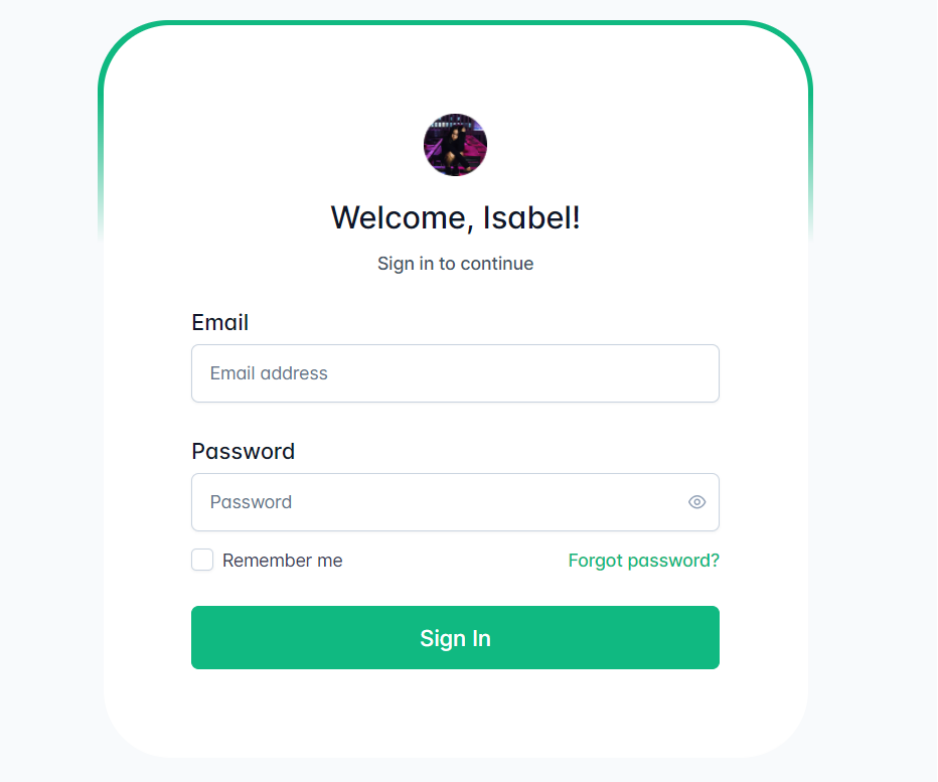


Рис. 2.5. Приклади форми реєстрації в макеті «Sakai»

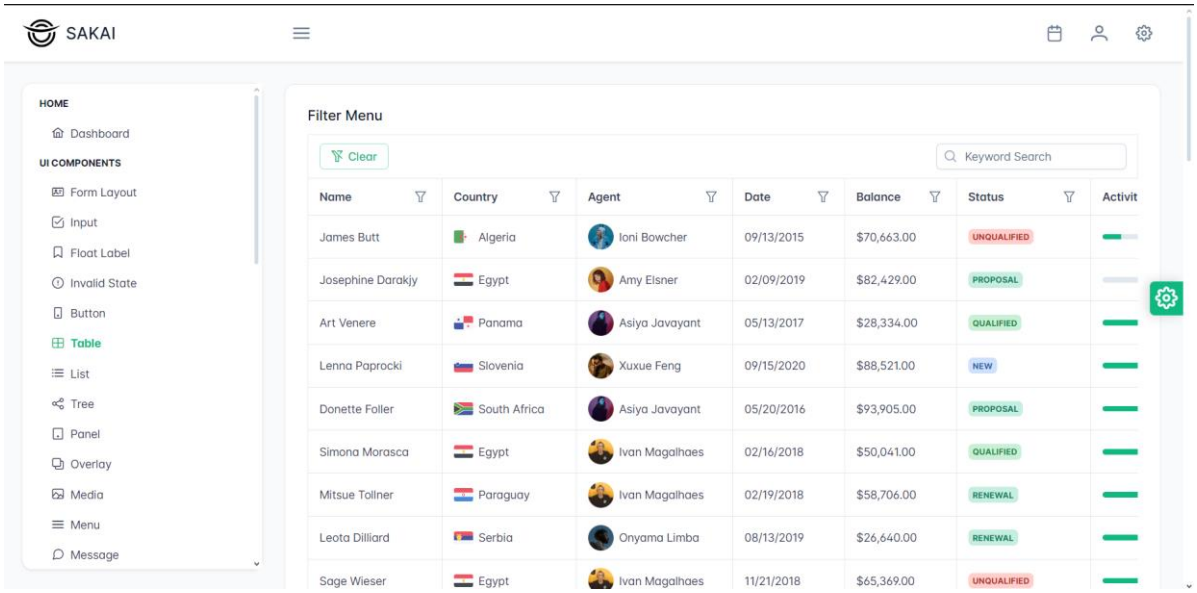


Рис. 2.5. Приклади використання таблиці в макеті «Sakai»

Для тестування нашої серверної частини буде використано такий інструмент як Swagger. Swagger – це розширення для створення документації та візуалізації кінцевих точок серверної частини, що дозволить нам взаємодіяти із сервером навіть без клієнтської частини, за допомогою згенерованих на основі кінцевих точок формах (рис. 2.6. – рис. 2.7.).

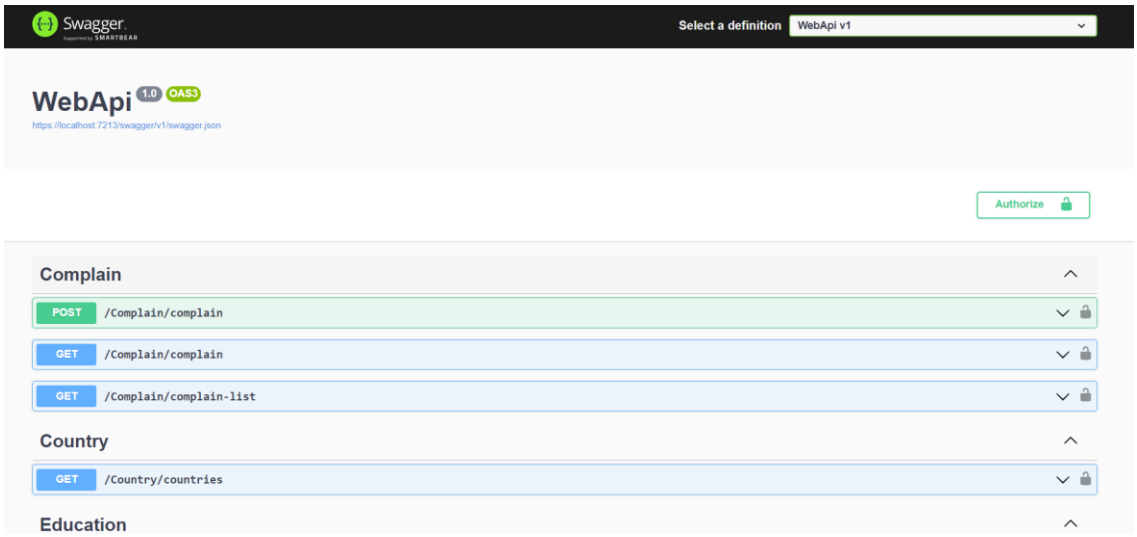


Рис. 2.6. Приклад згенерованої документації від Swagger, що описує серверні кінцеві точки

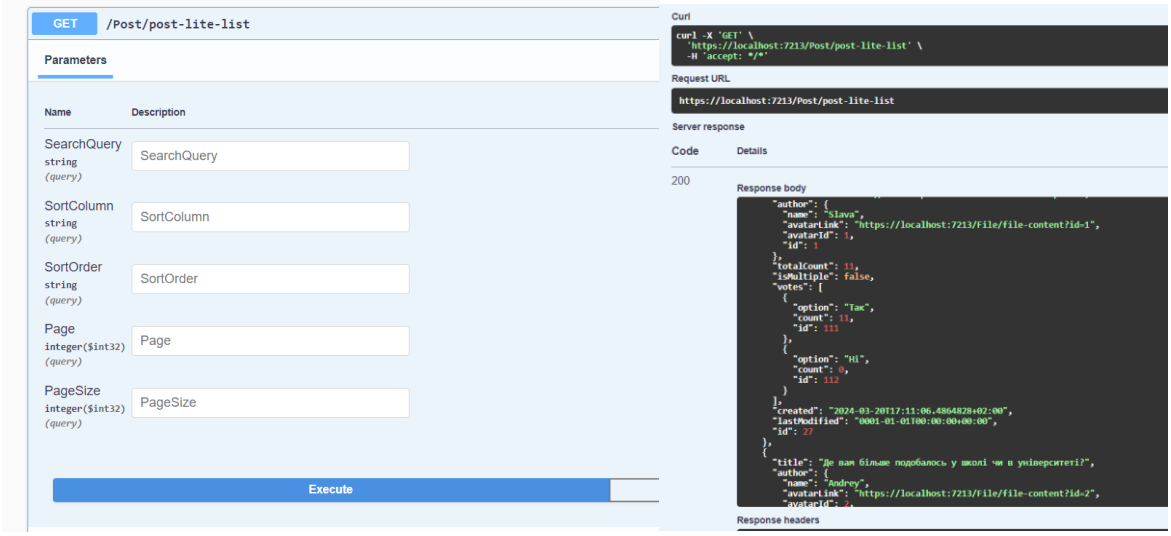


Рис. 2.7. Приклад згенерованої документації від Swagger, що описує взаємодію з кінцевими точками сервера, та їх відповідь

2.4. Опис структури системи та алгоритмів її функціонування

Повертаючись до описання архітектури серверної частини, варто пояснити яким саме чином буде досягнений слабкий зв'язок між компонентами нашого застосунку, та яку роль в цьому відіграє Clean architecture, CQRS, MediatR та патерн репозиторій. В цілому, набір цих архітектурних рішень та патернів, дає змогу пов'язати прикладний (найвищий рівень сервера) до абстракції, реалізація якої може мінятися без впливу на прикладний рівень, іншими словами прикладний рівень нічого не знає про кінцеву реалізацію свого функціонала, бо він звертається лише до абстракції. Детальна візуалізація роботи подібного підходу описана на рис 2.8.

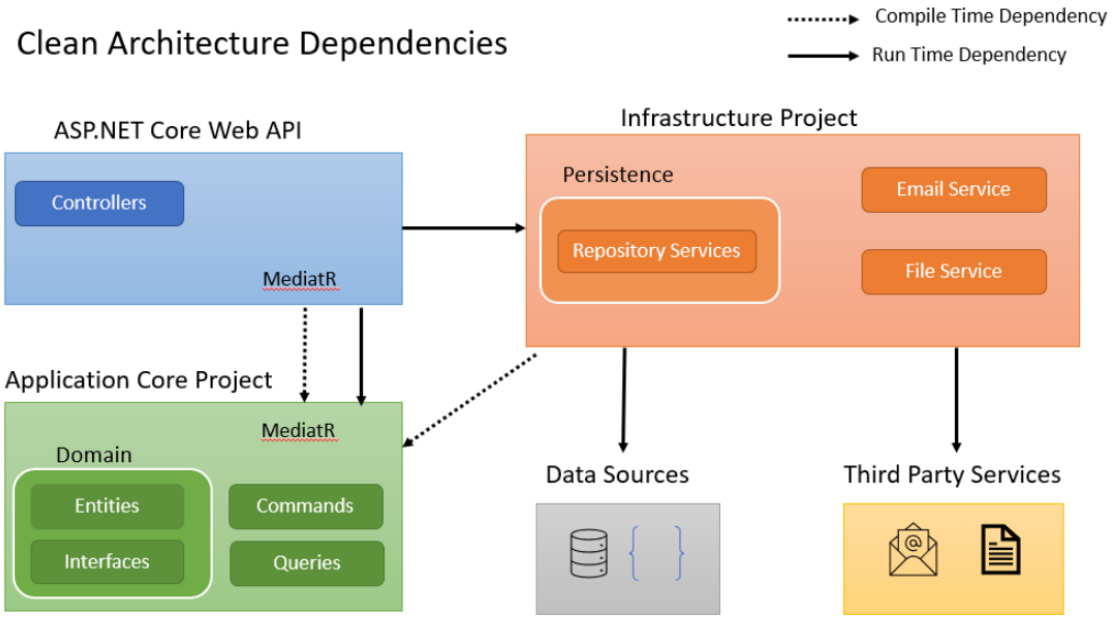


Рис. 2.8. Приклад використання Clean architecture, CQRS, MediatR та патерну репозиторію з ASP.NET Core [14]

Важливо підмітити, що на етапі проєктування та розробки прикладний рівень (ASP.NET Core Web API) звертається до рівня застосунку (Application Core Project), і вже потім ми вказуємо реалізацію абстракції в інфраструктурі (Infrastructure Project).

Коли програма компілюється, тоді абстрактні залежності напряму зв’язуються зі своєю реалізацією. Інфраструктура в свою чергу може взаємодіяти з нашою РБД та з третіми сервісами.

MediatR використовується для посилання команд та запитів, де команди – це виклик, що змінює дані в РБД в результаті виконання, а запити використовуються для читання даних без їх змін. Команди та запити викликаються на прикладному рівні сервера (ASP.NET Core Web API) (рис. 2.9.), а їх виконання відбувається на рівні застосунку (Application core) (рис. 2.10.). В C# для реалізації даного підходу існує бібліотека MediatR

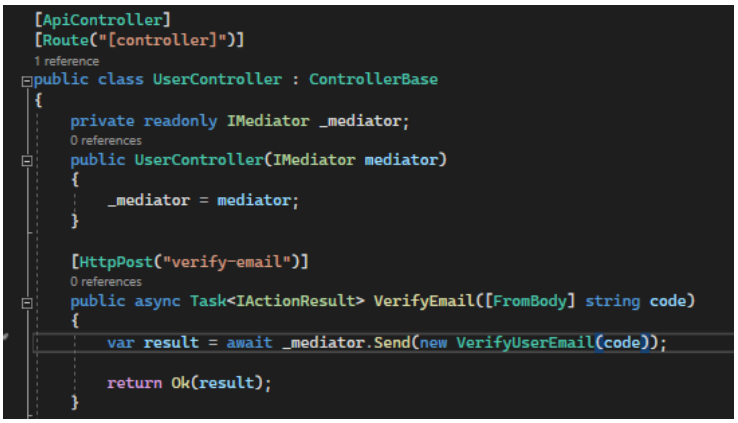


Рис. 2.9. Приклад вхідної точки для верифікації пошти користувача. В методі VerifyEmail викликається команда VerifyUserEmail.

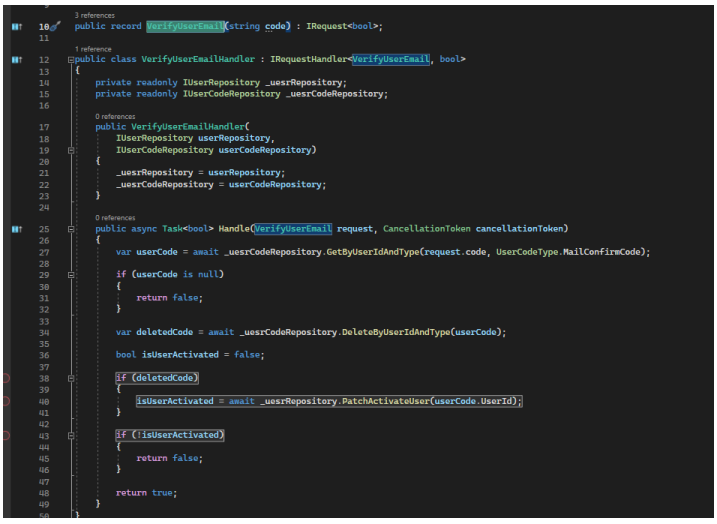


Рис. 2.10. Команда для верифікації пошти користувача та метод «Handle», який відпрацює при виклику цієї команди (рис.2.9.).

Варто відзначити, що кожен шар у Clean architecture відповідає створеній бібліотеці класів в рішенні нашого проєкта. Кінцевий архітектурний вигляд нашого застосунку зображено на рис. 2.11

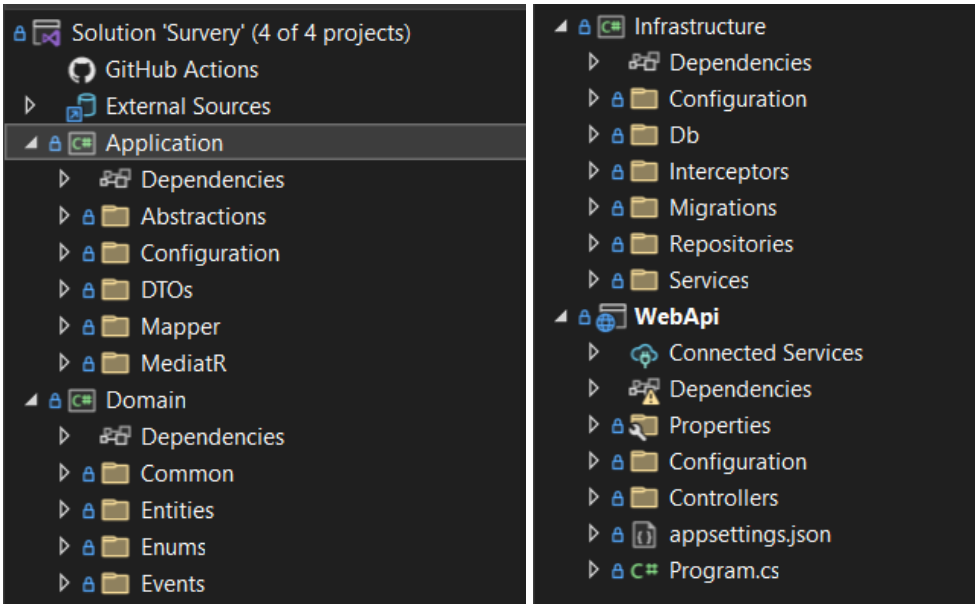


Рис. 2.11. Архітектура сервера у вигляді каталогів рішення

Одним із важливих аспектів роботи серверної частини є взаємодія з РБД. Як вже було зазначено раніше, для створення бази даних та взаємодії з нею, буде використано Entity Framework (EF). Важливо відмітити, що існує два способи розробки системи за допомогою цього інструменту. Перший спосіб – це так званий «code first», коли спочатку пишеться код, описуються класи та взаємодія між ними як між таблицями, а вже потім на основі цих класів розгортається база. Другий спосіб – це створення класів на основі таблиць на стороні сервера, цей варіант більш підходить під налаштування EF для вже існуючої бази, яку досить довгий час розробляли класичним способом. Оскільки наша база ще не створена і буде розроблятися разом с серверною та клієнтською частиною, то буде обраний перший спосіб розробки.

В наступному розділі детально буде описана саме ієрархія класів та діаграма таблиць у РБД, зараз же варто навести принцип роботи EF на нашому сервері. По-перше, для інтеграції EF треба створити так званий контекст бази даних – клас, який описує нашу РБД загалом, включно зі зв'язками та таблицями, які ми будемо використовувати (рис. 2.12).

```
13 references
public class ApplicationDbContext : DbContext, IApplicationDbContext
{
    0 references
    public ApplicationDbContext() { }

    0 references
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options) { }

    10 references
    public virtual DbSet<User> Users => Set<User>();
    14 references
    public virtual DbSet<Post> Posts => Set<Post>();
    2 references
    public virtual DbSet<PoolOption> PoolOptions => Set<PoolOption>();
    11 references
    public virtual DbSet<Vote> Votes => Set<Vote>();
    3 references
    public virtual DbSet<Media> Files => Set<Media>();
    2 references
    public virtual DbSet<Education> Educations => Set<Education>();
    2 references
    public virtual DbSet<Job> Jobs => Set<Job>();
    2 references
    public virtual DbSet<Hobby> Hobbies => Set<Hobby>();
    2 references
    public virtual DbSet<Country> Countries => Set<Country>();
    1 reference
    public virtual DbSet<UserJob> ProfileJobs => Set<UserJob>();
    1 reference
    public virtual DbSet<UserHobby> ProfileHobbies => Set<UserHobby>();
    1 reference
    public virtual DbSet<UserEducation> ProfileEducations => Set<UserEducation>();
    4 references
    public virtual DbSet<Complain> Complains => Set<Complain>();
    4 references
    public virtual DbSet<SavedPost> SavedPosts => Set<SavedPost>();
    4 references
    public virtual DbSet<UserCode> UserCodes => Set<UserCode>();
    0 references
    protected override void OnModelCreating(ModelBuilder modelBuilder)
```

Рис. 2.12. Опис контексту бази даних

На рис. 2.12. ми можемо побачити основні складові контексту бази даних, а саме це те, що клас унаслідується від базового класу DbContext (він описаний в бібліотеці EF), та має конструктор, що приймає параметри налаштувань. Всі властивості, що починаються з DbSet – це визначення наших таблиць у базі даних, що приймають тип класу, який наприклад описує користувача чи опитування, а потім під час під'єднання, до цієї змінної можна буде звертатися як до таблиці в РБД. Такий підхід не тільки дозволяє в майбутньому легко взаємодіяти з даними, але і чітко знати як виглядає наша БД навіть у випадку, коли до самої БД ми прямого доступу не маємо

Важливо також описати механізм за яким ми будемо будувати зв’язки між нашими таблицями, для цього буде використано Fluent API для EF [15]. Ця бібліотека використовується для того, щоб встановлювати зв’язки між нашими сутностями в РБД, використовуючи EF та контекст бази даних. На рис. 2.13. зображено приклад, де в метод ConfigurePost винесена конфігурація сутності опитувань в нашій системі (Post або іноді Pool). На рис 2.13. також можна побачити, що опитування у нас обов’язково буде мати Id автора, тобто зв’язок з ним один до одного; також в самого опитування є первинний ключ Id. В поста може бути багато варіантів відповідей (Options). Також важливою складовою такої конфігурації є налаштування поведінки залежних сутностей під час видалення чи зміни опитування, таким чином на рис 2.13. на 68 стрічці коду, описана логіка каскадного видалення, тобто якщо видалити опитування, то видаляться і всі питання до нього, бо питання без самого опитування існувати не може. По аналогії знизу можна побачити налаштування зв’язку скарг на опитування

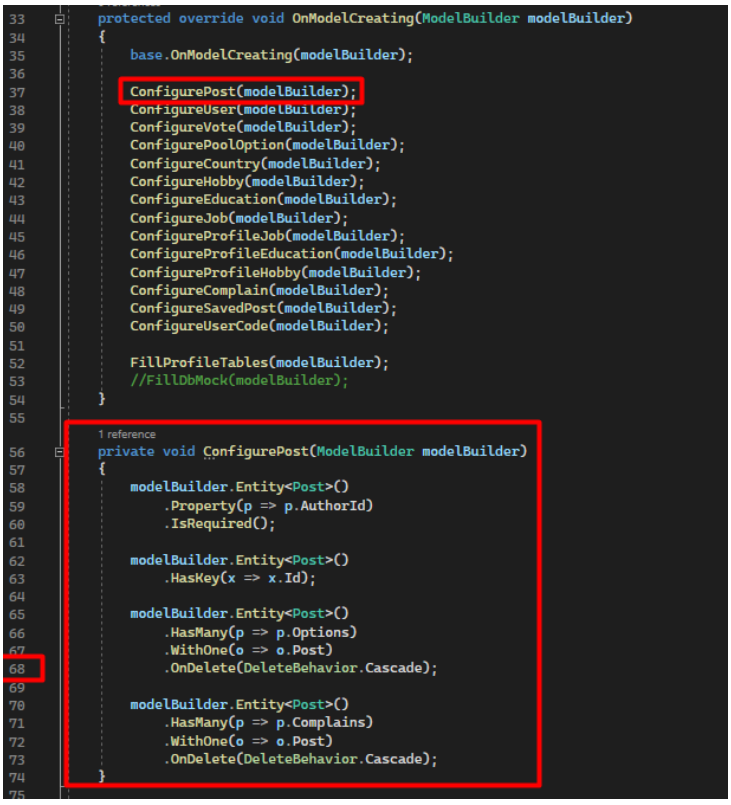
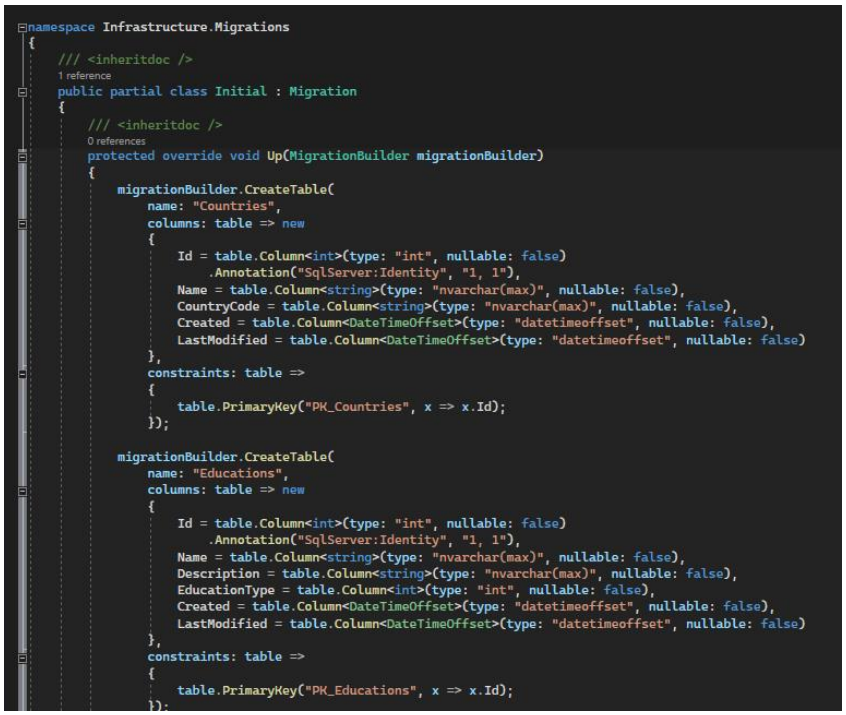


Рис. 2.13. Приклад конфігурації сутності опитування для РБД

Для того, щоб всі наші зв’язки та описані таблиці були створені у вигляді РБД, нам треба створити міграцію [16]. Міграція – це опис конструкції бази даних, створеної на основі раніше визначеного контексту. По суті в ASP.NET міграція представляє собою клас з двома методами «Up» і «Down». Метод «Up» описує той стан, якого набуде БД в наслідок виклику цієї міграції. Метод «Down» описує логіку для відкочування БД на попередню міграцію (рис. 2.14.). Внаслідок розробки системи виникають нові сутності, на основі яких буде робитися нова міграції, всі ці міграції збережено на рівні інфраструктури як і сам контекст бази даних (рис 2.15.)



```
namespace Infrastructure.Migrations
{
    /// <inheritdoc />
    1 reference
    public partial class Initial : Migration
    {
        /// <inheritdoc />
        0 references
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Countries",
                columns: table => new
                {
                    Id = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    Name = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    CountryCode = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    Created = table.Column<DateTimeOffset>(type: "datetimeoffset", nullable: false),
                    LastModified = table.Column<DateTimeOffset>(type: "datetimeoffset", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Countries", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "Educations",
                columns: table => new
                {
                    Id = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    Name = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    Description = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    EducationType = table.Column<int>(type: "int", nullable: false),
                    Created = table.Column<DateTimeOffset>(type: "datetimeoffset", nullable: false),
                    LastModified = table.Column<DateTimeOffset>(type: "datetimeoffset", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Educations", x => x.Id);
                });
```

Рис. 2.14. Приклад вмісту файлу міграції

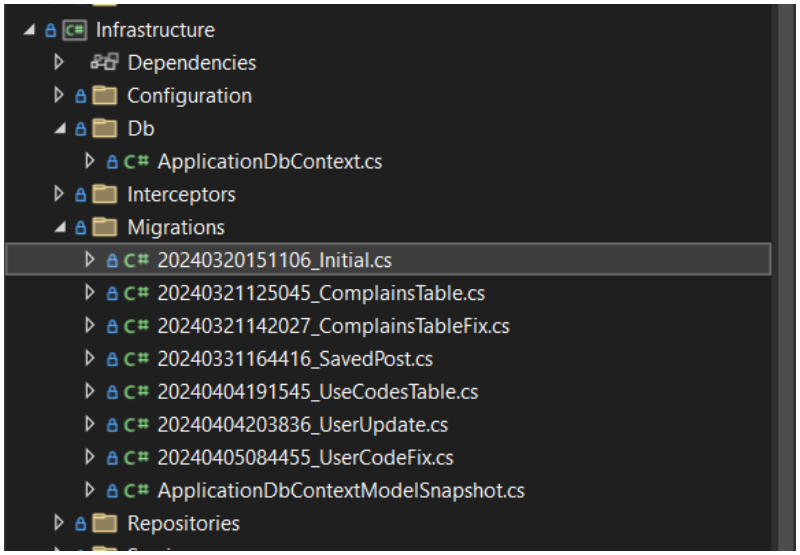


Рис. 2.15. Міграції, що були створені під час написання кваліфікаційної роботи, «Initial» – перша міграція, f «UserCodeFix» – остання, тобто актуальна

Щоб створити міграцію треба перейти за вкладинкою Tools, NuGet package manager, package manager console та прописати команду «Add-migration -Name», а щоб виконати міграцію там же в консолі треба написати «Update-database».

Також спеціально для тестування системи, було додано метод, який заповнює базу даних тестовими даними (mock data) (рис. 2.16.). В цьому методі є алгоритм який випадковим чином генерує відповіді на завчасно згенеровані опитування, що імітує реальну роботу системи, потім ці відповіді зберігаються в БД (рис. 2.17.). Цей метод має викликатися раз при накочувані всіх міграцій, а потім має бути закоментований, бо для того, щоб він відпрацював треба декілька хвилин. Варто зауважити, що створення тестових даних, досить громістка та однотипна робота, тому для генерації даних був використаний Chat GPT третьої версії [17].

Останнє, що треба зазначити про EF, це те як конфігурується під'єднання до бази даних. В JSON файлі appsettings.json є рядок DefaultConnection, в якому знаходяться всі дані для підключення до БД (рис. 2.18.).

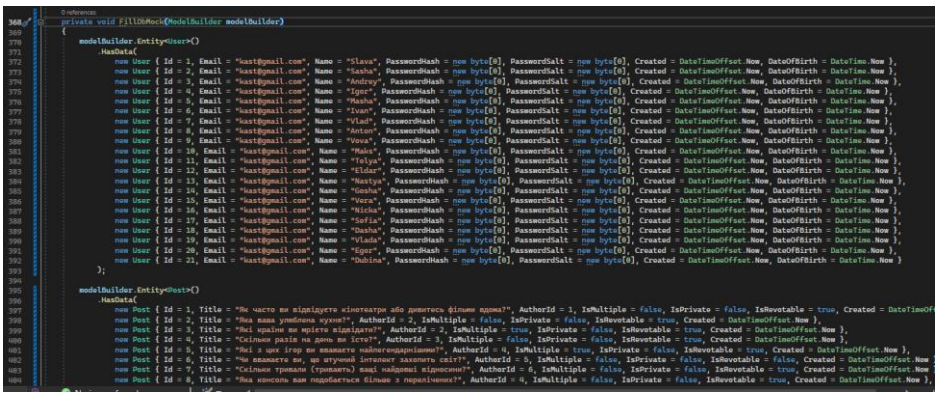


Рис. 2.16. Метод для заповнення в БД тестові дані

```
Random random = new Random();
List<Vote> votes = new();
int id = 0;

for (int i = 1; i < 112; i++)
{
    for (int j = 1; j < 22; j++)
    {
        if (random.Next(1, 10) >= 5)
        {
            id++;
            votes.Add(new Vote { Id = id, UserId = j, PoolOptionId = i, Created = DateTimeOffset.Now });
        }
    }
}

modelBuilder.Entity<Vote>()
    .HasData(votes);
}
```

Рис. 2.17. Алгоритм генерації відповідей на створені завчасно опитування

appsettings.Development.json | **appsettings.json** | Job.cs | launchSettings.json | UserController.cs

Schema: https://json.schemastore.org/appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefualtConnection": "Server=DESKTOP-RURF5OE\\SLAVA; Database=SurveyDb; Trusted_Connection=True; Encrypt=False"
  },
  "Jwt": {
    "Key": "survery test 2023",
    "Issuer": "https://localhost:7214/",
    "Audience": "https://localhost:7214/"
  },
  "Mail": {
    "Sender": "",
    "Password": ""
  }
```

Рис. 2.18. Рядок з даними для підключення до БД

Під час запуску нашого сервера та реєстрації залежностей (рис. 2.19.), з цього файлу вичитуються дані для під'єднання та на їх основі будується з'єднання з БД, в нашому випадку БД і сервер локально розміщені на комп'ютері, але цілком можливий варіант взаємодії сервера з вже задеплоєною базою, особливо під час релізу нашого застосунку.

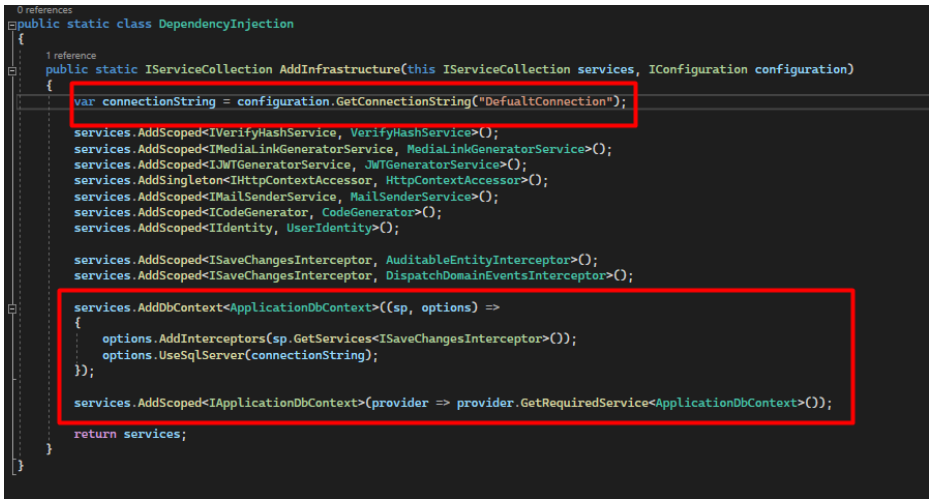


Рис. 2.19. Створення з'єднання з базою даних

Як вже було зазначено, для авторизації та аутентифікації буде використовуватися підхід з використанням JWT. Для того, щоб реалізувати цей підхід, нам треба виконати наступні дії:

1. Створити кінцеву точку, що буде приймати дані користувача, такі як пароль та логін, та віддавати JWT у разі верифікації користувача (рис. 2.20.).
2. Додати конфігурацію Barer auth, де описати як саме наш сервер має валідувати токен (рис. 2.21.).
3. Створити метод для генерації токену, який буде зберігати в токені Id користувача, а також підпис сервера, та від кого цей токен може бути надісланий (рис. 2.22.).
4. До тих кінцевих точок, що можуть бути викликані лише аутентифікованим користувачем додати атрибут «[Authorize]», таким чином .Net автоматично буде валідувати токен за описаною схемою, під час виклику.

```
[HttpPost("sign-in")]

public async Task<IActionResult> SignInUser([FromBody] SignInDTO query)
{
    var result = await _mediator.Send(new SignIn(query));

    return Ok(result);
}
```

Рис. 2.20. Створення кінцевої точки для отримання JWT

```
0 references
public static class AuthenticationExtension
{
    1 reference
    public static void Authentication(this WebApplicationBuilder builder)
    {
        builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
                .AddJwtBearer(options =>
                {

                    options.TokenValidationParameters = new TokenValidationParameters
                    {
                        ValidateActor = true,
                        ValidateLifetime = true,

                        ValidateIssuerSigningKey = true,
                        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8
                        .GetBytes(builder.Configuration["Jwt:Key"])),

                        ValidateIssuer = true,
                        ValidIssuer = builder.Configuration["Jwt:Issuer"],

                        ValidateAudience = true,
                        ValidAudience = builder.Configuration["Jwt:Audience"]
                    };
                });
    }
}
```

Рис. 2.21. Конфігурація Bearer auth

```
2 references
public string GenerateJwtToken(User user)
{
    List<Claim> claims = new()
    {
        new (JwtRegisteredClaimNames.Sid, user.Id.ToString()),
        new (JwtRegisteredClaimNames.Name, user.Name)
    };

    var token = new JwtSecurityToken(
        issuer: _builder.GetSection("Jwt:Issuer").Value,
        audience: _builder.GetSection("Jwt:Audience").Value,
        claims: claims,
        expires: DateTime.Now.AddDays(7),
        notBefore: DateTime.Now,
        signingCredentials: new SigningCredentials(
            new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_builder.GetSection("Jwt:Key").Value)),
            SecurityAlgorithms.HmacSha256)
    );

    var jwtToken = new JwtSecurityTokenHandler().WriteToken(token);

    return jwtToken;
}
}
```

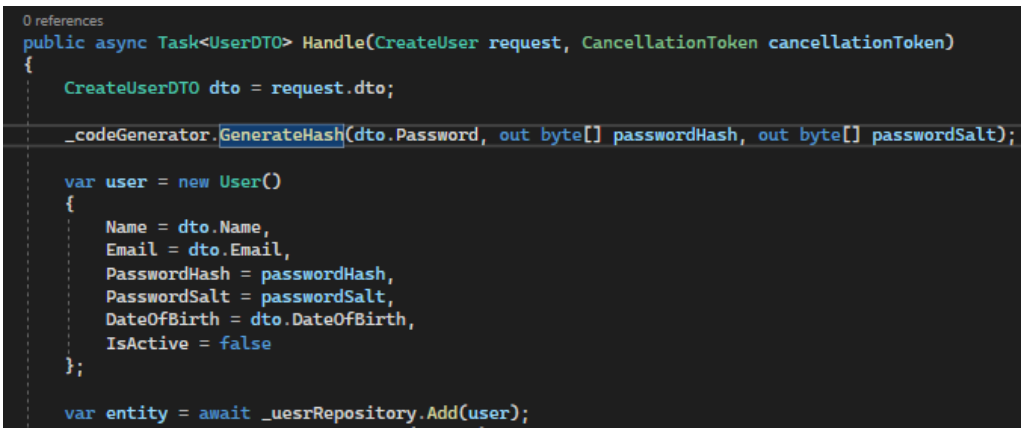
Рис. 2.22. Метод, який буде використовуватися під час виклику команди на аутентифікацію користувача, для видання токену на 7 днів

Важливим аспектом роботи системи є збереження паролю користувача. Для того, щоб зберігати пароль безпечно буде використано алгоритм додавання криптографічної солі до паролю та його подальше хешування і збереження в БД разом із криптографічною сіллю [18]. Це базовий підхід, який дуже надійний і використовується в багатьох компаніях, що розробляють інформаційні системи. По-перше, як можна зрозуміти з опису, ми не зберігаємо пароль у тому вигляді як його прописує користувач, для того, щоб жоден зловмисник не міг отримати доступ до кабінету користувача, навіть якщо дані з бази попадуть до інтернету. По-друге, сама перевірка паролю відбувається на рівні сервера, де повторюється той самий алгоритм, що і при збережені паролю – береться сам пароль, згенерована криптографічна сіль, вони хешуються і зіставляються з хешем паролю, який збережено в БД. Може здатися, що зберігати один хеш паролю без солі теж надійно, але варто розуміти, що один і той самий пароль, наприклад «Querty12345» буде захешований завжди в одну і ту саму послідовність байтів, якщо використовувати один і той самий алгоритм хешування, що потенційно може призвести до так званого «Brute force» [19] хакерської атаки, адже зловмисниками вже давно винайдені програми підбору простих паролів за їх хешем.

В нашій системі весь цей алгоритм відпрацьовує при реєстрації користувача та його аутентифікації (рис. 2.23. - 2.25).

```
public class CodeGenerator : ICodeGenerator
{
    2 references
    public void GenerateHash(string str, out byte[] passwordHash, out byte[] passwordSalt)
    {
        using (var hmacsha = new HMACSHA512())
        {
            passwordSalt = hmacsha.Key;
            passwordHash = hmacsha.ComputeHash(System.Text.Encoding.UTF8.GetBytes(str));
        }
    }
```

Рис. 2.23. Метод для генерації хешу та криптографічної солі



```
0 references
public async Task<UserDTO> Handle(CreateUser request, CancellationToken cancellationToken)
{
    CreateUserDTO dto = request.dto;

    _codeGenerator.GenerateHash(dto.Password, out byte[] passwordHash, out byte[] passwordSalt);

    var user = new User()
    {
        Name = dto.Name,
        Email = dto.Email,
        PasswordHash = passwordHash,
        PasswordSalt = passwordSalt,
        DateOfBirth = dto.DateOfBirth,
        IsActive = false
    };

    var entity = await _uesrRepository.Add(user);
```

Рис. 2.24. Виклик методу для генерації хешу в місці виконання команди на створення користувача

	Id	Name	Email	PasswordHash	PasswordSalt	DateOfBirth	Gender	Relationship
31	31	string	string	0xDA72EAFC...	0x1F77859...	2024-04-03 07:39:00.2220000	0	0
32	32	string	string	0xFD188018...	0x5575697...	2024-04-03 08:13:19.5410000	0	0
33	33	Slok	kasterov.v@gm...	0x4A9BF240...	0xE785213...	2024-04-08 21:00:00.0000000	0	0
34	34	Cybe...	kasterov.v@gm...	0x611121B0...	0x6FE5F04...	2024-04-29 21:00:00.0000000	0	0
35	35	Sege...	kasterov.v@gm...	0x80ED3671...	0xA227BD8...	2024-04-29 21:00:00.0000000	0	0
36	36	s	kasterov.v@gm...	0x7F0EA104...	0x6059BF0...	2024-04-28 21:00:00.0000000	0	0
37	37	Test ...	kasterov.v@gm...	0x789E36E3...	0x4DB50FD...	2024-04-24 21:00:00.0000000	0	0
38	38	s	kasterov.v@gm...	0xFF4AD52E...	0x015F74C...	2024-04-29 21:00:00.0000000	0	0
39	39	TOTA...	kasterov.v@gm...	0x54BEB466...	0xFFD99B2...	2024-04-29 21:00:00.0000000	0	0
40	40	LOXI	kasterov.v@gm...	0xFB8A5012...	0xFC51B92...	2024-04-28 21:00:00.0000000	0	0
41	41	sts	kasterov.v@gm...	0x4D502819...	0x1993241...	2024-04-29 21:00:00.0000000	0	0
42	42	Lol	kasterov.v@gm...	0x7386B363...	0x6E170C4...	2024-04-23 21:00:00.0000000	0	0
43	43	s	kasterov.v@gm...	0xFD4774CB...	0xFDBD6A1...	2024-04-23 21:00:00.0000000	0	0
44	44	200	kasterov.v@gm...	0x42704227...	0x1F4FD3D...	2024-04-29 21:00:00.0000000	0	0
45	45	Jolud	kasterov.v@gm...	0x17E40215...	0x789B42C...	2024-04-29 21:00:00.0000000	0	0
46	46	1	kasterov.v@gm...	0x80DF6F10...	0x814A0F1...	2024-04-22 21:00:00.0000000	0	0

Рис. 2.25. Вигляд одного і того самого паролю збереженого для різних користувачів в БД нашої системи

Останнім аспектом структури системи є відсилання повідомлень на пошту користувача для верифікації. З точки зору бізнес логіки, тільки користувачі, що підтвердили валідність своєї пошти після реєстрації, можуть увійти в систему та почати створювати опитування та голосувати в інших опитуваннях. Для того, щоб підтвердити валідність пошти користувача, було вирішено відправляти на вказані під час реєстрації поштові дані - лист. Цей лист містить актуальне посилання на клієнтську частину додатку. Перейшовши за цим посиланням, користувач підтверджує, що отримав листа і його пошта валідна.

В нашій системі відсилання листів поштою реалізоване сервісом «MailSenderService», який використовується в обробці команди реєстрації користувача (рис. 2.26.). Для відправки листів використано протокол SMTP [20]. Важливо відмітити, що для використання такого підходу треба створити пошту, від імені якої буде відправлятися повідомлення. Для того, щоб сервер міг відправляти повідомлення від імені цієї пошти, потрібно також згенерувати спеціальний код в особистому кабінеті Google account. На рис 2.27. наведено приклад листа, який має прийти від нашої системи для підтвердження пошти користувача.

```
2 references
public class MailSenderService : IMailSenderService
{
    private readonly IConfiguration _builder;

    0 references
    public MailSenderService(IConfiguration builder)
    {
        _builder = builder;
    }

    2 references
    public string SendMail(string subject, string bodyMessage, string recipient)
    {
        var email = new MimeMessage();
        string? sender = _builder.GetSection("Mail:Sender").Value;
        string? password = _builder.GetSection("Mail:Password").Value;

        email.From.Add(MailboxAddress.Parse(sender));
        email.To.Add(MailboxAddress.Parse(recipient));
        email.Subject = subject;
        email.Body = new TextPart(TextFormat.Html) { Text = bodyMessage };

        using var smtp = new MailKit.Net.Smtp.SmtpClient();
        smtp.Connect("smtp.gmail.com", 587, SecureSocketOptions.StartTls);
        smtp.Authenticate(sender, password);

        string res = smtp.Send(email);
        smtp.Disconnect(true);

        return res;
    }
}
```

Рис. 2.26. Сервіс для відправки повідомлень поштою

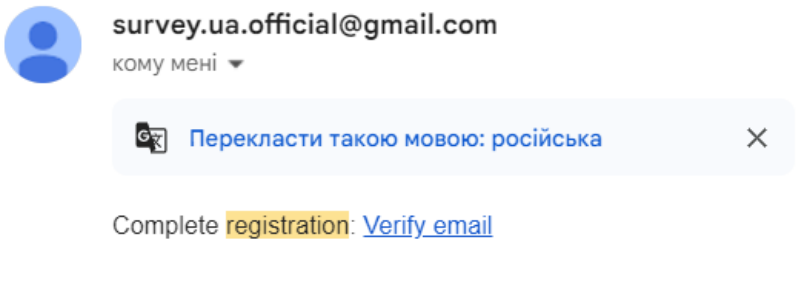


Рис. 2.27. Приклад повідомлення від системи «Survey»

Клієнтська частина має стандартну VUE.JS архітектру, де ми керуємо компонентами та їх станом, який в них зберігається. В каталозі «views» зберігаються компоненти, які представляють собою готові сторінки (рис. 2. 28.). В каталозі «repository» знаходяться класи та методи для взаємодії з нашим сервером, де за допомогою axios робляться HTTP запити (рис. 2. 28.). В каталозі «router» описані HTTP посилання на наші сторінки (рис. 2.29).

В цілому всі компоненти будуються за одним шаблоном. На рис. 2.30 – 2.31. наведено приклад того, як виглядає компонента для реєстрації користувача. Відображення цієї компоненти зображено на рис. 2.28.

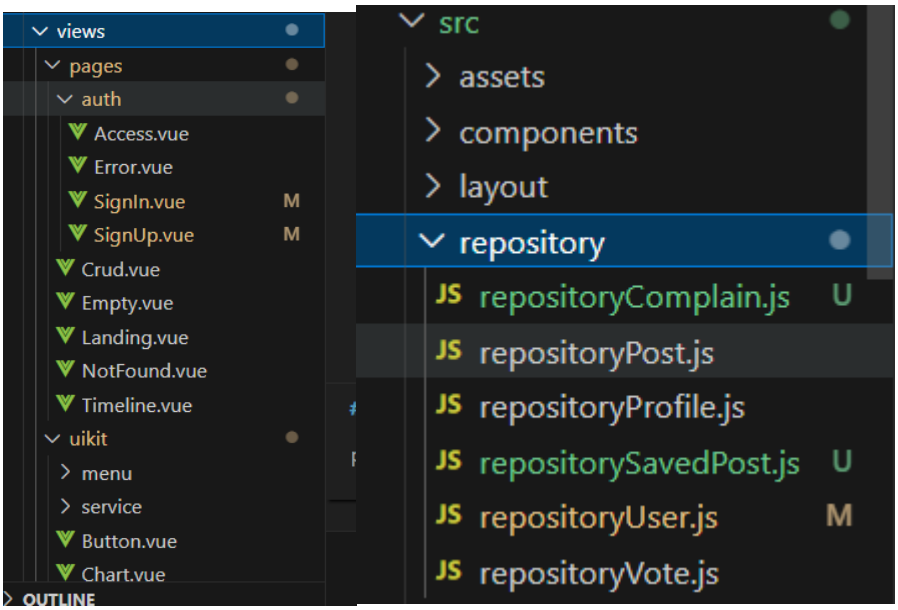


Рис. 2.28. Архітектура клієнтського рішення

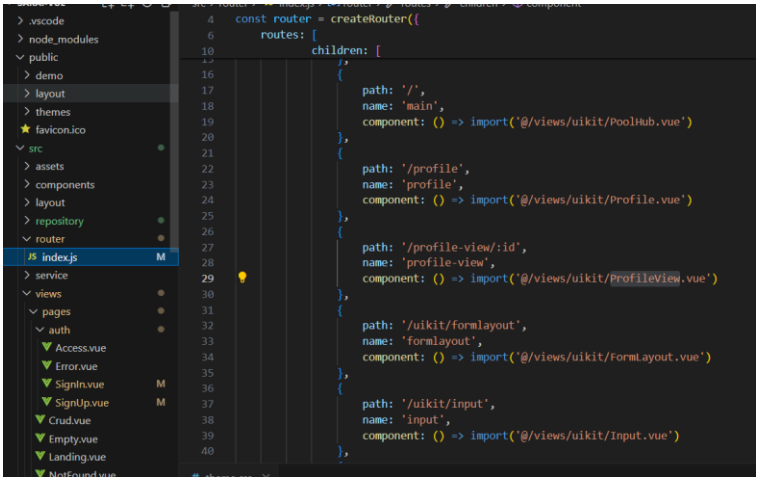
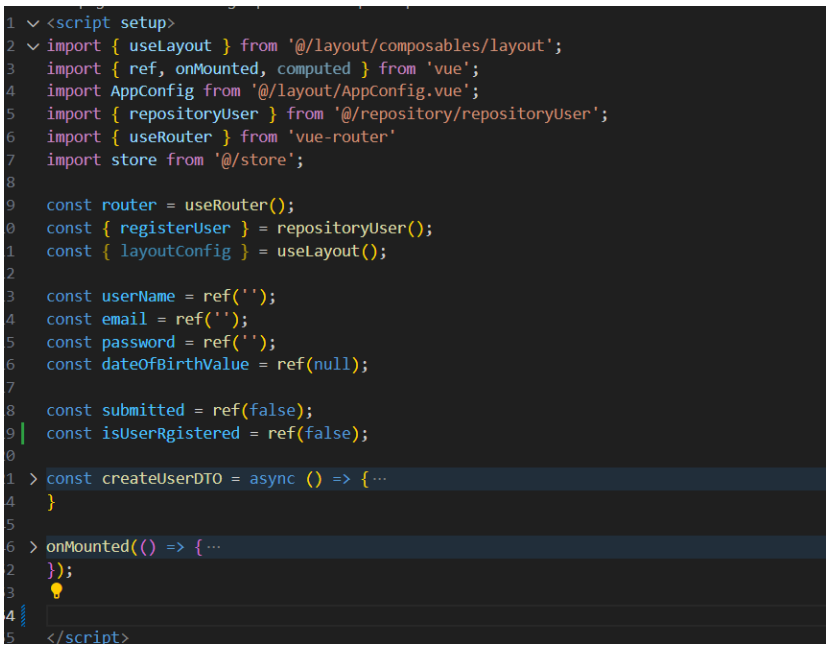
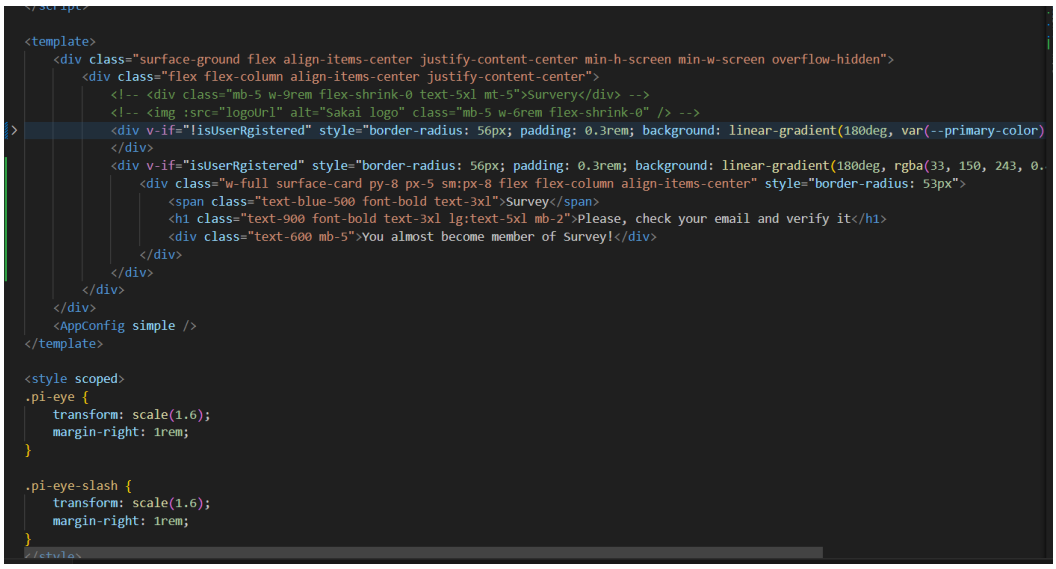


Рис. 2.29. Конфігурація HTTP запитів для доступу до клієнтської частини



```
<script setup>
import { useLayout } from '@/layout/composables/layout';
import { ref, onMounted, computed } from 'vue';
import AppConfig from '@/layout/AppConfig.vue';
import { repositoryUser } from '@/repository/repositoryUser';
import { useRouter } from 'vue-router'
import store from '@/store';

const router = useRouter();
const { registerUser } = repositoryUser();
const { layoutConfig } = useLayout();

const userName = ref('');
const email = ref('');
const password = ref('');
const dateOfBirthValue = ref(null);

const submitted = ref(false);
const isUserRgistered = ref(false);

const createUserDTO = async () => {···
}

onMounted(() => {···
});

</script>
```

Рис. 2.30. Конфігурація методів JS для компоненти реєстрації користувача



```
<template>
    <div class="surface-ground flex align-items-center justify-content-center min-h-screen min-w-screen overflow-hidden">
        <div class="flex flex-column align-items-center justify-content-center">
            <!-- <div class="mb-5 w-9rem flex-shrink-0 text-5xl mt-5">Survery</div> -->
            <!-- <img :src="logoUrl" alt="Sakai logo" class="mb-5 w-6rem flex-shrink-0" /> -->
            <div v-if="!isUserRgistered" style="border-radius: 56px; padding: 0.3rem; background: linear-gradient(180deg, var(--primary-color)
            </div>
            <div v-if="isUserRgistered" style="border-radius: 56px; padding: 0.3rem; background: linear-gradient(180deg, rgba(33, 150, 243, 0.
                <div class="w-full surface-card py-8 px-5 sm:px-8 flex flex-column align-items-center" style="border-radius: 53px">
                    <span class="text-blue-500 font-bold text-3xl">Survey</span>
                    <h1 class="text-900 font-bold text-3xl lg:text-5xl mb-2">Please, check your email and verify it</h1>
                    <div class="text-600 mb-5">You almost become member of Survey!</div>
                </div>
            </div>
        </div>
    </div>
    <AppConfig simple />
</template>

<style scoped>
.pi-eye {
    transform: scale(1.6);
    margin-right: 1rem;
}

.pi-eye-slash {
    transform: scale(1.6);
    margin-right: 1rem;
}
</style>
```

Рис. 2.31. Конфігурація макету форми реєстрації користувача за допомогою HTML, CSS та методів JS

2.5. Обґрунтування та організація вхідних та вихідних даних програми

У ході імплементації Clean architecture як вже було наголошено, важливо було відокремити різні шари нашого застосунку один від одного і звести все до абстракцій, але також варто зауважити, що створені нами класи для збереження сутностей в БД не мають використовуватися кінцевим користувачем, тобто ті дані, що ми зберігаємо у БД, і ті що відправляє нам користувач, чи ми йому, мають бути описані різними класами. Такий підхід називається Data transfer object (DTO). Тобто наш сервер майже завжди працює з DTO класами, а класи описані на рівні БД потрібні тільки для взаємодії з БД, це дозволяє розділити роботу з даними і абстрагуватись від кінцевої реалізації у БД. Для того щоб перетворити сутність класу з БД в сутність класу DTO, можна використати бібліотеку .NET AutoMapper, яка дозволяє автоматично створювати одні об’єкти на основі інших, використовуючи заготовлені конструктори. Чому це важливо? На рис. 2.32. показано приклад того як виглядає запит на створення користувача зі сторони сервера, а на рис 2.33. показано як виглядає клас, що описує користувача збереженого в БД. Різниця в тому, що користувач в БД буде зберігатися з такими властивостями як криптографічна сіль, ідентифікаційний номер (Id) та багато іншого. Під час створення користувача, тобто його реєстрації, вся ця інформація не потрібна, вона буде занесена на основі бізнес логіки серверу. Тому всі вхідні та вихідні дані описані на рівні шару Application в каталозі «DTOs», а всі класи, що описують сутності бази збережені в окремому шарі Domain в каталозі «Entites». Важливо відмітити, що у деяких випадках з БД можна одразу діставати модель DTO. На рис. 2.34. можна побачити такий приклад. Оскільки дістати з бази треба досить не стандартну модель, то за допомогою LINQ ми будуємо запит на отримання саме DTO а не звичайного класу з БД.

Рис. 2.32. Запит на створення користувача

```
65 references
public class User : BaseAuditableEntity
{
    34 references
    public string Name { get; set; }
    26 references
    public string Email { get; set; }
    23 references
    public byte[] PasswordHash { get; set; }
    23 references
    public byte[] PasswordSalt { get; set; }
    4 references
    public bool IsActive { get; set; } = true;
    26 references
    public DateTime DateOfBirth { get; set; }
    0 references
    public List<Post> Posts { get; set; }
    1 reference
    public List<Vote> Votes { get; set; }
    1 reference
    public List<Complain> Complains { get; set; }
    3 references
    public Gender Gender { get; set; }
    3 references
    public Relationship Relationship { get; set; }
    3 references
    public int? SpendingPerMonth { get; set; }
    4 references
    public string? Bio { get; set; }
    4 references
    public int? CountryId { get; set; }
    5 references
    public Country Country { get; set; }
    7 references
    public int? AvatarId { get; set; }
    4 references
    public Media Avatar { get; set; }
    1 reference
    public List<UserCode> UserCodes { get; set; }
    1 reference
    public List<SavedPost> SavedPosts { get; set; }
    7 references
    public List<UserJob> Jobs { get; set; }
    7 references
    public List<UserHobby> Hobbies { get; set; }
    7 references
    public List<UserEducation> Educations { get; set; }
}
```

Рис. 2.33. Клас користувача збереженого в БД

```
2 references
public async Task<IEnumerable<PostLiteDTO>> GetAll()
{

    List<PostLiteDTO> postLiteDTOList = await _context.Posts.Select(post => new PostLiteDTO()
    {
        Id = post.Id,
        Title = post.Title,
        Created = post.Created,
        LastModified = post.LastModified,
        Author = new AuthorDTO()
        {
            Name = post.Author.Name,
            AvatarId = post.Author.AvatarId,
            Id = post.Author.Id,
        },
        Votes = post.Options.Select(o => new VoteLiteDTO()
        {
            Id = o.Id,
            Option = o.Title,
            Count = o.Votes.Count,
        })
    }).AsNoTracking()
    .ToListAsync();

    return postLiteDTOList;
}
```

Рис. 2.34. Приклад формування DTO на пряму з БД

Тож у нашій системі у підсумку ми маємо таку логіку роботи з вхідними та вихідними даними: на вході ми маємо DTO, які приходять нам від нашої клієнтської частини, а зберігається все у вигляді моделей класів в БД.

Як вже було зазначено раніше, за рахунок міграцій ми можемо створювати на основі класів таблиці у БД. Детальний опис основних сутностей БД зазначено на рис. 2.35. – 2.38.

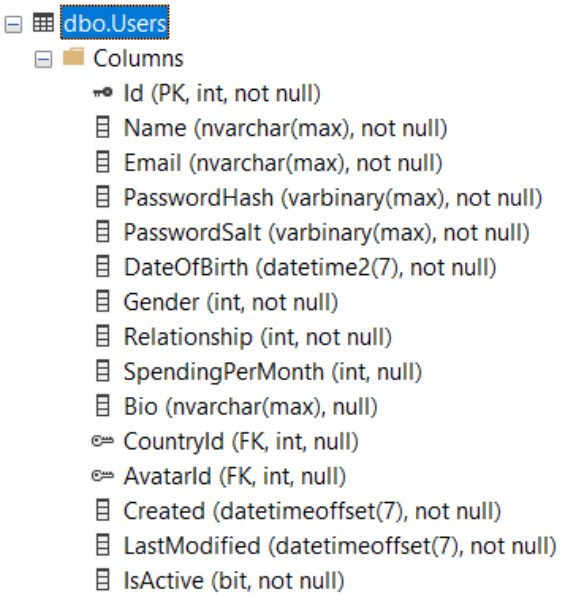


Рис. 2.35. Колонки таблиці Users (користувачів)

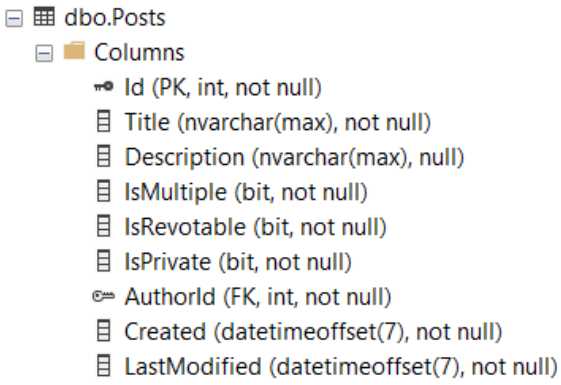


Рис. 2.36. Колонки таблиці Posts (опитування)

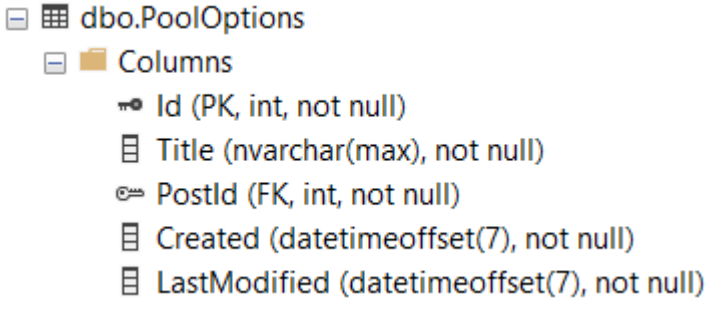


Рис. 2.37. Колонки таблиці PoolOptions (варіанти відповідей на опитування)

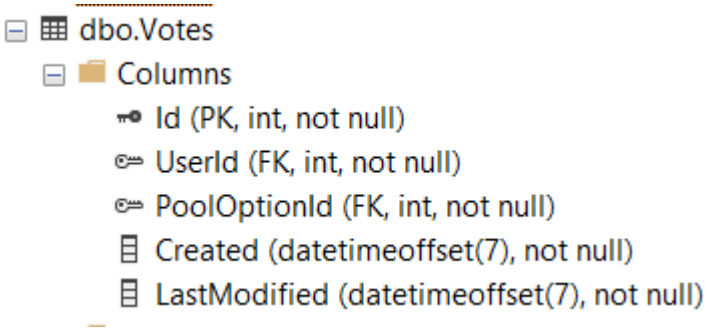


Рис. 2.38. Колонки таблиці Votes (Голоси, що були дані на відповіді у опублікованому опитуванні)

Таким чином була побудована структура РБД нашої системи. На рис. 2.39. зображений підсумковий варіант вигляду діаграми БД на момент накочування останньої міграції. Всі приклади наведені в даній кваліфікаційній роботі посилаються на функціонал розроблений да даній версії БД.

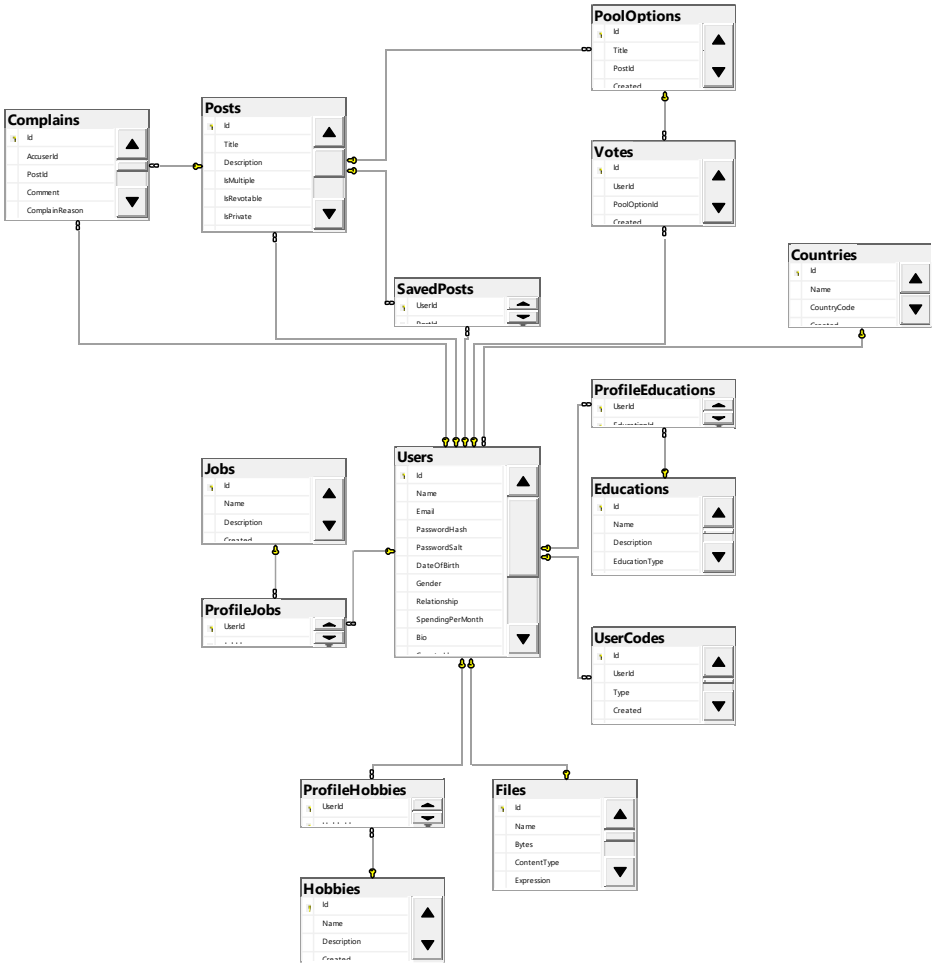


Рис. 2.39. Підсумкова ER-діаграма БД системи «Survey»

2.6. Опис розробленої системи

2.6.1. Використані технічні засоби

Вебплатформа не потребує великих потужностей для своєї роботи. Платформу було запущено на локальному сервері ноутбука з мінімальними характеристиками, що дозволило обробляти певну кількість клієнтських запитів. Для розробки платформи використовувалися такі характеристики системи:

- Операційна система: Windows;
- Процесор Intel Core I7 10 покоління;
- Обсяг оперативної пам'яті (RAM) у 16 гігабайт;
- Жорсткий диск або SSD з об'ємом 1 Т;
- Наявність монітора, миші (тачпаду) та клавіатури для розробки

2.6.2. Використані програмні засоби

Під час проєктування платформи були використані наступні програмні засоби:

- Браузер Chrome версії 109;
- IDE Microsoft Visual Studio Community 2022 (для розробки серверу);
- IDE Visual Studio Code (для розробки клієнтської частини);
- MS SQL Server;

Для серверної частини була використана мова програмування C# та платформа .NET 7. Під час виконання кваліфікаційної роботи для серверної частини були використані такі бібліотеки: EF core, AutoMapper, MediatR, Newtonsoft.Json, Shawshbuckle.AspNetCore, Microsoft.AspNetCore.Http.

Для клієнтської частини використано VUE.JS фреймворк та «Sakai» бібліотека готових стилів, мова програмування JS, а також HTML 5 та CSS 3. Під час розробки були використані такі бібліотеки: axios, vue-router, primevueapi та luxon.

2.6.3. Виклик та завантаження програми

Для запуску серверної частини потрібно клонувати проєкт з репозиторію [25], запустити його у Visual Studio 2022 (при цьому обов'язково повинен бути встановлений .NET). Для простої роботи основного функціоналу проєкта можна нічого не змінювати у файлах конфігурації, але для відправки листів онлайн поштою, треба занести дані для під'єднання пошти з якої буде відбуватися розсилка у файлі appsettings.json. Варто зазначити, що для створення БД треба перейти за вкладинкою Tools, NuGet package manager, package manager console та викликати команду «Update-database».

Для запуску клієнтської частини треба склонувати репозиторій клієнтської частини [24] та в Visual Studio Code відкрити термінал та прописати команду «npm install», а після цього «npm run dev». Варто зауважити, що клієнтська частина не буде працювати коректно без серверної.

Після цих дій відкриється вікно у браузері за замовчуванням, де можна почати роботу с користувальницьким інтфейсом сервера чи клієнта.

2.6.4. Опис інтерфейсу користувача

Головна сторінка нашої вебплатформи представляє собою динамічний список всіх постів, що створені в системі. Опитування відображаються за датою створення, тобто найперші – це найактуальніші (рис. 2.40.). Зверху розміщено поле пошуку, яким може скористуватися користувач і знайти опитування за назвою.

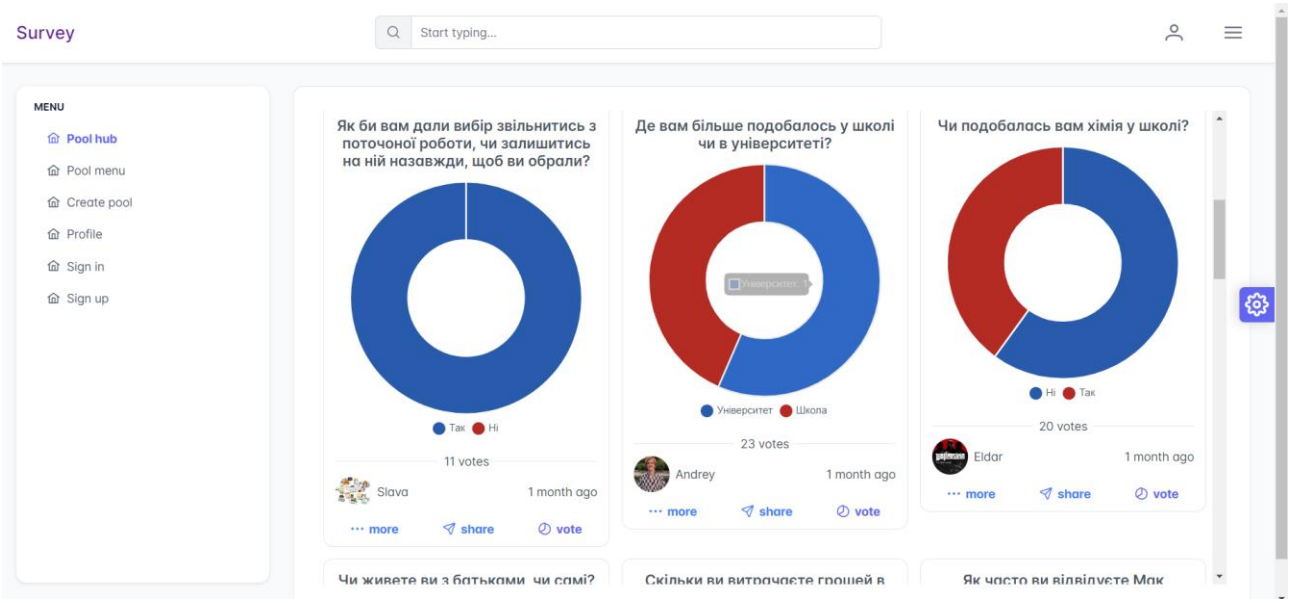


Рис. 2.40. Головна сторінка нашої вебплатформи

Варто більш детально описати як саме виглядає карточка опитування і яка інформація в ній відображається. По-перше, на рис. 2.40. можна побачити як зверху кожної карточки відображається саме питання, на яке користувачам пропонується відповісти. По-друге, знизу, на кожній карточці, відображається діаграма відповідей на опитування. Під діаграмою опис відповідей та їх колір на діаграмі, а ще нижче кількість користувачів, що вже відповіли. Якщо навестись курсором миші на діаграму, чи клацнути саме на неї з телефону, то ви побачите скільки саме респондентів обрали той чи інший варіант відповіді. Після загальної кількості голосів іде особиста інформація автора, а саме його ім’я та аватар, якщо він є. Справа від користувача описана дата публікації опитування у форматі «скільки часу пройшло з публікації».

З самого низу три кнопки, де центральна «share» для копіювання посилання на це опитування, кнопка «vote» для голосування (вона викликає модальне вікно з опціями вибору). Ліва кнопка відповідає за виклик меню, яке дозволяє обрати користувачу дію, або експортувати дані з цього опитування у форматі csv, або поскаржитись на опитування, або зберегти опитування, щоб не загубити його (рис. 2.41.).

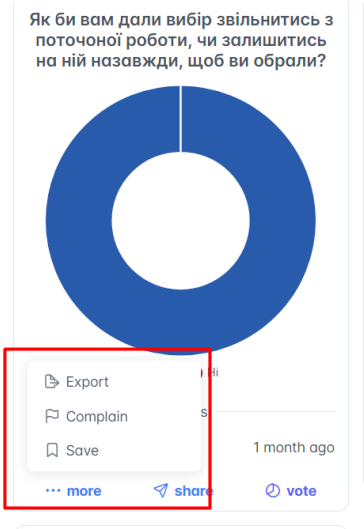


Рис. 2.41. Випадаючий список дій

Якщо користувач буде гортати опитування, тоді вони автоматично будуть підтягуватись з сервера, що забезпечує можливість нескінченного зручного перегляду опитувань.

Як вже було сказано, якщо натиснути на кнопку «vote» тоді відкриється модальне вікно для голосування. В модальному вікні окрім варіантів відповідей також є більш детальний опис цього опитування (рис. 2.42.). Також, якщо це опитування може мати декілька варіантів відповідей від одного користувача (рис. 2.43.), тоді поряд з варіантами будуть квадратні чекбокси, а якщо користувач може обрати лише одну відповідь, то круглі. Варто зауважити, що опитування може мати заборону на повторне голосування, тобто зміну відповіді, тоді при відкриті такого опитування після голосування замість кнопки «vote» буде напис про те, що користувач вже проголосував (рис. 2.43.).

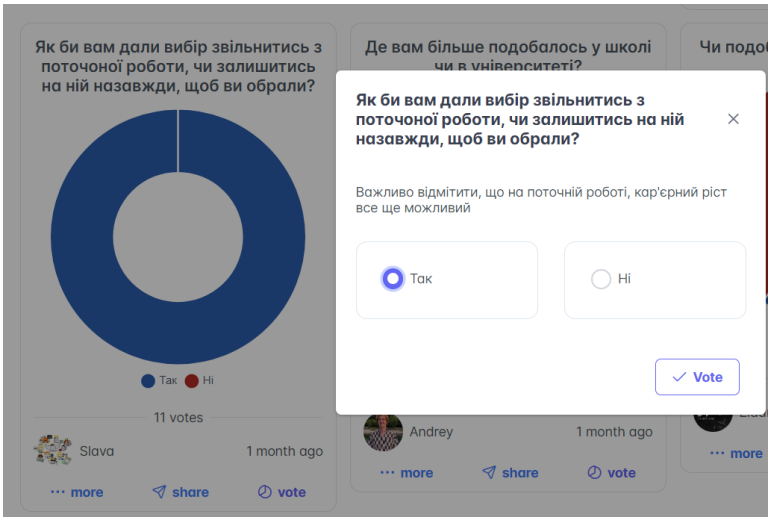


Рис. 2.42 Модальне вікно для голосування

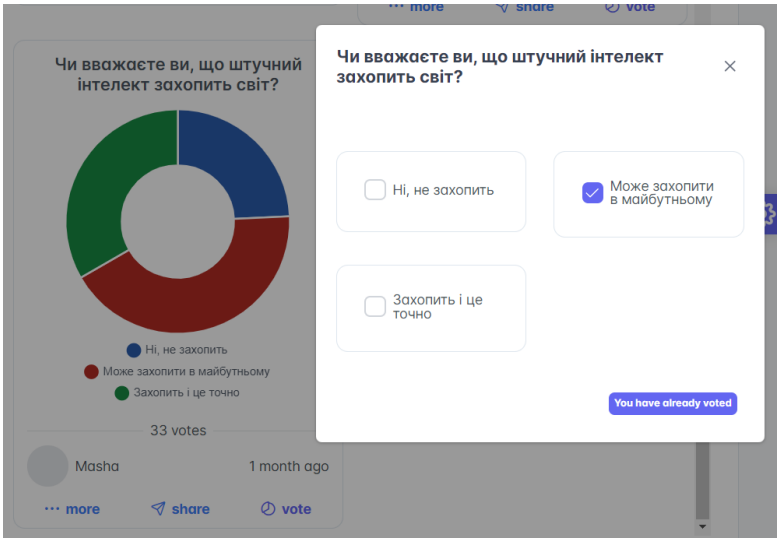


Рис. 2.43. Модальне вікно для голосування, де користувач вже проголосував, також користувач міг обрати декілька відповідей

Якщо спробувати поскаржитись на опитування, тоді теж з’явиться модальне вікно, де можна буде вибрати причину скарги з наведених варіантів та додати коментар, що саме вам не сподобалось в опитувані (рис. 2.44.).

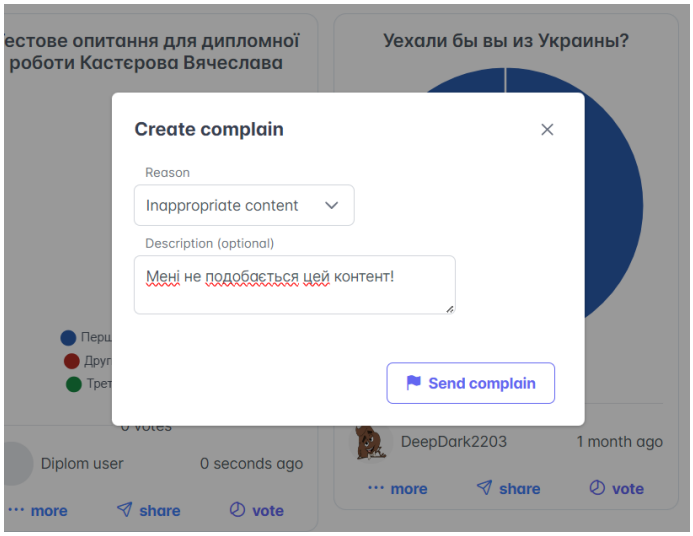


Рис. 2.44. Модальне вікно для створення скарги

Важливим аспектом роботи нашої системи є реєстрація користувача. На рис. 2.45. зображена форма, заповнивши яку користувачі можуть створити власний акаунт у системі «Survey».

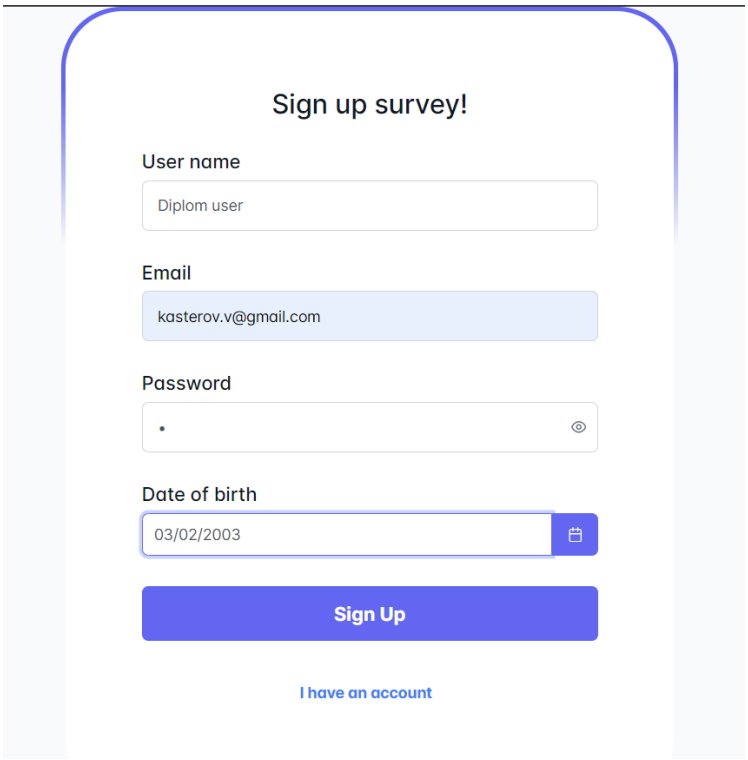


Рис. 2.46. Форма реєстрації заповнена тестовими даними

Після натискання кнопки «Sign Up» користувач буде нотифікований, що реєстрація пройшла успішно і йому треба підтвердити свою пошту, щоб почати користуватися платформою повністю (рис. 2.45.).

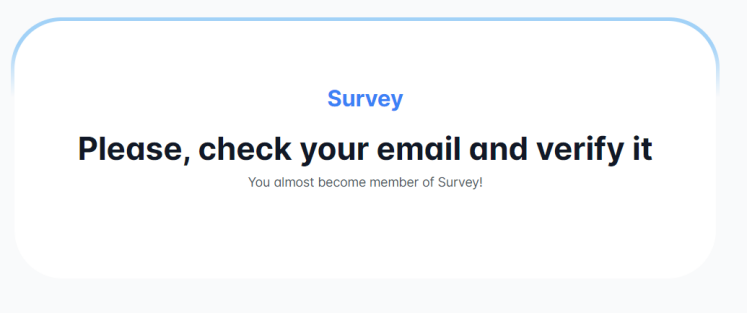


Рис. 2.47. Повідомлення, що користувач створений

На пошті ми знайдемо посилання на верифікацію користувача (рис. 2.27.). Перейшовши за посиланням, користувач побачить повідомлення про те, що він успішно приєднався до платформи «Survey» (рис. 2.46.). Далі у користувача дві опції, він може авторизувати себе, або залишитися анонімним. Для того щоб увійти у систему, користувач має клікнути «Sign In» і його перекине на форму авторизації (рис. 2.47.).

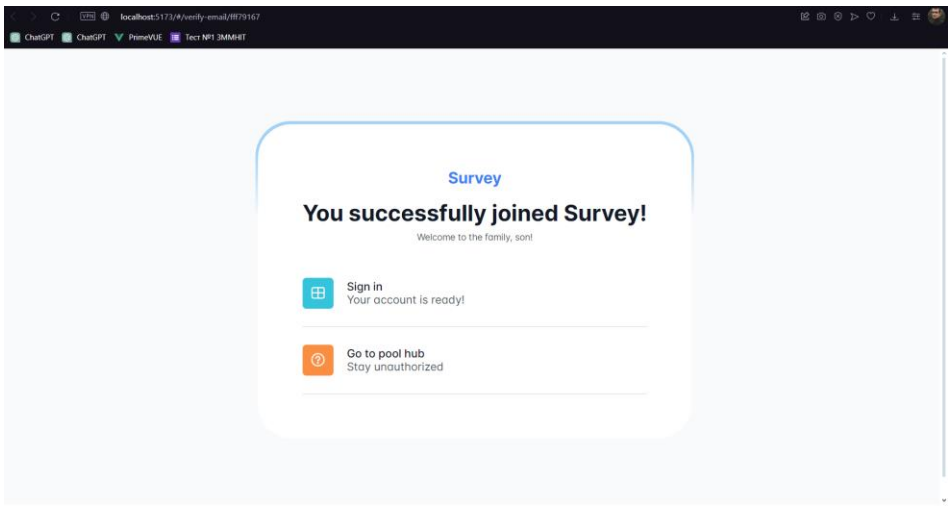


Рис. 2.46. Повідомлення, що користувач створений

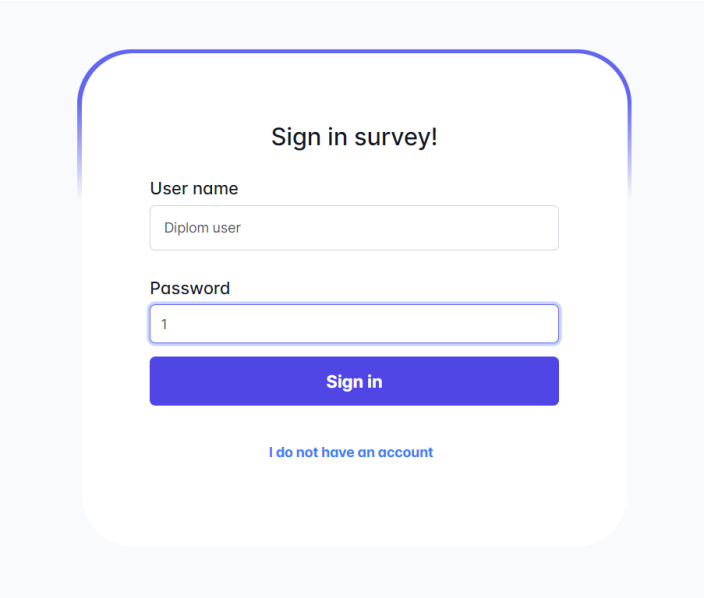


Рис. 2.47. Форма авторизації створеного користувача

Після авторизації користувача, він може створювати опитування, голосувати в інших опитуваннях, та керувати вже створеним опитуваннями. На рис. 2.48. зображена форма для створення нового опитування з тестовими даними. В формі можна обрати чи буде опитування мати декілька варіантів відповідей, чи можна буде переголосувати в ньому та чи не є це опитування приватним (з доступом лише за посиланням). Також до опитування можна додати детальний опис. Після створення опитування, можна перейти на головну сторінку та побачити, що воно вже додане (рис. 2.49.). Щоб змінити чи видалити опитування, користувач може перейти до вкладинки «Pool menu», та побачити весь контент, що був створений (рис. 2.50.). Таблиця контенту опитувань представляє собою такі колонки як «Title», «Status», «Votes», «People», «Created», «Updated». В цілому за назвами колонок зрозуміло, який контент в них зберігається, але варто зауважити, що різниця між «Votes» та «People» в тому, що в першій колонці кількість всіх голосів, а у другій колонці кількість людей, що проголосували (це може бути корисно, якщо опитування мають багато варіантів відповідей і кількість голосів не відображає реальну кількість респондентів). Також в меню є кнопка, що посилає на створення нового опитування.

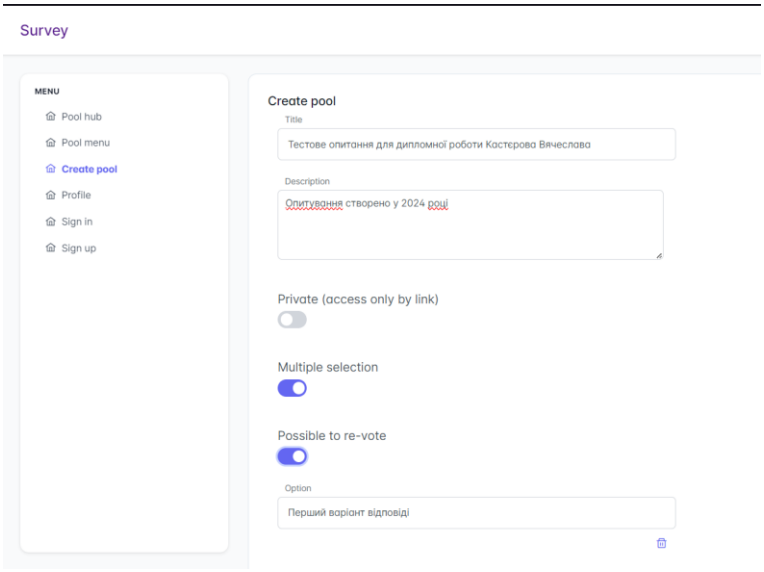


Рис. 2.48. Форма створення опитування

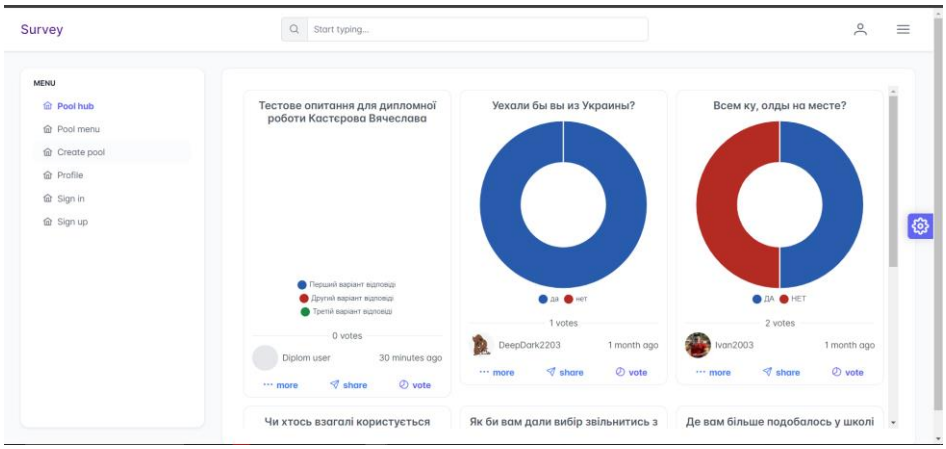


Рис. 2.49. Нове додане опитування

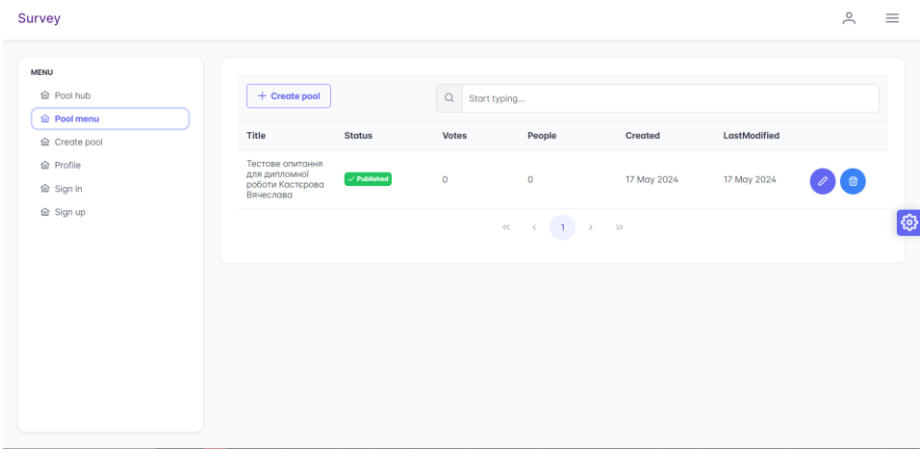


Рис. 2.50. Меню керування опитуваннями

Користувач має змогу перейти до особистого профіля та додати інформацію про себе, наприклад аватарку, обрати хобі, де працює та де навчається. Користувач може також обрати країну проживання та змінити пошту чи дату народження (рис. 2.51.). Також користувач може переглянути детальну інформацію та аналітику по опитуванню, якщо клікне на нього на головній сторінці (рис. 2.52.).

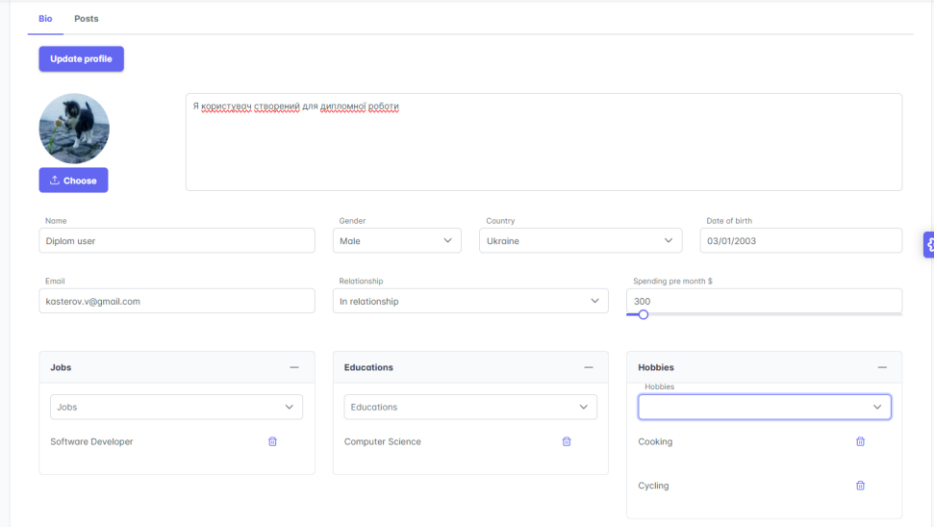


Рис. 2.51. Особистий кабінет користувача

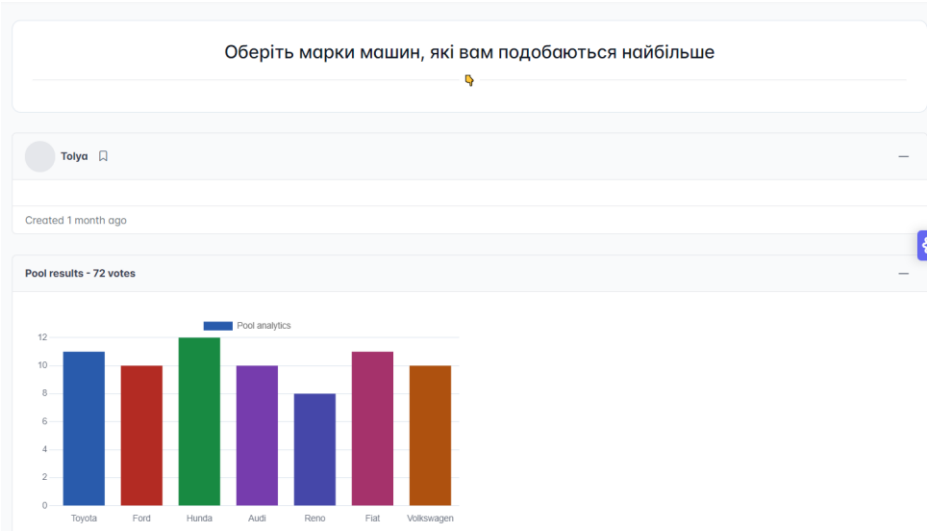


Рис. 2.52. Детальний огляд опитування

Інші користувачі мають змогу переглянути ваш профіль як і ви самі, для цього треба клікнути на аватарку автора на головній сторінці всіх опитувань (рис. 2.53.).

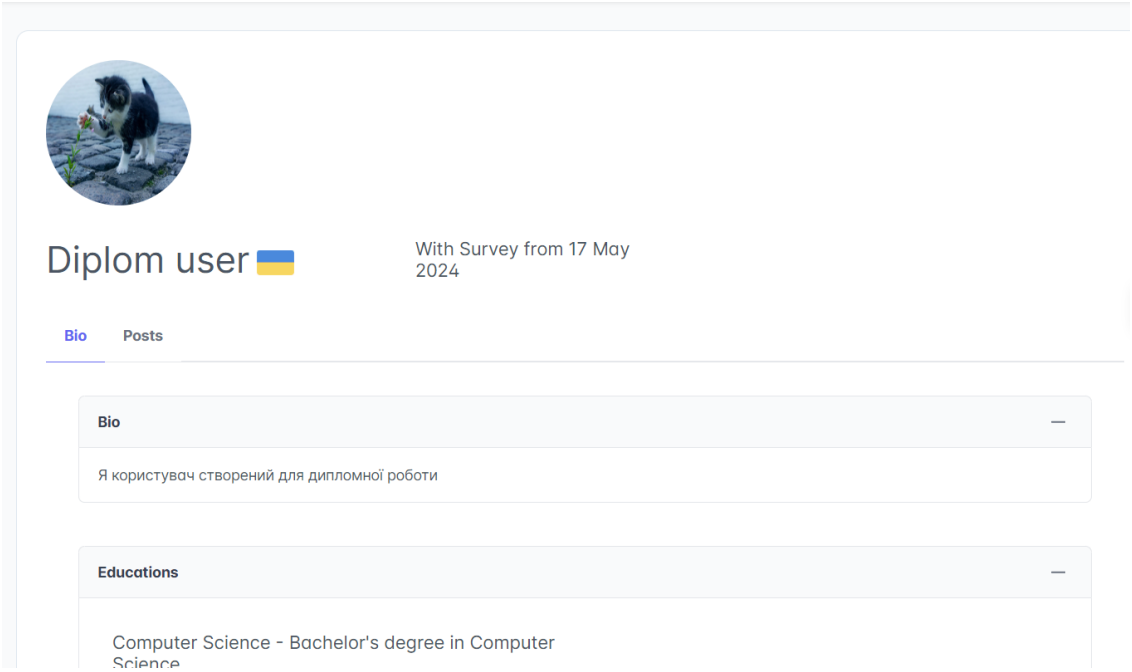


Рис. 2.53. Сторінка користувача для перегляду

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Під час розробки веб-платформи «Survey», для створення та поширення опитувань, було проведено розрахунок трудомісткості та вартості з урахуванням наступних вихідних даних:

Передбачуване число операторів програми: 2000;

Коефіцієнт складності програми: 1,2;

Коефіцієнт корекції програмного продукту в ході його розробки: 0,1;

Годинна заробітна плата розробника рівня Middle: 625 грн/год. Згідно зі статистикою з інтернет платформи «DOU» [21]. 2500 доларів за місяць, тобто за 160 робочих годин, приблизно 15 доларів за годину. Згідно поточного курсу долару до гривні (1 долар 40 гривень).

Коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі: 1,25;

Коефіцієнт кваліфікації програміста, залежний від стажу роботи: 1;

Вартість машино-години ЕОМ: 0.19 грн/год. Розрахунок був проведений за наступним алгоритмом: загалом написання кваліфікаційної роботи разом з написанням коду зайняло приблизно 8 місяців, де в середньому на роботу витрачалось півгодини кожен день, з урахуванням вихідних днів. Тобто за 8 місяців на написання роботи було витрачено 120 годин безперервного використання ноутбуку. Згідно тарифного плану [22] кВт/год коштує 2,64 грн з урахуванням ПДВ. Ноутбук використаної моделі витрачає 60 Вт на годину під час активної роботи, тобто в цілому було витрачено 7200 Вт, або 7,2 кВт, а сума витрат електроенергії на годину становить 0,02 гривень.

У ході виконання кваліфікаційної роботи, були використані різні інтернет джерела. Для забезпечення безперервного та стабільного інтернет з’єднання протягом 8 місяців використовувався тариф від компанії «Київстар» "Все разом

WOW+ТБ" [23], де місяць надання послуг коштує 125 грн. Загальна сума витрат на інтернет 1000 грн, а витрати на інтернет за годину 0,17 копійок.

Трудомісткість розробки програмного продукту можна розрахувати за допомогою формули:

(3.1)

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,}$$

де t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

(3.2)

$$Q = q \cdot C \cdot (1 + p),$$

де q – передбачуване число операторів (2000);

C – коефіцієнт складності програми (1,2);

p – коефіцієнт корекції програмного продукту в ході його розробки (0,1).

Таким чином, умовне число операторів становить:

$$Q = 2000 \cdot 1{,}2 \cdot (1 + 0{,}1) = 2640;$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

(3.3)

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}, \text{ людино-годин,}$$

де *B* – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі (1,25);

k – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності (1);

З урахуванням коефіцієнта кваліфікації k = 1, отримуємо витрати праці на вивчення опису завдання:

$$t_u = (2640 \cdot 1{,}25) / (85 \cdot 1) = 38{,}82 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

(3.4)

$$t_a = \frac{Q}{(20 \ldots 25) \cdot k}, \text{ людино-годин.}$$

де *Q* – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.4), отримаємо:

$t_a = 2640 / (20 \cdot 1) = 132$ людино-годин.

Витрати на складання програми по готовій блок-схемі:

(3.5)

$$t_n = \frac{Q}{(20 \ldots 25) \cdot k}, \text{ людино-годин.}$$

$t_n = 3960 / (25 \cdot 1) = 105{,}6$ людино-годин.

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

(3.6)

$$t_{отл} = \frac{Q}{(4..5) \cdot k}, \text{ людино-годин.}$$

$t_{отл} = 2640 / (5 \cdot 1) = 528$ людино-годин.

– за умови комплексного налагодження завдання:

(3.7)

$$t_{отл}^{k} = 1{,}5 \cdot t_{отл}, \text{ людино-годин.}$$

$t^{k}_{отл} = 1{,}5 \cdot 528 = 792$ людино-годин.

Витрати праці на підготовку документації:

(3.8)

$$t_{д} = t_{дp} + t_{до};$$

де $t_{дp}$ – трудомісткість підготовки матеріалів і рукопису;

(3.9)

$$t_{др} = \frac{Q}{(15..20) \cdot k}, \text{ людино-годин.}$$

(3.10)

$$t_{до} = 0{,}75 \cdot t_{дР};$$

Де $t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

Підставляючи відповідні значення, отримаємо:

$t_{дp} = 2640 / (18 \cdot 1) = 146{,}6$ людино-годин.

$t_{до} = 0{,}75 \cdot 220 = 109{,}95$ людино-годин.

$t_{д} = 220 + 165 = 256{,}55$ людино-годин.

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

t = 50 + 38,82 + 132 + 105,6 + 528 + 256,55 = 1110,97 людино-годин.

За результатами розрахунків, загальна трудомісткість розробки даного програмного продукту складає 1110,97 людино-годин.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн}, \quad (3.11)$$

$З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн}, \quad (3.12)$$

де t - загальна трудомісткість, людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

З урахуванням того, що середня годинна зарплата розробника становить 625 грн / год, отримуємо:

$З_{ЗП}$ = 1110,97 · 625 = 694356,25 грн.

Вартість машинного часу $З_{МВ}$, необхідного для налагодження програми на ЕОМ, визначається за формулою: (3.13)

$$З_{мв} = t_{отл} \cdot С_{мч}, \text{ грн},$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год;

$С_{мч}$ – вартість машино-години ЕОМ, грн/год.

$З_{мв}$ = 528 · 0,19 = 100,32 грн.

Звідси витрати на створення програмного продукту:

$К_{ПО}$ = 694356,25 + 100,32= 694456,57 грн.

Очікуваний період створення ПЗ:

(3.14)

$$T = \frac{t}{B_к \cdot F_р}, \text{міс},$$

де $В_к$ – число виконавців;

$F_р$ – місячний фонд робочого часу (при 40 годинному робочому тижні $F_р$ = 176 годин).

Звідси, за формулою (3.14) витрати на створення програмного продукту:

T = 1110,97 / (1·176) = 6,3 міс.

Висновок: розробка вебплатформи для створення та розповсюдження опитувань «Survey» буде коштувати 694456,57 грн з урахуванням роботи над проєктом однієї людини, яка має рівень Middle та два роки комерційного досвіду розробки програмного забезпечення. Приблизний час, потрібний для розробки складає 6,3 місяців при стандартному 40 годинному робочому тижні. Загальна кількість людино-годин, яка буде витрачена на розробку, складає 1110,97.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було розроблено веб платформу для створення, розповсюдження та аналізу опитувань. Платформа виконана з використанням C# та ASP.NET Core для серверної частини, VUE.JS для клієнтської частини. Також у якості РБД було використано MS SQL. Платформа була розроблена з використанням усіх раніше визначених патернів та архітектурних стилів, що дозволить розширювати функціонал у майбутньому. Підсумковий результат опубліковано на платформі Github у якості двох репозиторіїв, де перший репозиторій – це клієнтська частина, а другий – серверна.

У ході виконання кваліфікаційної роботи, були виконані усі вимоги зазначені у першому розділі. Веб платформа безпечно зберігає дані користувачів та працює відповідно до сучасних проєктно-технічних стандартів. Архітектурні рішення, прийняті на початкових етапах написання кваліфікаційної роботи, дозволять в майбутньому повертатися до даного проєкту та додавати в нього новий функціонал. Всі елементи вебплатформи слабо зв’язані, тому за потреби, можна, наприклад змінити фреймворк для клієнтської частини, що дозволить покращити та вдосконалити функціонал, чи найняти розробників, що будуть виключно займатись клієнтською частиною платформи, розробляючи її за допомогою інших технологій.

З використанням даної платформи, кожен охочий може опублікувати у вільний доступ опитування майже на будь-яку тему, та отримати справжні неупереджені відповіді таких самих реальних користувачів. Результатом публікації опитувань, може стати ретельний аналіз отриманих даних чи просто цікавий висновок про те як розмірковують респонденти з приводу того чи іншого питання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке соціальне опитування. [Електронний ресурс] URL: https://osvita.ua/vnz/reports/sociology/12369 (дата звернення: 14.05.2024).
2. ASP.NET Core. [Електронний ресурс] URL: https://dotnet.microsoft.com/en-us/apps/aspnet (дата звернення: 14.05.2024).
3. VUE.JS. [Електронний ресурс] URL: https://vuejs.org (дата звернення: 14.05.2024).
4. REST архітектурний стиль. [Електронний ресурс] URL: https://restfulapi.net (дата звернення: 14.05.2024).
5. Clean architecture. [Електронний ресурс] URL: https://www.c-sharpcorner.com/article/clean-architecture-in-asp-net-core-web-api (дата звернення: 14.05.2024).
6. MediatR патерн та CQRS. [Електронний ресурс] URL: https://medium.com/@shalinds/command-query-responsibility-segregation-pattern-with-mediatr-a908b53c33b2 (дата звернення: 14.05.2024).
7. Технологія EF . [Електронний ресурс] URL: https://learn.microsoft.com/en-us/ef (дата звернення: 14.05.2024).
8. LINQ в .NET. [Електронний ресурс] URL: https://learn.microsoft.com/en-us/dotnet/csharp/linq (дата звернення: 14.05.2024).
9. Патерн repository. [Електронний ресурс] URL: https://medium.com/@pererikbergman/repository-design-pattern-e28c0f3e4a30 (дата звернення: 14.05.2024).
10. JWT. [Електронний ресурс] URL: https://jwt.io (дата звернення: 14.05.2024).
11. JWT. [Електронний ресурс] URL: https://swagger.io/docs/specification/authentication/bearer-authentication (дата звернення: 14.05.2024).
12. Samuele Resca. Hands-On RESTful Web Services with ASP.NET Core, 2020. p. 34

13. VUE.JS «Sakai». [Електронний ресурс] URL: https://sakai.primevue.org (дата звернення: 16.05.2024).

14. Clean Architecture діаграма [Електронний ресурс] URL: https://code-maze.com/dotnet-clean-architecture (дата звернення: 16.05.2024).

15. Rebecca M. Riordan. Fluent Entity Framework, 2013. p. 123

16. Rebecca M. Riordan. Fluent Entity Framework, 2013. p. 25

17. Chat GPT [Електронний ресурс] URL: https://chatgpt.com (дата звернення: 16.05.2024).

18. Badrinarayanan Lakshmiraghavan. Pro ASP.NET Web API Security: Securing ASP.NET Web API, 2019. p. 57

19. Badrinarayanan Lakshmiraghavan. Pro ASP.NET Web API Security: Securing ASP.NET Web API, 2019. p. 24

20. David Wood. Programming Internet Email: Mastering Internet Messaging Systems, 2009. p. 54

21. Платформа «DOU», звітність по зарплатній медіані [Електронний ресурс] URL: https://jobs.dou.ua/salaries/?period=2023-12&position=Middle%20SE (дата звернення: 29.05.2024).

22. Платформа «Yasno» з розміщеними актуальними тарифами на електроенергію [Електронний ресурс] URL: https://yasno.com.ua/b2c-tariffs (дата звернення: 29.05.2024).

23. Офіційний веб сайт компанії «Київстар» з використаним тарифним планом [Електронний ресурс] URL: https://www.connect.net.ua/provayder/kyivstar (дата звернення: 29.05.2024).

24. Клієнтська частина веб додатку «Survey» [Електронний ресурс] URL: https://github.com/Kasterov/SurveyClient (дата звернення: 29.05.2024).

25. Серверна частина веб додатку «Survey» [Електронний ресурс] URL: https://github.com/Kasterov/SurveyAPI (дата звернення: 29.05.2024).

ДОДАТОК А

КОД ПРОГРАМИ

PostLiteDTO.cs

```
using Application.DTOs.Common;
using Application.DTOs.Users;
using Application.DTOs.Votes;

namespace Application.DTOs.Posts;

public class PostLiteDTO : BaseDTO
{
    public string Title { get; set; }
    public AuthorDTO Author { get; set; }
    public int TotalCount { get; set; }
    public bool IsMultiple { get; set; }
    public IEnumerable<VoteLiteDTO> Votes { get; set; }
}

public class AuthorDTO : BaseAuditableDTO
{
    public string Name { get; set;}
    public string AvatarLink { get; set; }
    public int? AvatarId { get; set; }

}
```

PostFullDTO.cs

```
using Application.DTOs.Common;
using Application.DTOs.Votes;

namespace Application.DTOs.Posts;

public class PostFullDTO : BaseDTO
{
    public string Title { get; set; }
    public AuthorDTO Author { get; set; }
    public string? Description { get; set; }
    public int TotalCount { get; set; }
    public bool IsMultiple { get; set; }
    public bool IsRevotable { get; set; }
    public bool IsPrivate { get; set; }
    public IEnumerable<VoteLiteDTO> Votes { get; set; }
}
```

PostTableDTO.cs

```
using Application.DTOs.Common;
using Application.DTOs.PoolOptions;
using Application.DTOs.Votes;
using Domain.Entities;

namespace Application.DTOs.Posts;

public class PostTableDTO : BaseDTO
```

```
{
    public string Title { get; set; }
    public int Status { get; set; }
    public int Votes { get; set; }
    public int People { get; set; }
}

public class PostTableFullDTO : BaseDTO
{
    public string Title { get; set; }
    public int Status { get; set; }
    public IEnumerable<PoolOptionDTO> PoolOptions { get; set; }

}
```

ApplicationDbContext.cs

```
public class ApplicationDbContext : DbContext, IApplicationDbContext
{
    public ApplicationDbContext() { }

    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) :
base(options) { }

    public virtual DbSet<User> Users => Set<User>();
    public virtual DbSet<Post> Posts => Set<Post>();
    public virtual DbSet<PoolOption> PoolOptions => Set<PoolOption>();
    public virtual DbSet<Vote> Votes => Set<Vote>();
    public virtual DbSet<Media> Files => Set<Media>();
    public virtual DbSet<Education> Educations => Set<Education>();
    public virtual DbSet<Job> Jobs => Set<Job>();
    public virtual DbSet<Hobby> Hobbies => Set<Hobby>();
    public virtual DbSet<Country> Countries => Set<Country>();
    public virtual DbSet<UserJob> ProfileJobs => Set<UserJob>();
    public virtual DbSet<UserHobby> ProfileHobbies => Set<UserHobby>();
    public virtual DbSet<UserEducation> ProfileEducations => Set<UserEducation>();
    public virtual DbSet<Complain> Complains => Set<Complain>();
    public virtual DbSet<SavedPost> SavedPosts => Set<SavedPost>();
    public virtual DbSet<UserCode> UserCodes => Set<UserCode>();
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        ConfigurePost(modelBuilder);
        ConfigureUser(modelBuilder);
        ConfigureVote(modelBuilder);
        ConfigurePoolOption(modelBuilder);
        ConfigureCountry(modelBuilder);
        ConfigureHobby(modelBuilder);
        ConfigureEducation(modelBuilder);
        ConfigureJob(modelBuilder);
        ConfigureProfileJob(modelBuilder);
        ConfigureProfileEducation(modelBuilder);
        ConfigureProfileHobby(modelBuilder);
        ConfigureComplain(modelBuilder);
        ConfigureSavedPost(modelBuilder);
        ConfigureUserCode(modelBuilder);

        FillProfileTables(modelBuilder);
        //FillDbMock(modelBuilder);

    }
```

```
private void ConfigurePost(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Post>()
        .Property(p => p.AuthorId)
        .IsRequired();

    modelBuilder.Entity<Post>()
        .HasKey(x => x.Id);

    modelBuilder.Entity<Post>()
        .HasMany(p => p.Options)
        .WithOne(o => o.Post)
        .OnDelete(DeleteBehavior.Cascade);

    modelBuilder.Entity<Post>()
        .HasMany(p => p.Complains)
        .WithOne(o => o.Post)
        .OnDelete(DeleteBehavior.Cascade);
}

private void ConfigureUser(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<User>()
        .HasKey(x => x.Id);

    modelBuilder.Entity<Media>()
        .HasOne(u => u.User)
        .WithOne(u => u.Avatar)
        .HasForeignKey<User>(u => u.AvatarId)
        .IsRequired(false)
        .OnDelete(DeleteBehavior.Cascade);
}

private void ConfigureVote(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Vote>()
        .HasKey(x => x.Id);

    modelBuilder.Entity<Vote>()
        .Property(x => x.UserId)
        .IsRequired();

    modelBuilder.Entity<Vote>()
        .Property(x => x.PoolOptionId)
        .IsRequired();

    modelBuilder.Entity<Vote>()
        .HasOne(v => v.User)
        .WithMany(u => u.Votes)
        .HasForeignKey(v => v.UserId)
        .OnDelete(DeleteBehavior.Restrict);

    modelBuilder.Entity<Vote>()
        .HasOne(v => v.PoolOption)
        .WithMany(o => o.Votes)
        .OnDelete(DeleteBehavior.Cascade);
}

private void ConfigurePoolOption(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<PoolOption>()
            .HasKey(x => x.Id);

    modelBuilder.Entity<PoolOption>()
       .Property(x => x.PostId)
```

```
        .IsRequired();

    modelBuilder.Entity<PoolOption>()
        .HasOne(o => o.Post)
        .WithMany(p => p.Options)
        .OnDelete(DeleteBehavior.Cascade);
}

private void ConfigureCountry(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Country>()
        .HasKey(x => x.Id);

    modelBuilder.Entity<Country>()
        .HasMany(p => p.Users)
        .WithOne(u => u.Country)
        .HasForeignKey(p => p.CountryId)
        .IsRequired(false)
        .OnDelete(DeleteBehavior.Restrict);
}

private void ConfigureHobby(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Hobby>()
        .HasKey(x => x.Id);
}

private void ConfigureEducation(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Education>()
        .HasKey(x => x.Id);
}

private void ConfigureJob(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Job>()
        .HasKey(x => x.Id);
}

private void ConfigureProfileJob(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<UserJob>()
    .HasKey(u => new { u.UserId, u.JobId });

    modelBuilder.Entity<UserJob>()
        .HasOne(u => u.User)
        .WithMany(up => up.Jobs)
        .HasForeignKey(u => u.UserId);

    modelBuilder.Entity<UserJob>()
        .HasOne(u => u.Job)
        .WithMany(j => j.ProfileJobs)
        .HasForeignKey(u => u.JobId);
}

private void ConfigureProfileEducation(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<UserEducation>()
    .HasKey(u => new { u.UserId, u.EducationId });

    modelBuilder.Entity<UserEducation>()
        .HasOne(u => u.User)
        .WithMany(up => up.Educations)
        .HasForeignKey(u => u.UserId);
```

```
    modelBuilder.Entity<UserEducation>()
        .HasOne(u => u.Education)
        .WithMany(j => j.ProfileEducations)
        .HasForeignKey(u => u.EducationId);
}

private void ConfigureProfileHobby(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<UserHobby>()
    .HasKey(u => new { u.UserId, u.HobbyId });

    modelBuilder.Entity<UserHobby>()
        .HasOne(u => u.User)
        .WithMany(up => up.Hobbies)
        .HasForeignKey(u => u.UserId);

    modelBuilder.Entity<UserHobby>()
        .HasOne(u => u.Hobby)
        .WithMany(j => j.ProfileHobbies)
        .HasForeignKey(u => u.HobbyId);
}

private void ConfigureComplain(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Complain>()
           .HasKey(x => x.Id);

    modelBuilder.Entity<Complain>()
       .Property(x => x.PostId)
       .IsRequired();

    modelBuilder.Entity<Complain>()
       .Property(x => x.AccuserId)
       .IsRequired();

    modelBuilder.Entity<Complain>()
        .HasOne(o => o.Post)
        .WithMany(p => p.Complains)
        .HasForeignKey(u => u.PostId)
        .OnDelete(DeleteBehavior.Restrict);

    modelBuilder.Entity<Complain>()
        .HasOne(v => v.User)
        .WithMany(o => o.Complains)
        .HasForeignKey(u => u.AccuserId)
        .OnDelete(DeleteBehavior.Restrict);
}
private void ConfigureSavedPost(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<SavedPost>()
    .HasKey(u => new { u.UserId, u.PostId });

    modelBuilder.Entity<SavedPost>()
        .HasOne(u => u.User)
        .WithMany(up => up.SavedPosts)
        .HasForeignKey(u => u.UserId)
        .OnDelete(DeleteBehavior.Restrict);

    modelBuilder.Entity<SavedPost>()
        .HasOne(u => u.Post)
        .WithMany(j => j.SavedPosts)
        .HasForeignKey(u => u.PostId)
        .OnDelete(DeleteBehavior.Restrict);

}
```

```
private void ConfigureUserCode(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<UserCode>()
    .HasKey(u => u.Id);

    modelBuilder.Entity<UserCode>()
        .HasOne(u => u.User)
        .WithMany(up => up.UserCodes)
        .HasForeignKey(u => u.UserId)
        .OnDelete(DeleteBehavior.Restrict);

}




private void FillProfileTables(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Country>()
        .HasData(
            new Country { Id = 1, Name = "Ukraine", CountryCode = "UA" },
            new Country { Id = 2, Name = "United States", CountryCode = "US" },
            new Country { Id = 3, Name = "Canada", CountryCode = "CA" },
            new Country { Id = 4, Name = "United Kingdom", CountryCode = "GB" },
            new Country { Id = 5, Name = "Germany", CountryCode = "DE" },
            new Country { Id = 6, Name = "France", CountryCode = "FR" },
            new Country { Id = 7, Name = "Japan", CountryCode = "JP" },
            new Country { Id = 8, Name = "Australia", CountryCode = "AU" },
            new Country { Id = 9, Name = "Brazil", CountryCode = "BR" },
            new Country { Id = 10, Name = "India", CountryCode = "IN" },
            new Country { Id = 11, Name = "Italy", CountryCode = "IT" },
            new Country { Id = 12, Name = "South Africa", CountryCode = "ZA" }
        );

    modelBuilder.Entity<Hobby>()
        .HasData(
            new Hobby { Id = 1, Name = "Photography", Description = "Capturing moments
through photography" },
            new Hobby { Id = 2, Name = "Cooking", Description = "Exploring and
creating new recipes in the kitchen" },
            new Hobby { Id = 3, Name = "Reading", Description = "Enjoying a variety of
books and literature" },
            new Hobby { Id = 4, Name = "Gardening", Description = "Cultivating plants
and flowers in the garden" },
            new Hobby { Id = 5, Name = "Cycling", Description = "Exploring scenic
routes on a bicycle" },
            new Hobby { Id = 6, Name = "Painting", Description = "Expressing
creativity through painting" },
            new Hobby { Id = 7, Name = "Traveling", Description = "Exploring new
places and cultures" },
            new Hobby { Id = 8, Name = "Playing Music", Description = "Creating
melodies on musical instruments" },
            new Hobby { Id = 9, Name = "Hiking", Description = "Venturing into the
great outdoors on hiking trails" },
            new Hobby { Id = 10, Name = "Yoga", Description = "Practicing yoga for
physical and mental well-being" },
            new Hobby { Id = 11, Name = "Gaming", Description = "Enjoying video games
and board games" },
            new Hobby { Id = 12, Name = "Fishing", Description = "Relaxing by the
water with a fishing rod" },
            new Hobby { Id = 13, Name = "Writing", Description = "Expressing thoughts
and ideas through writing" },
            new Hobby { Id = 14, Name = "Collecting", Description = "Building
collections of various items" },
```

```
            new Hobby { Id = 15, Name = "Dancing", Description = "Expressing joy and
creativity through dance" },
            new Hobby { Id = 16, Name = "Crafting", Description = "Creating handmade
crafts and projects" },
            new Hobby { Id = 17, Name = "Bird Watching", Description = "Observing and
identifying different bird species" },
            new Hobby { Id = 18, Name = "Sculpting", Description = "Creating
sculptures from various materials" },
            new Hobby { Id = 19, Name = "Playing Sports", Description = "Engaging in
various sports activities" },
            new Hobby { Id = 20, Name = "Stargazing", Description = "Observing the
night sky and celestial objects" }
        );


    modelBuilder.Entity<Job>()
        .HasData(
            new Job { Id = 1, Name = "Software Developer", Description = "Develops
software applications" },
            new Job { Id = 2, Name = "Data Scientist", Description = "Analyzes and
interprets complex data sets" },
            new Job { Id = 3, Name = "Marketing Manager", Description = "Plans and
executes marketing strategies" },
            new Job { Id = 4, Name = "Teacher", Description = "Educates and guides
students" },
            new Job { Id = 5, Name = "Nurse", Description = "Provides medical care and
support" },
            new Job { Id = 6, Name = "Architect", Description = "Designs and plans
buildings and structures" },
            new Job { Id = 7, Name = "Accountant", Description = "Manages financial
records and statements" },
            new Job { Id = 8, Name = "Graphic Designer", Description = "Creates visual
concepts using computer software" },
            new Job { Id = 9, Name = "Human Resources Specialist", Description =
"Handles HR-related tasks and policies" },
            new Job { Id = 10, Name = "Chef", Description = "Prepares and cooks food
in a professional kitchen" },
            new Job { Id = 11, Name = "Account Manager", Description = "Manages and
nurtures client relationships" },
            new Job { Id = 12, Name = "Civil Engineer", Description = "Plans, designs,
and oversees construction projects" },
            new Job { Id = 13, Name = "Social Media Manager", Description = "Manages
social media accounts and campaigns" },
            new Job { Id = 14, Name = "Mechanical Engineer", Description = "Designs
and analyzes mechanical systems" },
            new Job { Id = 15, Name = "Project Manager", Description = "Oversees and
coordinates project activities" },
            new Job { Id = 16, Name = "UX/UI Designer", Description = "Creates user
interfaces and experiences" },
            new Job { Id = 17, Name = "Dental Hygienist", Description = "Cleans and
examines patients' teeth and gums" },
            new Job { Id = 18, Name = "Research Scientist", Description = "Conducts
scientific research and experiments" },
            new Job { Id = 19, Name = "Sales Representative", Description = "Sells
products or services to customers" },
            new Job { Id = 20, Name = "Pharmacist", Description = "Dispenses
medications and provides healthcare advice" },
            new Job { Id = 21, Name = "Financial Analyst", Description = "Analyzes
financial data and trends" },
            new Job { Id = 22, Name = "Fitness Trainer", Description = "Guides
individuals in physical exercise routines" },
            new Job { Id = 23, Name = "Interior Designer", Description = "Plans and
designs interior spaces" }
```

Решта коду додається окремо на CD-диску.

ДОДАТОК Б

ВІДГУК

керівника економічного розділу

на кваліфікаційну роботу бакалавра на тему:

«Веборієнтована інформаційна платформа для створення публічних опитувань»

студента групи 122-22-2 Кастєрова Вячеслава Олександровича

Керівник економічного розділу **Л.В. Касьяненко**

доц. каф. ПЕП та ПУ, к.е.н

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	Опис
Пояснювальні документи	
Кастєров В.О.диплом.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кастєров В.О.диплом.pdf	Пояснювальна записка до кваліфікаційної роботи у форматі PDF
Програма	
Кастєров В.О.диплом.rar	Архів. Містить коди програми і скомпільовану програму
Презентація	
Кастєров В.О.диплом.pptx	Презентація кваліфікаційної роботи