КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет радіофізики, електроніки та комп'ютерних систем

Кафедра комп'ютерної інженерії

РОЗРОБКА СИСТЕМИ АНОНІМНОГО ОЦІНЮВАННЯ ЯКОСТІ ОСВІТИ В МЕРЕЖІ УНІВЕРСИТЕТУ

Дипломна робота бакалавра

студентки 4 року навчання

Спеціальність: 123 «Комп'ютерна інженерія»

Ольги МЕЛЬНИК

Науковий керівник

канд. технічних наук Євген СЛЮСАР,

асистент кафедри комп'ютерної інженерії

Рецензент

доктор фізико-математичних наук Євген ІВОХІН,

професор кафедри системного аналізу та теорії прийняття рішень

факультету комп'ютерних наук та кібернетики

До захисту допускаю: Завідувач кафедрою

Юрій БОЙКО /

Ухвалено на засіданні кафедри "____" _____2022 р., протокол № ____

Київ 2022

РЕФЕРАТ

Обсяг роботи: 40 сторінок, містить 16 рисунків, 7 додатків та використано 11 інформаційних джерела.

Ключові слова : ASP.NET, Dapper, React, Material UI, Microservices, MSSQL, Redis, .NET 6, CQRS, анонімність, голосування, зворотній зв’язок

Актуальність роботи полягає у тому, що наразі якість викладання напряму залежить від зацікавленості студентів. В умовах дистанційного навчання та масової цифровізації великій структурі Університету допоможе власна система оцінювання якості освіти у виявленні сторін, які можуть фруструвати студентство.

Мета: створити архітектуру, обрати інструменти для застосунку та виконати програмну реалізацію, що має забезпечити безпечне та своєчасне оцінювання якості навчання в Університеті та зручної обробки даних.

Методи розробки: створення програмного продукту шляхом ручного написання коду.

Інструменти розробки: MSSQL Server, Visual Studio Code, Visual Studio 2022, мови програмування C# та TypeScript.

Результати роботи: розроблено архітектуру застосунку збору даних, виконано його програмну реалізацію.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ І ПОЗНАЧЕНЬ

БД – база даних

MUI – Material UI

SQL – Structured Query Language

СУБД – система управління базами даними

CQRS – Command-Query Responsibility Segregation

PGO – Profile-guided optimization

CI – Continuous Integration

SPA – Single Page Application

MSSQL – Microsoft Server SQL

CIL – Common Intermediate Language

ВСТУП

Для того щоб покращувати процеси будь-якої організації необхідно отримувати якісний фідбек від усіх учасників. Якщо в обов'язки викладачів входить давати зворотній зв'язок студентам про рівень їх знань, то студенти хочуть отримувати якісну освіту та можуть подати цікаві ідеї для покращення.

Наш Університет завжди прагнув до покращення якості освіти, тому кожного семестру збирається зворотній зв'язок від студентів про задоволенність якістю викладання дисциплін. Для цього можуть використовуватись різні засоби і форми. В еру цифровізації та на період дистанційного навчання використання роздуківок або залишання вражень на аркуші в клітинку може справляти архаїчне враження. ОСС(органи студентського самоврядування) проводили подібні опитування з використанням гугл-форм.

Для спрощення заповнення даних і забезпечення анонімності постала необхідність у розробці системи оцінювання якості освіти в мережі Університету, чому і буде присвячена дана робота.

РОЗДІЛ 1. ПІДХОДИ ТА СПОСОБИ ЗБЕРЕЖЕННЯ АНОНІМНИХ ДАНИХ

В цьому розділі буде розглянуто та проаналізовано існуючі рішення збору даних без прив’язки до персональних даних.

1.1 Способи забезпечення анонімності

У роботі [1] наведено алгоритм, при якому учасник голосування ідентифікується за допомогою девайсу та особистих даних. Після валідації та вибору учасника його голос кодується та записується на сервері. Такий варіант не зберігає анонімність виборця, порушити таємницю голосування можуть працівники, які мали б доступ до бази, спроектованої таким чином.

Як інструмент для збору зворотного зв’язку від студентів використовувались гугл форми, які налаштовувались учасниками ОСС відповідно до інформації, яку потрібно було збирати для аналізу.

1.2 Способи забезпечення анонімності

Інформаційна захищеність характеризується конфіденційністю, цілісністю та доступністю.[3]

Одним з аспектів інформаційної захищеності є конфіденційність даних, тобто доступ до даних має надаватись лише особи, які мають до них права. Цілісність це уникненням несанкціонованої модифікації даних. Тобто розроблена система має відповідати цим характеристика.

Розглянемо принципи забезпечення захисту даних відомих світових систем.[2] Для прикладу, Google використовує для цього деякі методи анонімізації. Розглянемо два приклади:

- Генералізація даних
- Додавання шуму до даних

Генералізація даних передбачає часткове видалення або заміну загальними значеннями. Генералізація k-анонімності приховує особисті дані у

групі схожих людей. Наприклад, у наборі даних з 50 одиниць є властивість факультет, на кожного учасника групи припадають ще 49 особі з таким самим значенням цієї властивості, тому лише за цим параметром ідентифікувати особу не можливо.

Диференційна конфіденційність полягає у тому, що результат алгоритму завжди однаковий, тому важко визначити чи інформація про особу є частиною набору даних. Зоб цього досягти до набору додається шум. Але цей спосіб може зробити набір даних менш корисним для аналізу.

В розробленій системі авторизація через тритон дозволяє забезпечити виконання усіх трьох принципів захисту даних. Цілісність досягається в той час, коли студент має авторизуватись для голосування, тобто модифікація може здійснюватися лише таким способом. Конфіденційність досягається авторизацією користувача як адміністратора, який може отримати звіти по збору даних. І доступність досягається шляхом, коли адміністратор може отримати доступ до результатів, в той час як студент має змогу тільки надіслати дані.

В результаті виконаної роботи мають виконуватись усі принципи захисту даних відповідно до описаних вище критеріїв, вимог та реалізацій.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ АНОНІМНОГО ГОЛОСУВАННЯ

2.1.1 Процес голосування

Голосування має доступне тільки студентам, які зареєстровані в системі Тритон. Відповідно до флоу, що зображене на рис. 2.1.1.1, кожен учасник голосування отримає особисте поклікання, попередньо авторизувавшись, на сторінку, в якому вже буде вказана інформація про курс, спеціальність та предмети в поточному семестрі. Для кожної дисципліни має сформуватись форма оцінювання. Вона має сталий список питань. Студент має обирати викладачів, які викладали цей предмет. Існують ситуації, коли необхідно обрати кількох викладачів для однієї дисципліни або не всі вказані викладачі викладали у певного студента, тому тут передбачена функція множинного вибору. Також, є обов’язковим вибір викладача-лектора, проте поле вибору викладача-семінариста можна залишити пустим. Студент має заповнювати форму для кожного викладача за кожним предметом і загальну за предметом. Після заповнення усіх форм студент має натиснути кнопку “Надіслати опитувальник”, його відповіді мають бути збережені для подальшого використання для формування статистики.

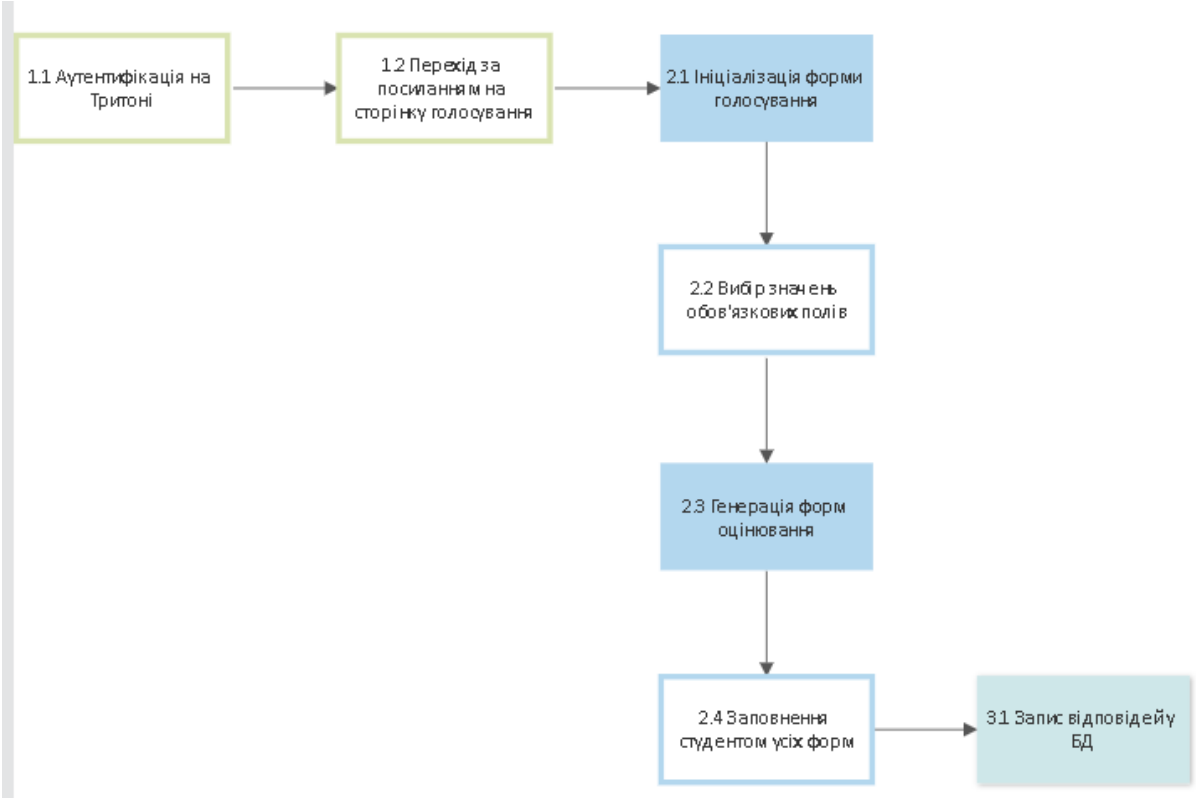

Рис 2.1.1.1

2.1.2 Обробка даних та статистика

Для зручного користування результатами буде передбачена функція вигрузки даних у форматі .xls, яка на зараз реалізована у вигляді ендпоінта API. Основними вимогами до обробки є можливість застосувати фільтри, такі як «Дані за Викладачем», «Результати за предметом» та «Результати семестру», або комбінація кількох фільтрів. Реалізацію цієї вимоги також було враховано при проектуванні БД.

2.2 Вибір інструментів

Основним середовищем розробки є середовище Visual Studio 2022. Програмний продукт розробляється по технології ASP.NET Core MVC на платформі .NET 6, для створення зовнішнього вигляду та інтерактивності веб-сторінок використовується джаваскріптова бібліотека React та MUI.

2.2.1 Платформа .NET 6

На сьогодні мова програмування C# є однією з найпотужніших і затребуваних мов в IT сфері. В теперішній час з її використанням пишуть велику кількість різних застосунків: від невеликих десктопних додатків до найбільших веб-порталів і веб-сервісів, якими користуються мільйони людей кожного дня.

Поточною версією мови є версія C# 10, яка вийшла 8 листопада 2021 року разом з релізом .NET 6. Платформа .NET 6 дає три нові інструменти:

Попередня компіляція – нова технологія Crossgen2 розбирає IL-код , створює деякий граф додатку. Після чого запускає всередині себе JIT-компілятор для необхідної платформи, а цей компілятор аналізує створений граф і створює нативний код, застосовуючи при необхідності різні оптимізації. Оптимізація на основі профілювання – в першу чергу, це через додаток пропускають дуже велику кількість різних кейсів. Результати профілюються. Після чого результати профілювання аналізуються компілятором і він починає розпізнавати найбільш використовуваний код. Це так звана PGO-оптимізація та має ще кілька підходів в реалізації: розділення на часто та рідко використовуваний код. Частини, які використовуються найчастіше групуються та поміщаються поряд з результуючим бінарним файлом.

- Динамічна PGO. В цьому випадку усі етапи навчання JIT-компілятора пропускаються, замість цього він уважно дивиться як додаток працює в реальному середовищі.
- Поєднання динамічного та статичного підходів. Обидві вище описані техніки використовуються одночасно, тому результуючий бінарний файл може частково містити нативний код для швидкого запуску, який потім може бути оптимізований ще більше.

Гаряче перезавантаження додатку – зручна фіча для розробника, що дозволяє вносити зміни в код і застосувати їх одразу до працюючого додатку.

.NET 6 демонструє очевидні переваги, в першу чергу тому, що більше немає розділення на .NET Framework, який використовувався тільки для розробки під Windows та кросплатформенний .NET Core.

2.2.2 Мапер Dapper

Dapper є популярним і зручним інструментом мапінгу. В першу чергу він був створений для тих сценаріїв, де ми хочемо працювати з даними в строгому вигляді – як в нашому випадку, об’єкти бізнес логіки в .NET додатках, але якщо ви не хочете витрачати багато годин на мапінг результатів запитів з ADO.NET. Dapper є open source проєктом з використанням Apache license. Доступно як пакет NUGET.

2.2.3 Бібліотека React

React — це бібліотека JavaScript, зосереджена на створенні декларативних інтерфейсів користувача (UI) з використанням концепції на основі компонентів. Він використовується для обробки шару перегляду та може використовуватися для веб- та мобільних додатків.

React - це не фреймворк, це конкретна бібліотека. Пояснення цього полягає в тому, що React займається лише візуалізацією інтерфейсів і зберігає багато речей на розсуд окремих проєктів. Стандартний набір інструментів для створення програми за допомогою ReactJS часто називають стеком.

React надає декларативний API, тому вам не доведеться турбуватися про те, що саме зміниться під час кожного оновлення. Це значно полегшує написання програм, але може бути неочевидним, як це реалізовано в React. У цій статті пояснюється вибір, який ми зробили в алгоритмі «розрізнення» React, щоб оновлення компонентів були передбачуваними і при цьому були достатньо швидкими для високопродуктивних програм.

2.2.4 Мова програмування Typescript

Typescript це об’єктно орієнтована мова програмування розроблена Майкрософтом, в той час як Javascript це скриптова мова програмування для вебу. Typescript це мова з відкритим вихідним кодом для створення масштабованих програм, в той час як Javascript це мова для створення інтерактивних сторінок на стороні веб-сервера.

TypeScript додає до JavaScript додатковий синтаксис для підтримки більш тісної інтеграції з вашим редактором. Tupescript дозволяє виявити помилки на початку написання коду. Код TypeScript перетворюється на JavaScript, який запускається скрізь, де виконується JavaScript: у браузері, на Node.js або Deno і подібіні. TypeScript розуміє JavaScript і використовує висновок типу, щоб надати вам чудові інструменти без додаткового коду.

2.2.5 Реакт компоненти MUI

MUI – це компоненти React для швидшої та простішої веб-розробки. MUI працюють ізольовано. Вони є самопідтримуваними і вводять лише ті стилі, які їм потрібні для відображення. Вони не покладаються на жодні глобальні таблиці стилів, такі як normalize.css. Ця бібліотека дозволяє імпортувати та використовувати різні компоненти для створення інтерфейсу користувача в програмах React. Це значно економить час, оскільки розробникам не потрібно писати все з нуля.

2.2.6 СУБД MSSQL

Microsoft SQL Server – система управління базами даних, яка розробляється Microsoft. Основна мова запитів - Transact-SQL, створений разом Microsoft та Sybase. Transact-SQL є реалізацією стандарту ANSI/ISOпо структурованій мові запитів. Використовується для роботи з базами даних розміром від особистих до великих БД масштабу підприємств. Виконує функції

по збереженню та наданню даних у відповідь на запити інших застосунків, які можуть виконуватись на тому самому сервері, так і в мережі.

2.3 Підходи реалізації

2.3.1 SPA

SPA - це архітектура односторінкового застосунку, який взаємодіє з користувачем динамічно переписуючи поточну веб-сторінку новими даними з сервера. Мета такого підходу швидкі переходи, завдяки яким веб-сторінка буде більше схожа на нативну програму. У SPA оновлення сторінки ніколи не відбувається, весь необхідний код HTML, CSS JavaScript підтягується браузером із завантаженої сторінки або відповідні динамічні ресурси завантажуються та додаються на сторінку за потреби.

2.3.2 Мікросервіси та багаторівнева архітектура

Мікросервісна архітектура є дуже популярною у сучаних вистемах. Це не дивно, адже вона має дуже багато переваг.

- Масштабованість. Кожен сервіс можна запускати у великій кількості екземплярів. Якщо одна частина є високонавантаженою, то саме це місце можна розмножити і горизонтально масштабувати.
- Кожен мікросервіс можна розгорнути незалежно один від одного. У випадку великого домену рішення можна розвивати паралельно.
- Кожен мікросервіс окремо є набагато простішим рішенням, ніж моноліт в цілому. Якщо конкретний мікросервіс перетає задовольняти вимоги бізнесу, його можна переписати або викинути. Це буде не настільки дорого і довго, як переписати цілу систему-моноліт.

- В мікросервісній архітектурі є можливість застосовувати різні технології.

Але, звісно, є і недоліки:

Кількість зусиль, що має бути витрачена на вирішення одних і тих самих задач. Кожен сервіс має свій CI, своє інтеграційне тестування , що ускладнює деякі задачі, наприклад при зміні розгортання рішення його треба буде змінити для кожного мікросервісу.

Сама система мікросервісів є складнішою ніж моноліт, необхідно кожного разу перевіряти контракти, що накладає необхідність інтеграційного тестування.

Не менш важливим є цілісність контрактів і даних. Кожен мікросервіс працює зі своїм сховищем, дані не інтегровані між собою: немає зовнішнього ключа, і, відповідно набагато складніше зберігати цілісність даних.

Оскільки кожен із сервісів можна розробляти незалежно, так само він може незалежно змінити контракти. В цьому випадку добре якщо це буде помічено під час інтеграційного тестування, в найгіршому випадку це буде помічено в production.

Як висновок, в даному випадку мікросервісна архітектура є нерелевантно складною. Проте система є невеликою і має лише один сервіс, тому в майбутньому при потребі масштабування не має викликати серйозних проблем.

2.3.3 Будівельник

Будівельник — це породжувальний патерн проектування, що дає змогу створювати складні об’єкти крок за кроком. Будівельник дає можливість використовувати один і той самий код будівництва для отримання різних відображень об’єктів. В реалізації сервісу даний патерн використовується для побудування запитів до БД. Як приклад, у додатку В наведено лістинг коду SubjectSqlBuilder, який виконує єдину функцію отримання всіх предметів, які

існують на цьому факультеті. В майбутньому цей запит може бути розширено відповідно до потреб, наприклад, отримати всі предмети для окремої спеціальності.

2.3.4 Репозиторій

За визначенням, шаблон проектування репозиторію в C# є посередником між доменом і шарами відображення даних за допомогою інтерфейсу, схожого на колекцію, для доступу до об’єктів домену. Шаблон проектування репозиторію відокремлює логіку доступу до даних і зіставляє її з об’єктами бізнес-логіки. Він працює з об’єктами домену та виконує логіку доступу до даних. У шаблоні Repository сутності домену, логіка доступу до даних і бізнес-логіка спілкуються один з одним за допомогою інтерфейсів. Він приховує деталі доступу до даних від бізнес-логіки.

До переваг шаблону проектування Репозиторій відносять:

- Тестування контролерів стає простим, тому що платформа тестування не обов’язково повинна працювати з фактичним кодом доступу до бази даних.

- Шаблон проектування репозиторію відокремлює фактичну базу даних, запити та іншу логіку доступу до даних від решти програми.

- Бізнес-логіка може отримати доступ до об’єкта даних, не володіючи знаннями про базову архітектуру доступу до даних.

- Бізнес-логіка не знає, чи використовує програма LINQ to SQL чи ADO.NET. У майбутньому базові джерела даних або архітектуру можна буде змінити без впливу на бізнес-логіку.

- Централізована логіка доступу до даних, що полегшує підтримку коду.

Приклад реалізації VoteRepository наведено в Додатку Г.

2.3.5 Асинхронність

Всі зовнішні пристрої, що не працюють на одній шині з мікропроцесором, - мережеві адаптери, відеокарти, сховища даних - повертають результат своєї роботи не відразу. Тобто, при велекій кількості хаписів у БД, сайт може просто зависнути. Отже, потрібно вибирати: або наш потік виконання буде зупинятися і очікувати на результат операції, або виконувати якийсь інший код. Таким чином, код, написаний з неблокуючим (асинхронним) очікуванням на результат, споживає менше ресурсів і є більш продуктивним на дистанції. Для вирішення цієї задачі я використала async/await. У додатку Д наведено приклад використання цих інструментів у FormRepository, який має кілька зпитів до бази для формування даних форми, тому для цього запиту є критичним використання асинхронності.

2.3.6 HTTP

HTTP — широко поширений протокол передачі даних, спочатку призначений передачі гіпертекстових документів (тобто документів, які можуть містити посилання, дозволяють організувати перехід до інших документів).

Абревіатура HTTP розшифровується як HyperText Transfer Protocol, «протокол передачі гіпертексту». Відповідно до специфікації OSI, HTTP є протоколом прикладного (верхнього, 7-го) рівня. Актуальна на даний момент версія протоколу HTTP 1.1 описана в специфікації RFC 2616.

Протокол HTTP передбачає використання клієнт-серверної структури передачі. Клієнтська програма формує запит і відправляє його на сервер, після чого серверне програмне забезпечення обробляє цей запит, формує відповідь і передає його клієнту. Після цього клієнтська програма може продовжити надсилати інші запити, які будуть оброблені аналогічним чином. Завдання, яке традиційно вирішується за допомогою протоколу HTTP - обмін даними між додатком користувача, що здійснює доступ до веб-ресурсів (зазвичай це веб-

браузер) і веб-сервером. На даний момент завдяки протоколу HTTP забезпечується робота Всесвітньої павутини.

На фронтенді було використано HTTP клієнт - axios. Лістинг прикладу запиту наведено у додатку Є. На бекенді ендпоінти для того щоб розпізнати на який метод приймати запити використовуються спеціальні атрибути [HttpGet({id})], який в результаті приймає всі запити Get за адресою {domain}/<ControllerName>/api/{id}. Ці атрибути знаходяться у сторонньому пакеті Microsoft.AspNetCore.Mvc.

РОЗДІЛ 3: РЕАЛІЗАЦІЯ ВЕБ СЕРВІСУ

3.1 Взаємодія компонентів веб сервісу

Кінцевий користувач має унікальне поклікання в особистому кабінеті на Triton.Students, після переходу за яким генерується сторінка для заповнення. Поля Освітній рівень, Освітня програма та Рік навчання заповнюється автоматично для користувача. Ця інформація надходить у тілі запиту POST зі сторони тритону. Нижче користувач бачить три обирайки для вибору предмету оцінювання, викладача лекційних і семінарських занять. Після вибору значень для всіх полів становиться активною кнопка “Далі”. При кліку на неї користувач отримує сторінку з формою опитування для лектора(-ів) за цим предметом, для якого з фронтенду йде запит на FormController, який вертає актуальні запитання в залежності від типу форми. Після заповнення всіх питань заданої форми внизу сторінки становиться активною кнопка до наступної форми, при переході генерується наступна форма, це може бути форма для іншого лектора або семінариста. За такою ж логікою генеруються наступні форми. Клік на кнопку після заповнення останньої форми надсилає результати на VotesController, який звертається до репозиторію, що записує результат в базу і при успіху вертає код 200. В кінці опитування з’являється остання сторінка з подякою за участь в опитуванні. Як взаємодіють компоненти системи розглянуто на діаграмі в пункті 3.2.2

3.2 Реалізація бекенду

3.2.1 Проєктування БД

В процесі проектування була побудована наступна діаграма:

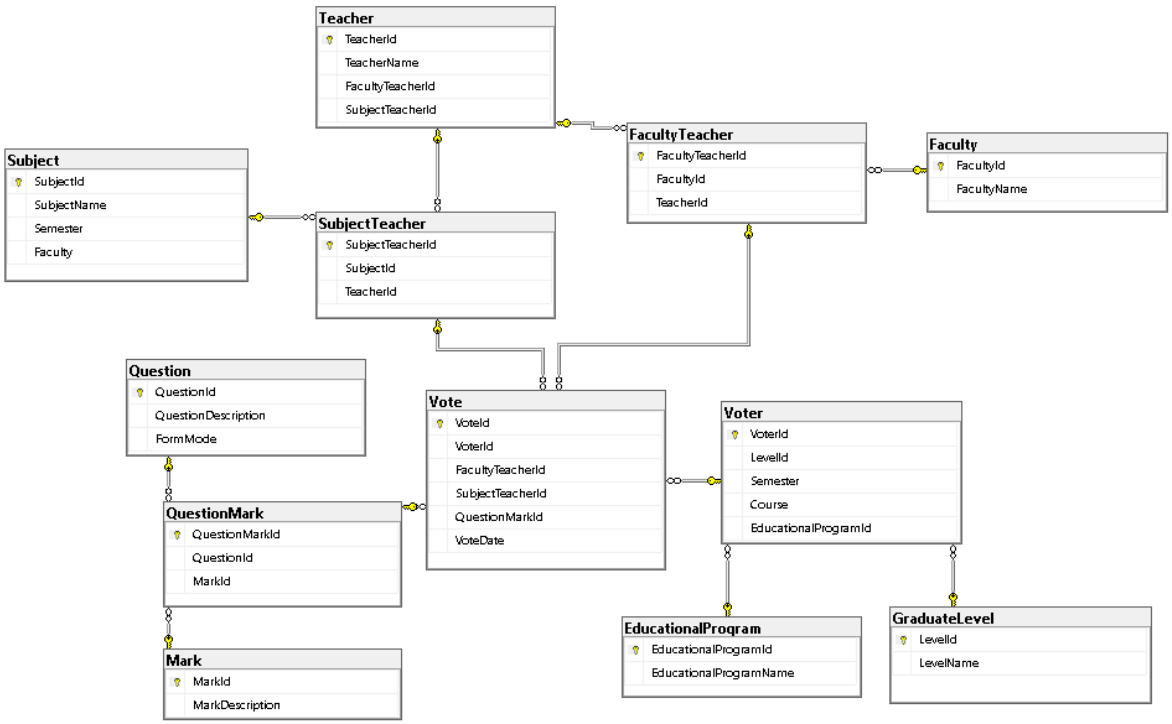

Рис. 3.2.1.1 ER-діаграма БД

Згідно до вимог абсолютно очевидно, що «Голос» має бути анонімним, тому при проектуванні було вирішено не передавати до системи персональні дані студентів, тоді як генерувати псевдорандомний ідентифікатор на стороні Тритону. Сутність «Vote» є головною сутність, яка зберігає дані для аналізу. У пункті 2.3 було розглянуто необхідність експорту результатів, зі спроектованою моделлю є нескладним отримання статистики відповідно до фільтру, до прикладу запит на вибірку статистики відповідно для Викладача.

3.2.2 Взаємодія програмних компонентів

На діаграмі нижче наведено флоу запису результатів опитування. Спочатку запит користувача формується на фронтенді та по HTTP надходить до контролера, який надсилає запит до репозиторію. Репозиторій агрегує білдер сіквел запитів, який в свою чергу по CRUD надходять до бази даних.

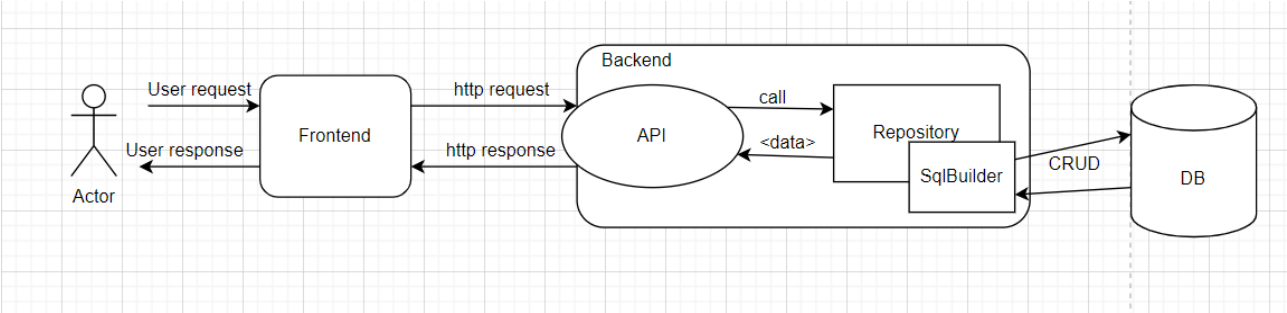

Рис 3.2.2.1

Нижче представлена на малюнку 3.1.5 UML діаграма, яка описує взаємодію серверної частини сервісу. ConcreteController є реалізацією абстрактного базового контролеру з пакету Microsoft.AspNetCore.Mvs.Core. Реалізації представлені на рисунку 3.1.6 та в додатку Б наведено лістинг коду FormController. Кожна реалізація контролера агрегує в собі конкретний репозиторій, який здійснює роль посередника та має доступ до бази даних. Кожен репозиторій унаслідується від BaseRepository, який містить базову інформацію для всіх реалізацій, таку як мапер, стрінга підключення та метод отримання моделей по ідентифікатору. В свою чергу кожен репозиторій має конкретну реалізацію BasicSqlBuilder.

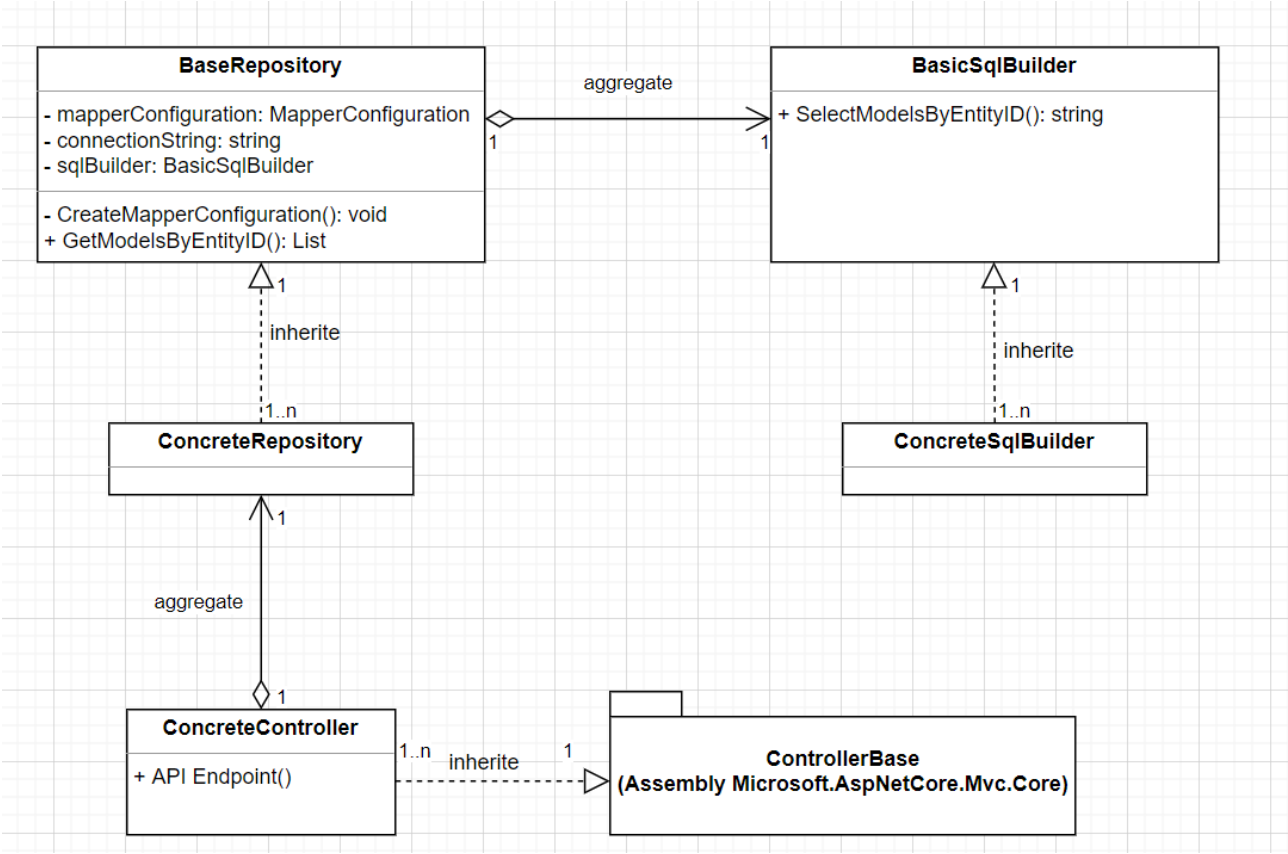

Рис. 3.2.2.2 Структура сервісної частини

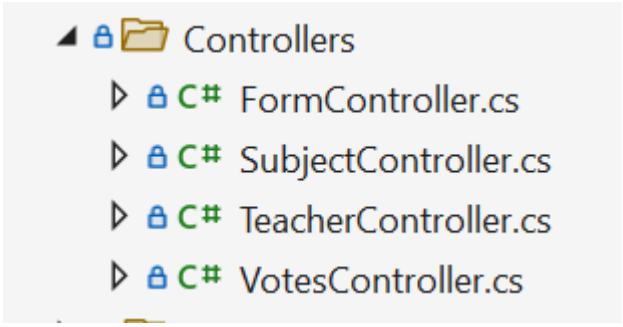

Рис 3.2.2.3 Реалізації контролерів

API проєкту містить 4 контролери та клас Program, так як в .NET 6 розробники відмовились від методу Main.

На рисунку 3.2.2.4 зображено компонентну діаграму взаємодії. Вона демонструє відношення між компонентами системи, опускаючи деталі специфікації. Triton.Stidents - це компонент, який використовується в сервісі авторизації на бекенді системи опитування. Survey Front-End та Survey Back-End обмінюються запитами за допомогою API. Controller в свою чергу використовую репозиторій,

а репозиторій приймає послугу від Sql Builder’а.

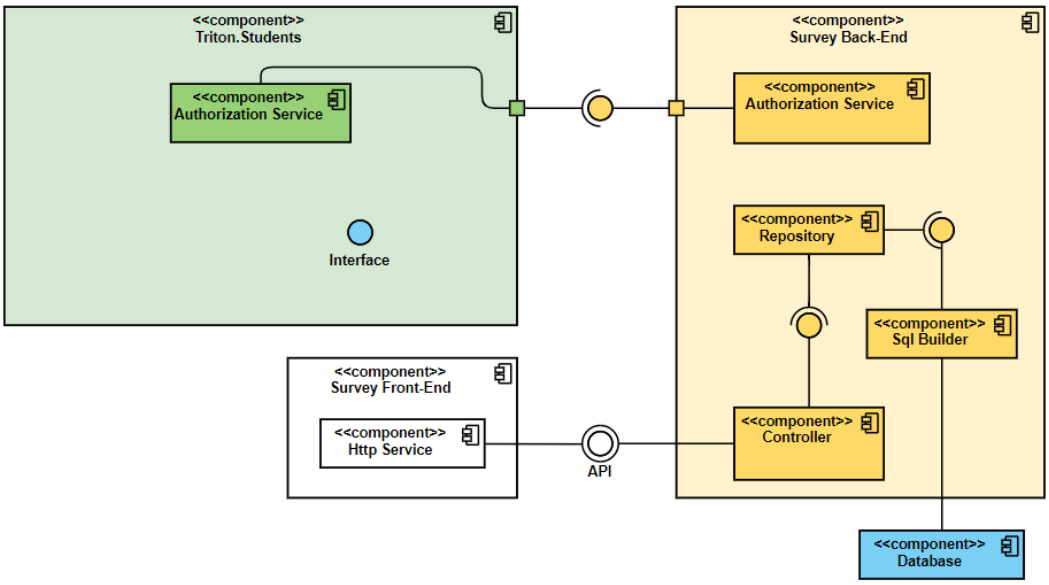

Рис 3.2.2.4

3.2.3 Фільтрація даних

Одна з вимог є вибірка даних відповідно до фільтру. Для цього є необхідним інтеграція з системою тритон. На даному етапі інтеграцію не було проведено. Додано ендпоінт, який повертає дані про викладача відповідно до заданого фільтру. Майбутня інтеграція має лише посилати запит на цей ендпоінт та реалізовувати функцію вивантаження даних у файл. Додано приклади запитів фільтрування даних. У додатку А додано лістинг коду для вибору статистики відповідно до викладача.

3.2.4 OPEN API

Open API, яка також називається public API це інтерфейс прикладного програмування, доступний для розробників програмного забезпечення. Open API публікуються в Інтернеті та вільно поширюються, що дозволяє надавати користувачам універсальний доступ. На рисунку наведено приклад специфікації swagger для VotesController. Ця специфікація є дуже корисною для інтеграції з Тритоном, так як не обов’язково лізти в контролер та дивитись яку модель він приймає.

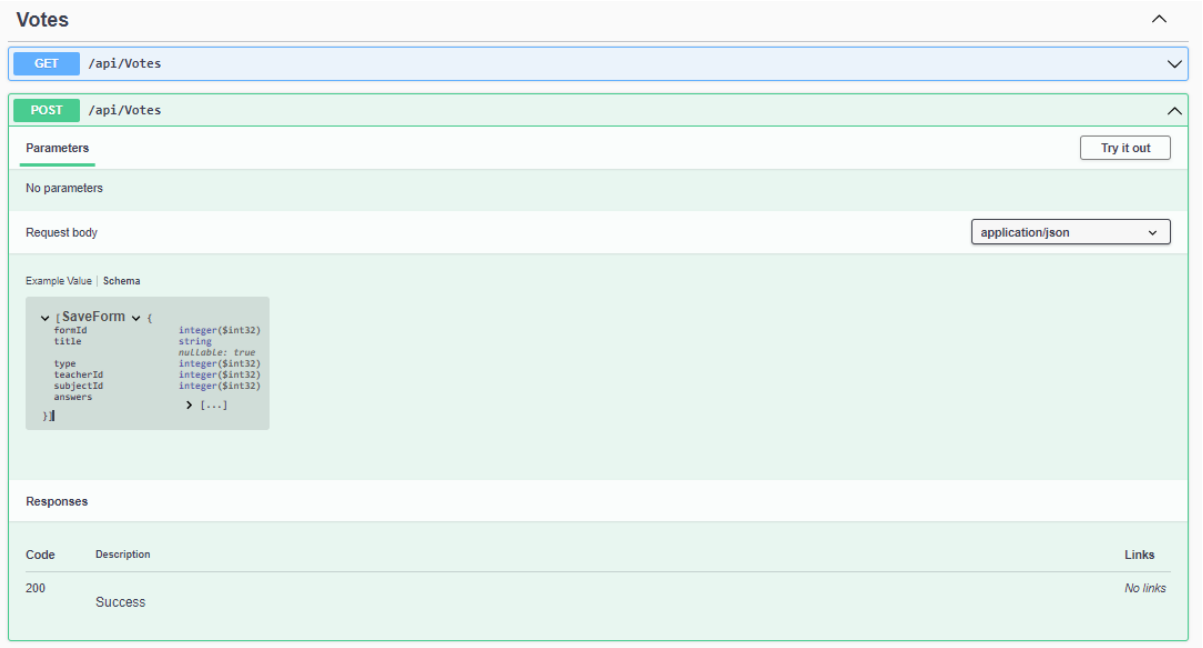

Рис 3.2.4.2

3.3 Реалізація frontend

3.3.1 Архітектура

Вся фронтенд частина розділена на контейнери. Для зручності використання і регулювання ними використовується state. App відповідає за комунікацію контейнерів. Є кілька контейнерів, які розділені за сферою відповідальності. Контейнери відповідають за логіку відображення і надсилають запити до API.

Контейнер Greeting container з’являється першим і відповідає за сторінку привітання, де університет дякує за участь в опитуванні та наголошує на його важливості. Наступний Common container, який відповідає за вибір предмету і викладачів, на базі якого буде формуватись надалі форма. На базі нього збирається ця інформація та надсилається до Form container. Form container відповідає за генерацію опитувальника і збереження цієї інформації.

Між формами можна переходити в рамках однієї сесії, так як відповіді зберігаються у стейті. Після заповнення всіх форм формується запит на збереження відповідей та надсилається до API. Нижче наведена діаграма-

архітектура компонентів реалізації фронтенд частини та структура файлів 3.3.2 та 3.3.3 відповідно

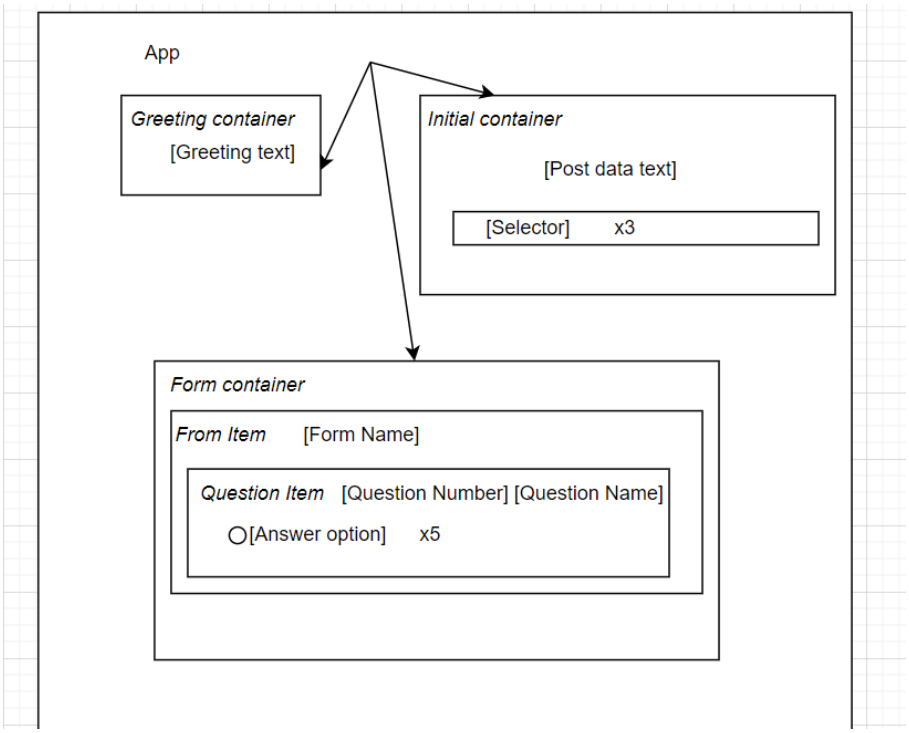

Рис 3.3.1.1

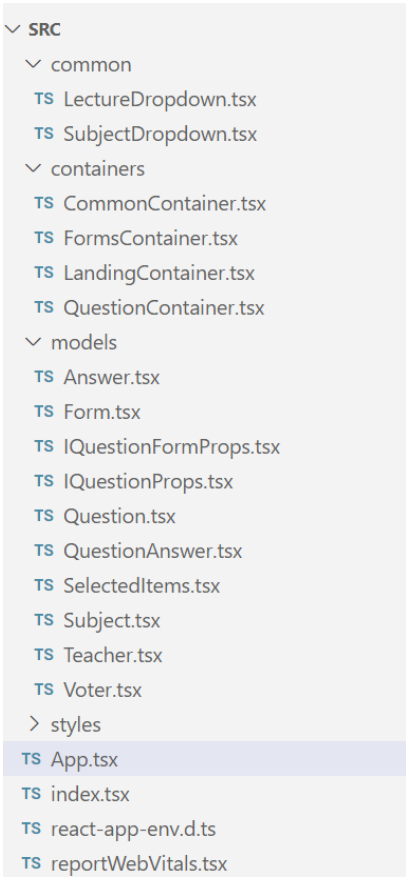

Рис 3.3.1.2

3.3.2 Користувацький інтерфейс

При переході на першу сторінку голосування користувач обирає предмет та викладачів, яких хоче оцінити.

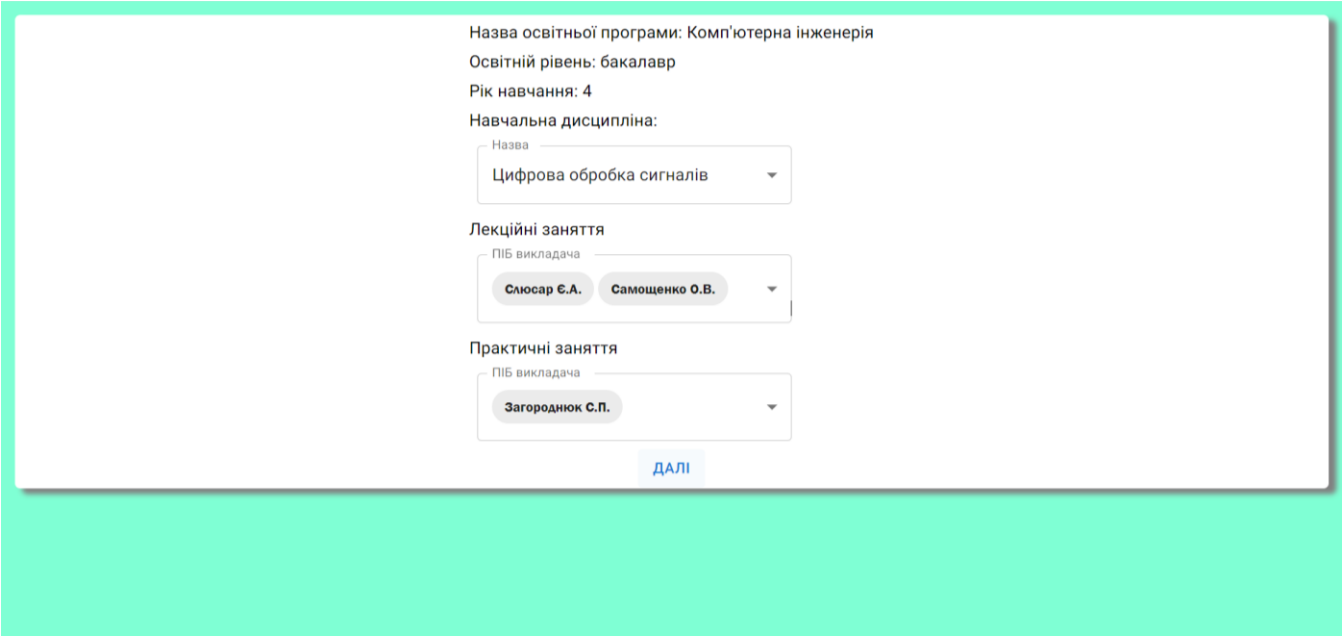

Рис 3.3.2.1

Відповідно до його запиту після натискання кнопки “Далі” генеруються форми з питаннями. Так як студент вибрав двох лекторів та семінариста, буде згенеровано 4 форми. Візуально вони мають ідентичний інтерфейс, відрізняються лише наповненням питань.

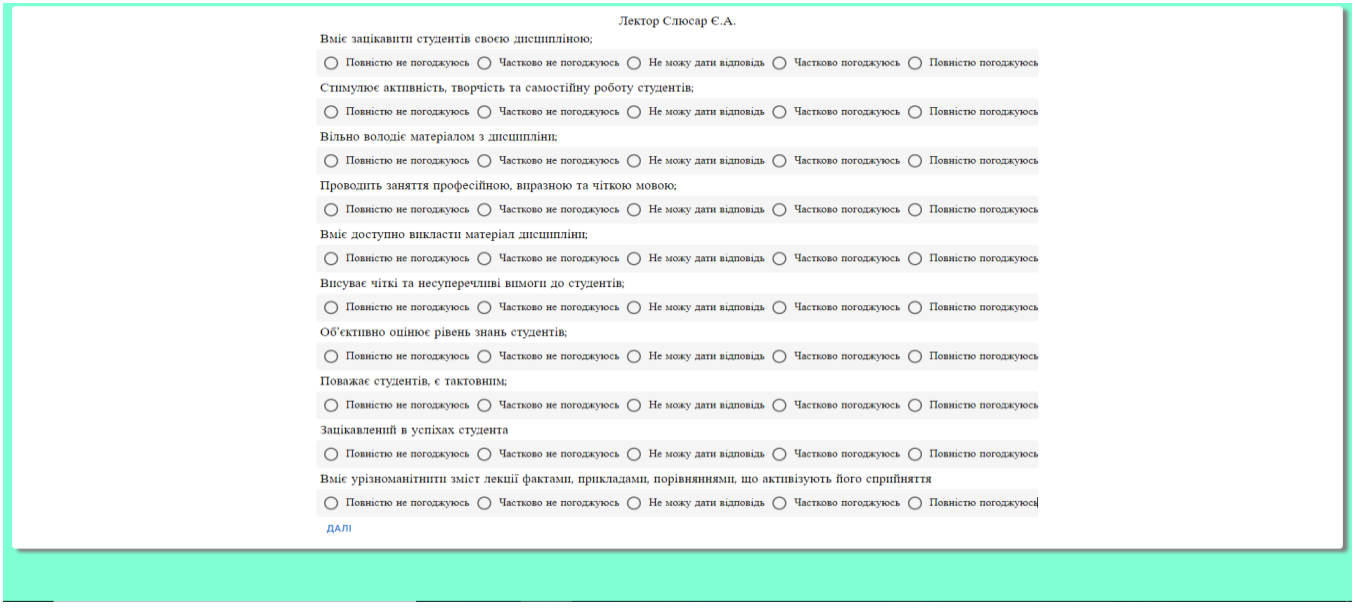

Рис 3.3.2.2

При менших розмірах екрану, як зображено на рисунку форма підлаштовується під користувача, що дозволить проходити опитування з мобільних пристроїв. Це було досягнуто з використанням властивості ‘flex-wrap: wrap’. При зменшенні екрану вона переносить контейнери у зручному порядку. Приклад наведено на рисунку 3.3.2.3.

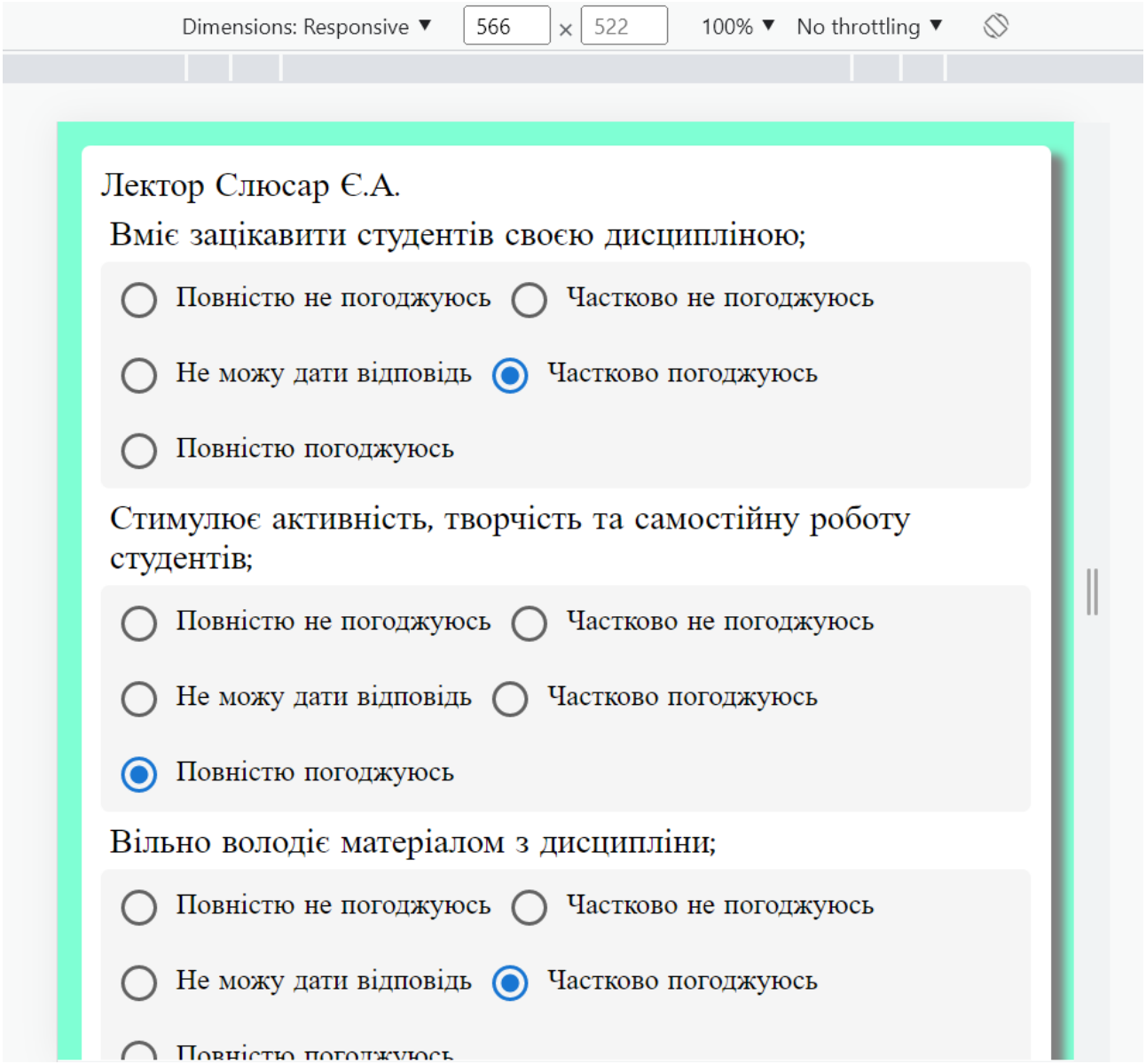

Рис 3.3.2.3

Остання сторінка, яка з’являється після заповнення всіх форм містить текст подяку. Всі результати, які були вибрані в результаті опитування зберігаються в БД. Варто зазначити, що на фронтенді присутня валідація, яка не дозволяє приступити до наступної форми або закінчити оцінювання, якщо дана відповідь не на всі питання.

3.4 Тестування API

Одним із важливих етапів розробки є етап тестування. Тестування API — це набір дій із забезпечення якості, які включають надсилання викликів до API, отримання вихідних даних та перевірку відповіді системи на визначені вхідні параметри, зокрема, точність даних і формату даних, коди статусу HTTP та коди помилок.

Зазвичай тестування API виконується на API, вироблених власною командою розробників. Ми не тестуємо сторонні API, але ми можемо перевірити, як наше програмне забезпечення приймає їхні запити.

Підхід до тестування API значною мірою залежить від типу API. Існують веб-API, або веб-сервіси, API бази даних, які з'єднують програми з системами керування БД, API операційних систем і віддалені API для доступу до ресурсів, розташованих за межами пристрою, який їх запитує.

В цьому підрозділі описано тестування власного API.

3.4.1 Успішний сценарій

Успішний сценарій характеризується поверненням списку потрібних об'єктів відповідно до запиту. На рисунку наведено скріншот успішного виконання запиту до розробленого API з платформи для побудови і використання API запитів Postman. На рисунку 3.4.1.2 наведено результат рендеру сторінки на основі повернених даних з малюнку 3.4.1.1.

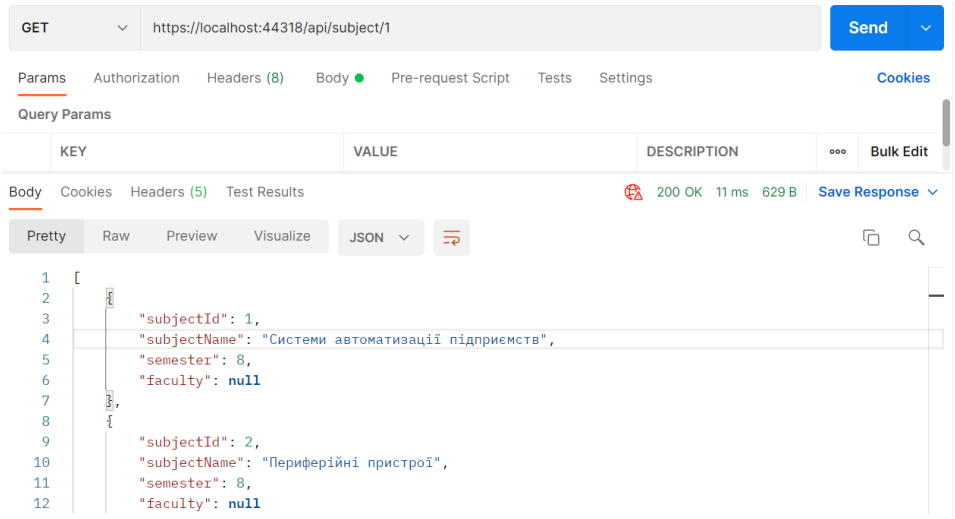

Рис 3.4.1.1

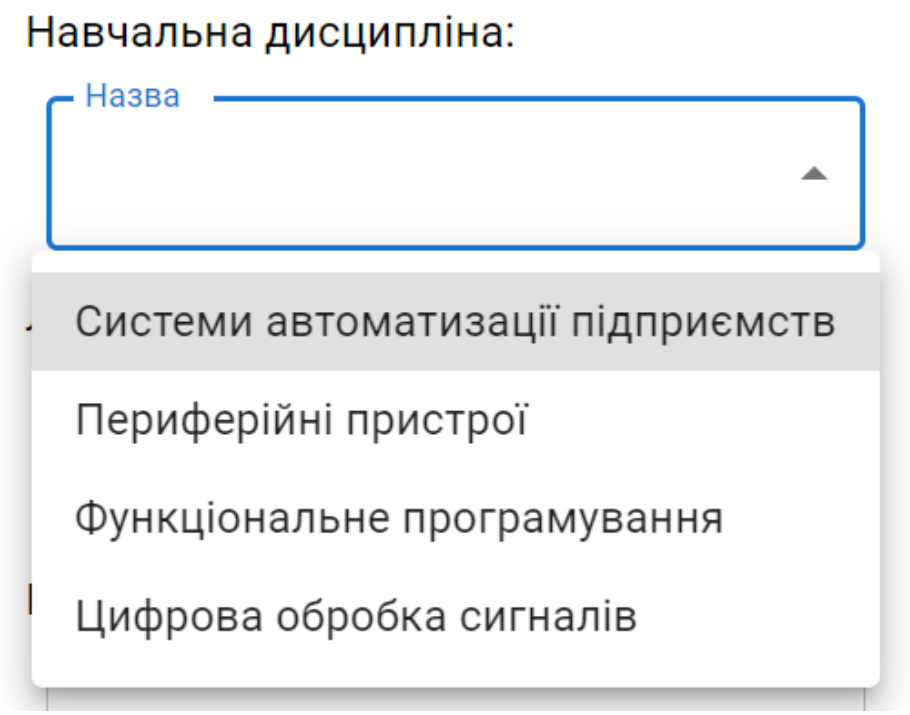

Рис 3.4.1.2

3.4.2 HTTP статус коди

На рівні контролеру було реалізовано обробку виключень. На рисунку 3.4.2.1 зображено результат обробки виключення, коли у базі не знайдено значення з відповідним ідентифікатором та повертає код помилки 404. На рисунку 3.4.2.2 зображено результат обробки невалідного запиту. Для цього використано інтерфейс IActionResult та методи класу ControllerBase, які вертають відповідні виключення. Код обрбонику наведено у додатку Е. На скріншотах показано повертання коректних статус кодів та повідмолення про помилку.

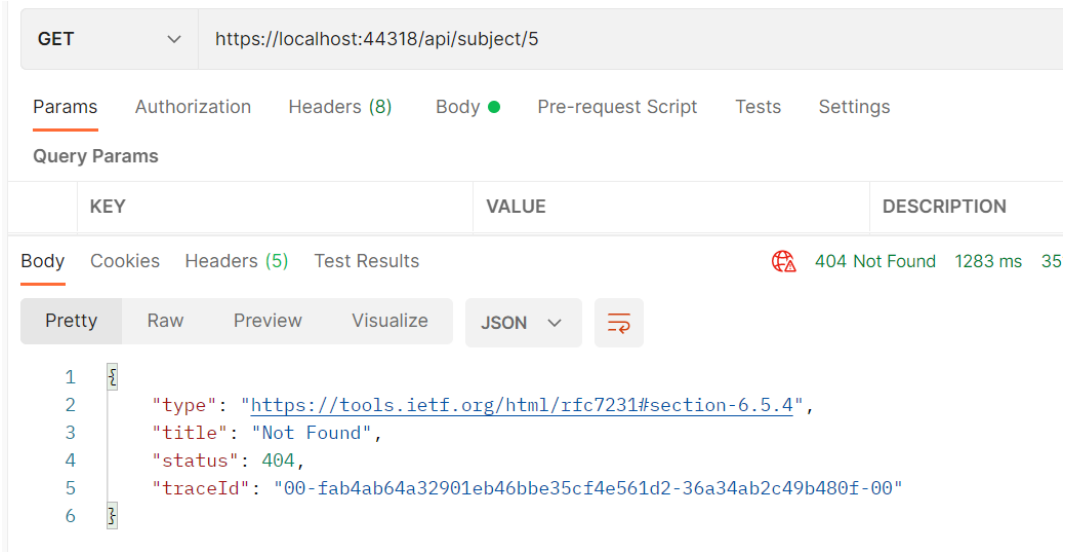

Рис 3.4.2.1

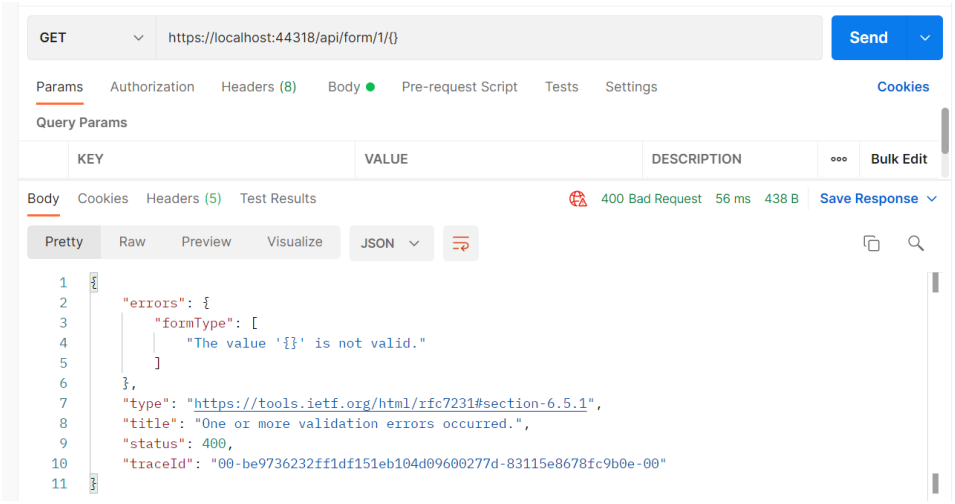

Рис 3.4.2.1

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було створено архітектуру, обрано інструменти реалізації, розроблено структуру БД для Анонімної Системи Оцінювання Якості Освіти. Використано архітектуру системи, яка включає набір спеціалізованих підсистем та базу даних, концептуальна модель якої сформована на основі методики побудови реляційних баз даних. Обрано засоби розробки, до яких відносяться мова програмування C#, бібліотеки React та MUI, пакет Dapper, які дозволяють зручно розроблювати та підтримувати систему, а також додавати нові компоненти. Потенційними користувачами системи є студенти, як учасники голосування, та аналітики, які будуть оброблювати результати оцінювання.

Було спроєктовано архітектуру відповідно до вимог та виконано програмну реалізацію веб-сервісу згідно розробленої архітектури, що забезпечує збір даних у студентів стосовно якості викладання в університеті, який можна інтегрувати в існуючі системи. Сервіс дозволяє також аналізувати дані.

Подальший розвиток системи передбачає:

- Програмну інтеграцію з системою Тритон
- Інтеграцію Power BI для графічного представлення аналізованих даних

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Design and implementation of an E-Voting system https://www.researchgate.net/publication/340234262_DESIGN_AND_IMPLEMENTATION_OF_AN_E-VOTING_SYSTEM

2. Як Google анонімізує дані https://policies.google.com/technologies/anonymization?hl=uk

3. Суттєві позиції з безпеки властивостей інформації https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%85%D0%B8%D1%81%D1%82_%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D1%97#%D0%A1%D1%83%D1%82%D1%82%D1%94%D0%B2%D1%96_%D0%BF%D0%BE%D0%B7%D0%B8%D1%86%D1%96%D1%97_%D0%B7_%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D0%BA%D0%B8_%D0%B2%D0%BB%D0%B0%D1%81%D1%82%D0%B8%D0%B2%D0%BE%D1%81%D1%82%D1%96_%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D1%97

4. Builder - патерн проектування https://refactoring.guru/uk/design-patterns/builder

5. Repository - патерн проектування https://habr.com/ru/post/248505/

6. Нові можливості .NET 6 https://docs.microsoft.com/ru-ru/dotnet/core/whats-new/dotnet-6

7. WHat is Survey? http://www.rickweil.com/s2211/whatisasurvey.pdf

8. Простое руководство диаграмы компонентов https://creately.com/blog/ru/uncategorized-ru/%D1%83%D1%87%D0%B5%D0%B1%D0%BD%D0%BE%D0%B5-%D0%BF%D0%BE%D1%81%D0%BE%D0%B1%D0%B8%D0%B5-%D0%BF%D0%BE-%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D0%BD%D0%BE%D0%B9-%D0%B4%D0%B8%D0%B0%D0%B3/

9. Component Diagram https://sites.google.com/site/prattshomepge/home/uml/component-diagram

10. Happy path testing https://www.techtarget.com/searchsoftwarequality/definition/happy-path-testing

11. API testing https://www.altexsoft.com/blog/api-testing/

ДОДАТКИ

ДОДАТОК А

```
SELECT
ep.EducationalProgramNa
me,

                        gl.LevelName,

                        v.Course,

                        s.SubjectName,

                        t.TeacherName,

                        q.QuestionDescription,

                        v.MarkId

                        FROM

                        Vote v join EducationalProgram ep on v.EducationalProgramId
                        = ep.EducationalProgramId

                        join GraduateLevel gl on v.LevelId = gl.LevelId

                        join Subject s on v.SubjectId = s.SubjectName

                        join Teacher t on v.TeacherId = t.TeacherName

                        join Question q on v.QuestionId = q.QuestionId

                        WHERE

                        v.VoteDate <= '2021-12-01' AND v.VoteDate >= '2021-11-22'

                        AND t.TeacherName = 'Слюсар Є.А.'
```

ДОДАТОК Б

```
using Microsoft.AspNetCore.Mvc;

using System;

using System.Threading.Tasks;

using VotingProcess.Models;
```

```
using VotingSystemBackend.Models;
using VotingSystemBackend.Repositories;
namespace VotingSystemBackend.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class FormController : ControllerBase
    {
        private readonly FormRepository<FormDto, Form> formRepository = new FormRepository<FormDto, Form>();
        [HttpGet("{id}")]
        public async Task<IActionResult> GetFormByTeacherId(int id, int formType)
        {
            try
            {
                return new JsonResult(await formRepository.GetFormsByTeacherID(new Tuple<int, int>(id, formType)));
            }
            catch(Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }
    }
}
```

ДОДАТОК В

```
namespace VotingProcess.Extensions
{
    public class SubjectSqlBuilder : BasicSqlBuilder
```

```
    {
        public override string SelectModelsByEntityID(int entityID)
        {
            return $"SELECT * FROM [Subject] WHERE FacultyId = {entityID}";
        }
    }
}
```

ДОДАТОК Г

```
using Dapper;
using System.Data.SqlClient;
using VotingProcess.Extensions;
using VotingProcess.Models;
using VotingSystemBackend.Models;

namespace VotingSystemBackend.Repositories
{
    internal class VotesRepository<TEntity, TModel> : BaseRepository<TEntity, TModel> where TEntity : VoteDto where TModel : Vote
    {
        internal VotesRepository()
        {
            sqlBuilder = new VoteSqlBuilder();
        }

        public void SaveVote(Vote vote)
        {
            using (var connection = new SqlConnection(connectionString))
            {
```

```
            connection.Execute(new
VoteSqlBuilder().InsertVote(GetMapper.Map<VoteDto>(vote)));
        }
    }
  }
}
```

ДОДАТОК Д

```
using Dapper;
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;
using System.Threading.Tasks;
using VotingProcess.Extensions;
using VotingProcess.Models;
using VotingSystemBackend.Models;

namespace VotingSystemBackend.Repositories
{
  internal class FormRepository<TEntity, TModel> : BaseRepository<TEntity,
TModel> where TEntity : FormDto where TModel : Form
  {
    internal FormRepository()
    {
      sqlBuilder = new FormSqlBuilder();
    }

    public async Task<List<Form>> GetFormsByTeacherID(Tuple<int, int>
parameters)
```

```
        {
            IEnumerable<QuestionDto> questions;
            var q = new List<Question>();
            var t = new TeacherDto();
            var teacherSqlScript = (sqlBuilder as FormSqlBuilder).SelectTeacherByTeacherID(parameters.Item1);
            var questionsSqlScript = (sqlBuilder as FormSqlBuilder).SelectQuestionsByFormMode(parameters.Item2);
            using (var connection = new SqlConnection(connectionString))
            {
                questions = await connection.QueryAsync<QuestionDto>(questionsSqlScript);
                t = await connection.QueryFirstOrDefaultAsync<TeacherDto>(teacherSqlScript);
            }
            var forms = new List<Form>();
            questions.ToList().ForEach(dto => q.Add(GetMapper.Map<Question>(dto)));
            forms.Add(new Form
            {
                Title = t.TeacherName,
                Type = questions.FirstOrDefault().FormMode,
                TeacherId = t.TeacherId,
                Questions = questions.ToList()
            });
            return forms;
        }
    }
}
```

ДОДАТОК Е

```
[HttpGet("{id}")]
        public async Task<IActionResult> GetSubjectsByFacultyId(int id)
        {
            try
            {
                var result = await subjectRepository.GetModelsByEntityID(id);

                if (result.Count == 0)
                {
                    return NotFound();
                }

                return new JsonResult(result);
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }
```

ДОДАТОК Є

```
let clientAxios = axios.create({
  baseURL: "https://localhost:44330/",
  headers: {
    "Access-Control-Allow-Origin": "*",
    "Access-Control-Allow-Methods": "GET, POST, PATCH, PUT, DELETE, OPTIONS",
    "Access-Control-Allow-Headers": "Origin, Content-Type, X-Auth-Token",
    "Content-Type": "application/json , multipart/form-data",
    withCredentials: true,
```

```
  },
});
let path: string = "https://localhost:44318/api/";
  const GetTeacherFormById = async (teacherId: Number) => {
    clientAxios.get(path + `form/${teacherId}`).then((res: any) => {
      const formItems = res.data;
        setForms(formItems);
    });
  };
```