НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Навчально науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО:
В.о. завідувача кафедри
_____Олександр КОВАЛЬ
«___»_____2023 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій»
спеціальності 121 Інженерія програмного забезпечення
на тему: «Формування результатів щодо опитування»

Виконав:
студент IV курсу, групи ТІ-91
Гуріненко Андрій Русланович _____

Керівник:
Асистент кафедри ІПЗЕ
Швайко Валерій Григорович _____

Рецензент:

_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2023

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер - фізичних систем і веб-технологій.

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри
_____ Олександр КОВАЛЬ
(підпис)
”___”_____2023р.

ЗАВДАННЯ
на дипломну роботу студенту

Гуріненку Андрію Руслановичу

1. Тема роботи: «Формування результатів щодо опитування»

   керівник роботи: Асистент кафедри ІПЗЕ Швайко Валерій Григорович

   (прізвище, ім’я, по батькові науковий ступінь, вчене звання)

   затверджена наказом вищого навчального закладу від червня 2023р. № 2039-с від 29.05.2023 р.
2. Строк подання студентом роботи: 12.06.2023 р.
3. Вихідні дані до роботи: мова програмування JavaScript, середовище WebStorm 2023, база даних PostgreSQL
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Розробка засобів відтворення результатів опитування.
5. Перелік ілюстративного матеріалу: Створення моделей БД, створення моделей архітектури додатку, інструкції щодо користування ПП.

Дата видачі завдання «16» квітня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Отримання завдання	30.09.22	виконано
2.	Дослідження предметної області	17.10-20.12.2022	виконано
3.	Постановка вимог до програмної системи	20.01-22.02.2023	виконано
4.	Дослідження існуючих рішень	22.02-24.03.2023	виконано
5.	Розробка програмного продукту	24.04-10.05.2023	виконано
6.	Тестування	11.05-21.05.2023	виконано
7.	Захист програмного продукту	15.05 - 19.05.2023	виконано
8.	Оформлення дипломної роботи	22.05-04.06.2023	виконано
9.	Передзахист	05.06-11.06.2023	виконано
10.	Захист	19.06-23.06.2023	виконано

Студент: _____ Гуріненко А. Р.

(підпис) (прізвище та ініціали)

Керівник роботи: _____ Швайко В.Г.

(підпис) (прізвище та ініціали)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 77 сторінок, 29 рисунків, 1 додаток та 18 посилань.

Метою роботи є формування результатів щодо опитування у вигляді мікросервісу для загальної системи кафедри.

Для досягнення поставленої мети виконано такі завдання:

- проаналізовано підходи, моделі та методи формування результатів щодо опитування;

- проаналізовано підходи конкурентів їхню точність, сильні та слабкі сторони.

Розроблено моделі та методи, здатні:

- формувати дані у графіки та статистику;
- формувати дані в PDF-форматі;
- надавати користувачу зручний інтерфейс для представлення даних.

Була створена система, що використовує розроблені моделі та методи, і надає користувачу простий графічний інтерфейс для демонстрації сценаріїв використання.

Отримані результати мають практичне значення, оскільки вони надають модель і методи для класифікації намірів, які мають свої особливості та переваги, що описані у роботі. Було проведений огляд існуючих рішень, які вирішують певні задачі, і визначено напрямки подальшого розвитку та досліджень. Розроблена архітектура для асинхронної системи, яка може працювати на декількох пристроях. Був реалізований базовий інтерфейс для демонстрації роботи моделі.

Ключові слова: Програмний продукт, веб-додаток, REST API, результати опитування, PostgreSQL, мікросервіси, React, Fastify JS.

ABSTRACT

Structure and scope of the thesis. The work contains 77 pages, 29 figures, 1 appendix and 18 references.

The purpose of the work is to generate survey results in the form of a microservice for the department's general system.

To achieve the set goal, the following tasks were completed:

- the approaches, models and methods of forming survey results were analyzed;
- analyzed competitors' approaches, their accuracy, strengths and weaknesses.

Models and methods have been developed that are capable of:

form data into graphs and statistics;

generate data in PDF format;

provide the user with a convenient interface for presenting data.

A system was created that uses the developed models and methods and provides the user with a simple graphical interface to demonstrate usage scenarios.

The obtained results are of practical importance, as they provide a model and methods for the classification of intentions, which have their own characteristics and advantages, which are described in the work. An overview of existing solutions that solve certain problems was conducted, and directions for further development and research were determined. An architecture for an asynchronous system that can work on multiple devices is developed. A basic interface was implemented to demonstrate the operation of the model.

Keywords: Software Product, Web Application, REST API, Survey Results, PostgreSQL, Microservices, React, Fastify JS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

HTTP (англ. HyperText Transfer Protocol) - це протокол передачі гіпертекстових даних через мережу, який використовується для комунікації між клієнтом і сервером в Інтернеті. Він визначає правила, за якими клієнт і сервер обмінюються даними і здійснюють взаємодію.

БД - база даних.

REST (англ. Representational State Transfer) - архітектурний стиль для написання веб-сервісів.

API (англ. Application Programming Interface) - це набір правил та механізмів, що визначають, як програмні компоненти або системи можуть взаємодіяти між собою.

URI (англ. Uniform Resource Identifier) - це рядок символів, що ідентифікує унікальний ресурс, такий як веб-сторінка, зображення, відео або будь-який інший об'єкт, що доступний в Інтернеті.

UI (англ. User Interface) - користувацький інтерфейс.

PDF (англ. Portable Document Format) - формат файлу, що використовується для збереження та обміну документами.

HTML (англ. HyperText Markup Language) - це стандартна мова розмітки, яка використовується для створення та відображення веб-сторінок в Інтернеті.

CSS (англ. Cascading Style Sheets) - мова стилів, яка використовується для опису вигляду та форматування веб-сторінок, що написані з використанням HTML або інших мов розмітки.

ID - унікальний ідентифікатор.

Axios - JavaScript бібліотека для створення HTTP запитів.

Redux - JavaScript бібліотека для створення глобального сховища даних у клієнті.

ВСТУП

В сучасному світі, де інформація відіграє ключову роль у прийнятті рішень на всіх рівнях, збір і аналіз даних стає надзвичайно важливим. Один із способів збору інформації - опитування, які допомагають зрозуміти ставлення, думки або поведінку групи людей стосовно певного питання або проблеми.

Ця дипломна робота присвячена розробці клієнт-серверного додатка для формування матеріалів щодо опитування та представлення результатів опитування у веб-інтерфейсі. Додаток має на меті полегшити процес створення, проведення опитувань і візуалізації їх результатів. Це може бути корисним для організацій, дослідників, маркетологів та інших, хто зацікавлений у зборі і аналізі даних.

Робота складається із кількох розділів, включаючи вступ, огляд задачі, опис архітектури та дизайну додатка, деталі реалізації, і, нарешті, висновки. У вступі ми зосередимося на меті та завданнях дослідження, а також на актуальності і практичній значимості розробки.

Використання сучасних технологій у цій роботі допоможе вирішити проблему ефективного збору та аналізу даних опитувань, що є важливим у сучасному суспільстві, що швидко розвивається.

Додаток, що розробляється в рамках цієї роботи, має бути не тільки ефективним, але й зручним для користувача, що вимагає детального вивчення сучасних тенденцій в області веб-дизайну, а також знання принципів інтерфейсу користувача.

Використання сучасних технологій у цій роботі допоможе вирішити проблему ефективного збору та аналізу даних опитувань, що є важливим у сучасному суспільстві, що швидко розвивається. Крім того, наявність веб-інтерфейсу дозволить користувачам легко доступ до результатів опитувань з будь-якого місця і в будь-який час.

Ця робота спрямована на використання інформаційних технологій для вирішення практичних задач і викликів, пов'язаних з опитуваннями. Метою цієї

роботи є розробка та тестування клієнт-серверного додатка, що дозволить створювати та аналізувати опитування в одному централізованому місці, що полегшить процес збору та аналізу даних.

У рамках цієї роботи, ми також дослідимо потенційні виклики та обмеження, які можуть виникнути під час розробки та впровадження такого додатка. Ми також розглянемо можливості для подальшого розвитку та удосконалення додатка.

1 ЗАДАЧА ОБРОБКИ ТА ПЕРЕДАВАННЯ ДАНИХ ЩОДО ОПИТУВАННЯ

Сучасні технології відіграють важливу роль у нашому повсякденному житті, а Інтернет є необхідною складовою більшості компаній, які володіють власними веб-сайтами. КПІ не є винятком, і кафедрі необхідні зручні та функціональні інструменти для отримання даних про результати опитування студентів. Це відіграє важливу роль в акредитації.

Існують відкриті сервіси-аналоги для формування результатів опитування, таких як Google Form, TypeForm, SurveyMonkey та інші [1]. Проте, дані про результати опитування мають бути у доступі завжди, мати зручний інтерфейс для пошуку та відображення необхідної інформації, тому необхідно розробити мікросервіс, який матиме можливість інтегруватись в загальну міскросервісну архітектуру кафедри.

1.1 Мікросервісна архітектура

Мікросервісна архітектура є підхідом до розробки програмного забезпечення, в якому програма розбивається на набір невеликих та незалежних мікросервісів, що працюють разом для надання більшої функціональності. Кожен мікросервіс виконує конкретну задачу та має власну базу коду, незалежну розгортання та може комунікувати з іншими мікросервісами за допомогою легковагих протоколів.

Розробка мікросервісу включає в себе ряд задач, що потребують уваги. Спочатку потрібно визначити загальну функціональність додатку. Це можна зробити, виділивши логічні границі та окремі бізнес-функції, які можуть функціонувати незалежно від інших мікросервісів.

Розробка мікросервісу передбачає визначення протоколів та механізмів комунікації між мікросервісами. Це може включати використання HTTP-запитів,

мікросервісних шин або інших протоколів та технологій, що дозволяють мікросервісам обмінюватися даними та спілкуватися між собою.

Мікросервіс може мати свою власну базу даних або використовувати спільну базу даних. В нашому випадку, мікросервіс буде використовувати загальну базу даних, тому мають бути продумані таблиці необхідні для даного додатку в загальній БД.

Однією з переваг мікросервісної архітектури є можливість горизонтального масштабування окремих мікросервісів в залежності від навантаження. Потрібно, щоб додаток був масштабованим та ефективним в загальній архітектурі.

Також мікросервіс повинен мати можливість інтегрувати в загальну роботу системи та коректно працювати як її частина під керівництвом та управлінням загальної системи.

При розробці мікросервісу важливо враховувати ці задачі та забезпечити належну інтеграцію мікросервісу в загальну систему, згідно зі специфікаціями та потребами вашого проекту.

1.2 REST API

REST API (Representational State Transfer) є одним з найбільш популярних та ефективних підходів до створення мікросервісів, оскільки пропонує простоту використання, легкість сприйняття, масштабованість та стандартизацію. Використовуючи стандартні HTTP методи, такі як GET, POST, PUT і DELETE, REST API дозволяє розробникам швидко створювати та розширювати мікросервіси. Цей підхід також забезпечує простоту інтеграції з іншими системами [2].

Однією з переваг REST API є його масштабованість. Це досягається завдяки розділенню клієнтського та серверного стану (рис. 1.1). Кожен запит до REST API

містить усю необхідну інформацію для обробки, що дозволяє легко масштабувати мікросервіс, додавати нові екземпляри та балансувати навантаження між ними [3].

Ще однією перевагою REST API є його стандартизація. Використовуючи стандарти HTTP, такі як URI та методи запитів, REST API забезпечує зручну комунікацію зі сервісом. Це дозволяє розробникам використовувати загальні бібліотеки та фреймворки для спрощення роботи з API та легшої інтеграції з іншими системами.

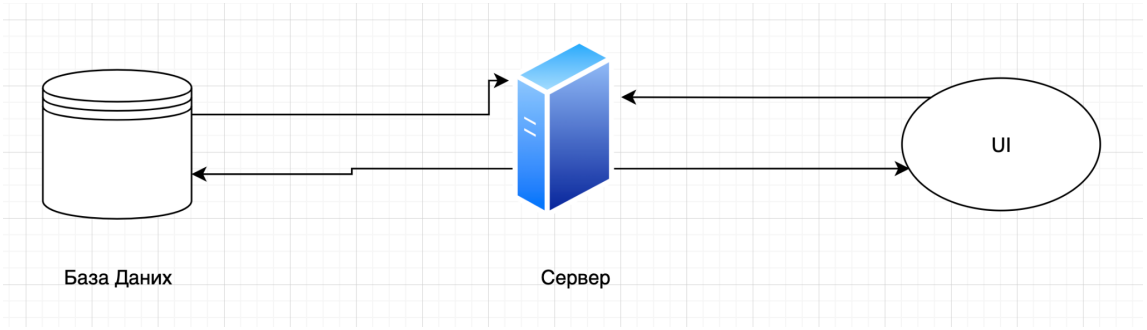

Рисунок 1.1 – Схема роботи REST API

Завдяки своїй розширюваності, REST API дозволяє розробникам створювати різні версії API та додавати нові ресурси без впливу на існуючі. Це дає більшу гнучкість та можливість розширювати функціонал мікросервісу з часом [4].

В цілому, REST API є ідеальним вибором для створення нашого мікросервісу, оскільки він надає простоту, масштабованість, стандартизацію та розширюваність.

1.3 Інтерфейс

Інтерфейс буде головною частиною роботи користувача з додатком, тому треба також вирішити всі необхідні завдання пов’язані з його коректною роботою.

Необхідно визначити енд-поінти для взаємодії з REST API нашого додатку, продумати шляхи отримання даних та їх зберігання на клієнті, перевірку та формування цих даних.

Також необхідно виконати перевірку та перехоплення можливих помилок пов’язаних з підключенням до API та отримання даних. При їх наявності необхідно повідомити про їх наявність користувачеві за допомогою інтерфейсу.

Інтерфейс має бути зручним та інтуїтивним у використанні, тому необхідно спираючись на юзер-кейси визначити загальну ієрархію сторінок та компонентів інтерфейсу та зробити зручний доступ до окремих її частин [5].

Клієнтська частина має мати можливість бути інтегрованою у фронт-енд загальної системи, інтуїтивною для інших розробників, простою в підтримці та розширенні.

1.4 Проектування бази даних

При проектуванні бази даних для мікросервісу в загальній мікросервісній архітектурі можуть виникати наступні задачі: визначення сутностей та атрибутів, встановлення зв'язків між сутностями, нормалізація даних, визначення типів даних та обмежень, розгортання бази даних.

Необхідно визначити основні сутності, які потрібно зберігати в базі даних мікросервісу, і визначте атрибути для кожної сутності. В даному мікросервісі про опитування, сутностями можуть бути "Опитування", "Питання" та "Відповідь", а їх атрибути можуть включати назву опитування, текст питання, варіанти відповідей.

Також необхідно проаналізувати, які зв'язки існують між сутностями в базі даних мікросервісу. Наприклад, одне опитування може містити багато питань, а питання можуть мати багато відповідей. Встановлення правильних зв'язків допоможе забезпечити цілісність даних та зручний доступ до них.

Необхідно проаналізувати принципи нормалізації даних для забезпечення ефективності та цілісності бази даних. Однією зі складових задачі є, як розподілити дані на таблиці та встановити відповідні залежності та ключі.

Також треба встановити правильні типи даних для атрибутів та визначити обмеження, що відповідають логіці мікросервісу. Наприклад, встановити обмеження на довжину текстових полів або обмежень на діапазон числових значень [6].

Останньою складовою є визначення системи управління базою даних та способом її розгортання та міграції.

1.5 Опис задачі

Для проектування даного мікросервісу необхідно:

1. Доступ до сховища даних, щоб мати можливість отримати інформацію.
2. Створення REST API серверу, який буде обробляти запити клієнта, відправляти запит у базу даних для повернення даних, що необхідні клієнту.
3. Створення клієнта з продуманим UI, що дасть можливість відобразити необхідні дані.
4. Правильне налаштування шляхів для відправки та отримання запитів між клієнтом на сервером.
5. Організація відловки помилок на випадок, якщо сервер чи база даних вийдуть з ладу.
6. Вибір правильного стеку технологій.
7. Надання користувачеві інтерфейсу, який має можливість покрити більшість необхідних йому задач.

В даному випадку API буде найголовнішою складовою нашого проекту, оскільки і буде відповідати за функції мікросервісу та матиме можливість

подальшої інтеграції в загальну систему, коли клієнта частина може змінитись або ж включити в себе наявне інтерфейсне рішення.

Основою ж задачею користувацького інтерфейсу буде надати користувачеві, всі необхідні інструменти для того, щоб він між можливість коректно отримати дані та наглядно побачити їх результати.

Висновки до розділу

В цьому розділі була описана задача, поставлена для створення додатку для формування результатів опитування. Також були описані головні етапи роботи та необхідні для реалізації проекту архітектурні та функціональні рішення.

2 АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ ПРОДУКТІВ ДЛЯ ФОРМУВАННЯ РЕЗУЛЬТАТІВ ОПИТУВАННЯ

В цьому розділі проведемо аналіз програм аналогів та визначимо основні особливості та рішення, які зустрічаються в програмах такого типу.

2.1 Пошук програм аналогів

При задачі по аналізу існуючих веб-орієнтованих додатків для перегляду результатів опитувань першим, що спадає на думку, авжеж є Google Forms від Google. Цей сервіс набув особливої популярності завдяки своїй простоті та зручному та інтуїтивному користувацькому інтерфейсу. Він надає всі можливості для створення опитувань та, що для нас найголовніше перегляду результатів цих опитувань.

Іншим популярним сервісом для створення опитувань є TypeForm [7]. Цей інструмент надає зручний та інтуїтивно зрозумілий інтерфейс для створення опитувань з різноманітними типами питань, такими як однорядкові поля, багаторядкові поля, питання з варіантами відповідей, оцінювання, вбудовані зображення та відео.

Для третього сервісу-аналогу був обраний сервіс проведення опитувань про викладачів КПІ СОВА. Цей сервіс покриває одну з головних задач, що має виконувати наш додаток, а саме проведення опитувань про викладачів. Він проводить опитування за допомогою Telegram бота, і потім публікує результати у вільному доступі. Хоч формування результатів у КПІ СОВІ не є автоматизованим процесом, а цим займається команда і потім викладає на свій канал, проте нас в цьому контексті цікавить саме спосіб представлення та відображення отриманих даних.

2.2 Google Forms

Google Forms є популярним і безкоштовним інструментом для створення опитувань та збору даних. Він надає зручний та інтуїтивно зрозумілий інтерфейс для створення опитувань з різноманітними типами питань, такими як питання з одним варіантом відповіді, багаторядкові поля, вибір зі списку, шкали оцінювання та багато інших. Ви можете налаштовувати обов'язкові питання та додавати варіанти відповідей за вашим вибором.

Одна з ключових переваг Google Forms полягає у можливості налаштування дизайну опитування. Ви можете вибрати кольори, шрифти та макет, що дозволяє створювати унікальні та привабливі опитування, що привертають увагу учасників.

Google Forms також надає зручні інструменти для збору та аналізу результатів опитування. Ви можете переглядати відповіді у реальному часі, отримувати зведену статистику та аналізувати дані за допомогою візуалізацій. Крім того, ви можете експортувати дані в таблицю Google Sheets для подальшого аналізу та обробки.

Як ми можемо побачити, Google Forms надає загальний огляд по опитуванню, формує графіки для більш наглядної демонстрації результатів, а також має окремі вкладки, де також можна переглянути результати по питанням та користувачам (рис. 2.1).

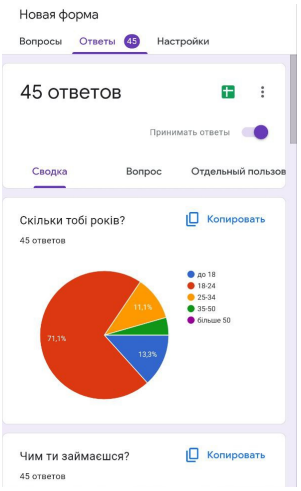

Рисунок 2.1 – Результати щодо опитування у Google Form

Огляд результатів від Google Forms включає в себе загальну кількість відповідей користувачів та назву форми. Також бачимо, що для загальної навігації по даним форми використовуються вкладки вгорі. Результати відображаються по принципу карток, кожна з яких відображає окреме питання, що було у опитуванні.

Результати кожного питання, які мали варіанти відповіді, відображена у вигляді графіку - стовпчикового або радіального (діаграма “пиріг”). Також вказується саме питання та кількість відповідей по цьому питанню (рис. 2.2).

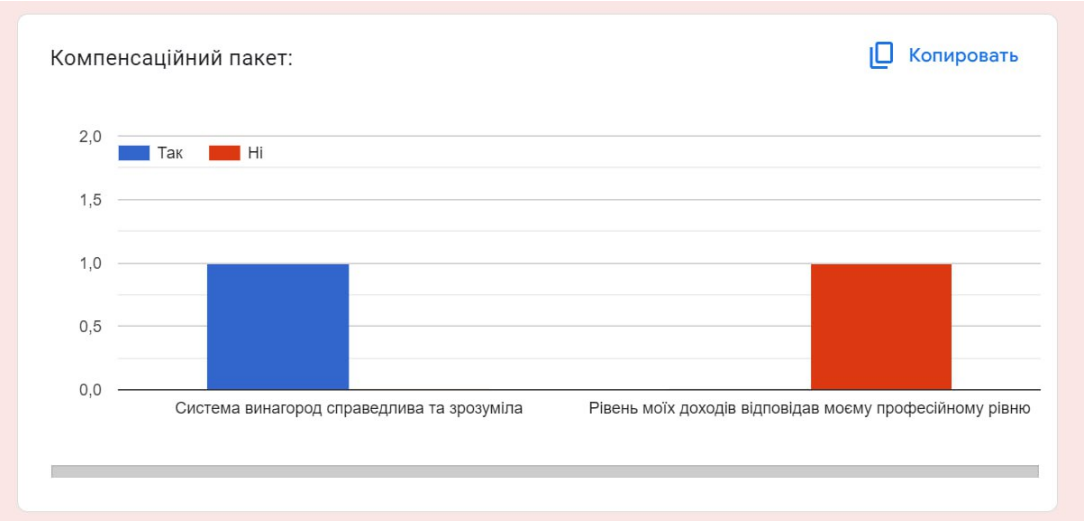

Рисунок 2.2 – Результати по окремому питанню Google Forms у вигляді стовпчикової діаграми

Також дані, можна переглянути у вигляді таблиці, яка буде відображати відповіді, що є альтернативним способом перегляду даних.

2.3 TypeForm

Typeform - це інтерактивна платформа для створення привабливих та залучаючих опитувань. Цей інструмент надає зручний та інтуїтивно зрозумілий інтерфейс для створення опитувань з різноманітними типами питань, такими як однорядкові поля, багаторядкові поля, питання з варіантами відповідей, оцінювання, вбудовані зображення та відео.

Одна з ключових переваг Typeform полягає у можливості налаштування дизайну опитування. Ви можете вибрати кольори, шрифти та макет, що дозволяє створювати унікальні та привабливі опитування, що привертають увагу учасників [8].

Typeform також пропонує інтерактивні елементи, що роблять опитування більш цікавими та залучаючими. Ви можете використовувати анімації, плавні переходи між питаннями, умовне відображення питань в залежності від вибору учасника та створювати персоналізовані відповіді.

Одним з важливих аспектів є можливість аналізу результатів опитування. Typeform надає зручні інструменти для перегляду відповідей у режимі реального часу, отримання зведених статистичних даних та застосування фільтрів та сегментації для детальнішого вивчення результатів.

Тому, Typeform є зручним інструментом для створення привабливих та інтерактивних опитувань, який дозволяє легко збирати дані від учасників та отримувати корисні інсайти зі зібраних результатів.

Переглядаючи загальні результати опитування TypeForm можна побачити загальну навігацію по даним форми у вигляді вкладок (рис. 2.3)[8].

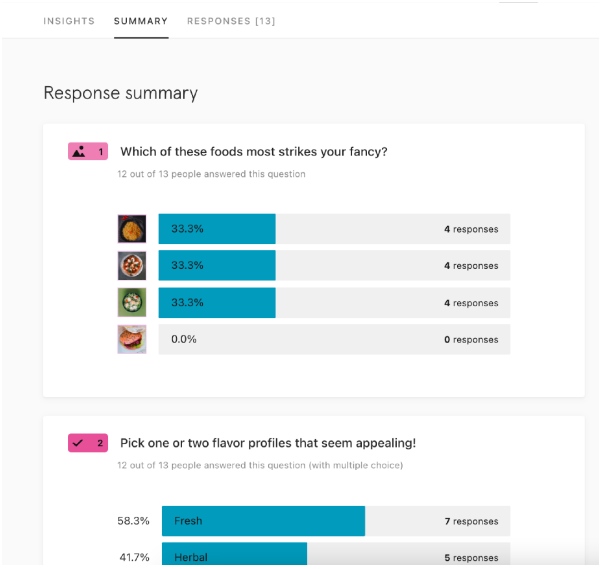

Рисунок 2.3 – Результати щодо опитування TypeForm (TypeForm Documentation)

Перша вкладка, відображає загальні дані форми, що включає в себе дані про кількість переглядів форми, дату початку, кількість відповідей, відсоток завершеності відповідей та час для завершення форми. Також є лінійний графік, що відображає залежність кількості відповідей від часу (рис. 2.4)[8].

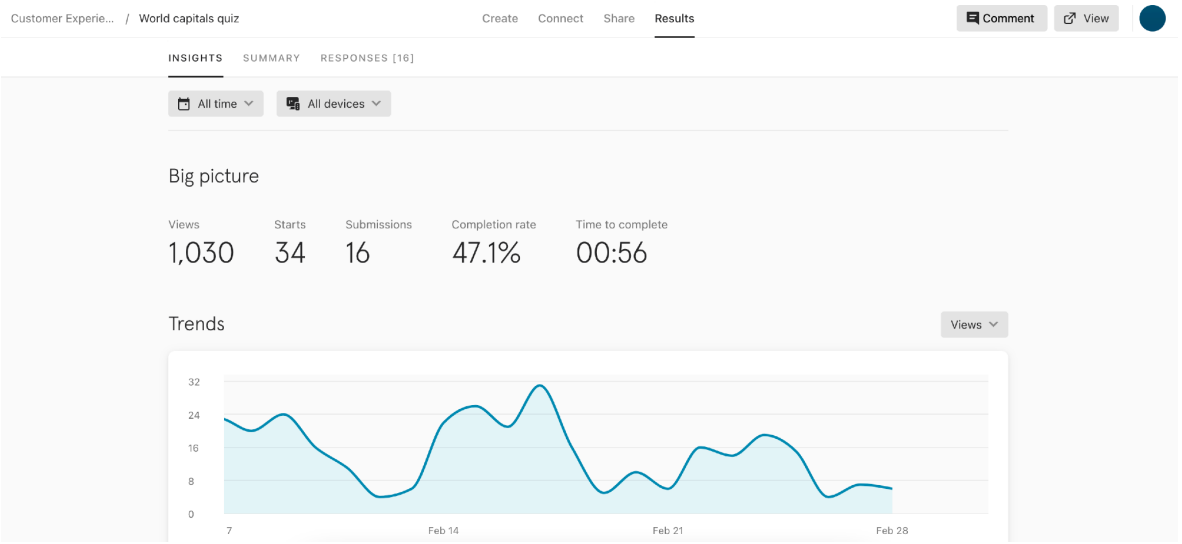

Рисунок 2.4 – Загальні результати TypeForm (TypeForm Documentation)

Друга вкладка відображає результати по кожному з питань. Кожне питання представлене у вигляді картки. Кожна картка також має в собі саме питання, кількість відповідей та стовпчикову діаграму для представлення частки кожної з відповідей по відношенню до загальної кількості відповідей.

Остання вкладка відображає кожну з відповідей у вигляді таблиці. Кожен рядок має в собі загальні дані по відповіді, включаючи час відповіді та відповіді на конкретні питання (рис. 2.5) [8].

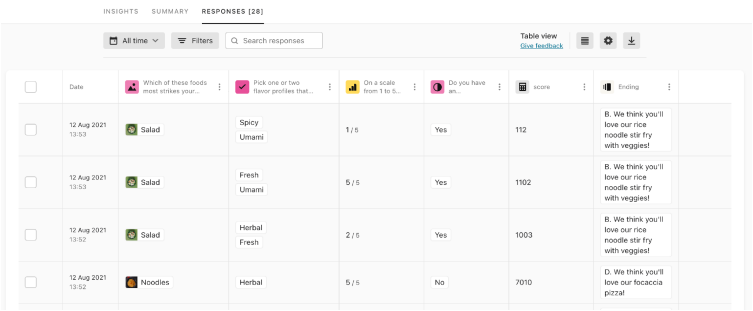

Рисунок 2.5 – Вкладка з результатами TypeForm (TypeForm Documentation)

Також TypeForm при формуванні результатів опитування має кнопку для формування звіту по опитуванню. При натисканні на цю кнопку, окремі дані по результатам цього опитування формуються у PDF файл, що і буде звітом, та може використовуватись як документ для пересилання чи прикладання до певних даних.

2.4 Опитування студентів СОВА

КПІ СОВА це студентська ініціатива для проведення оцінки викладачів КПІ серед студентів. Вона має на меті покращення якості викладання предметів в КПІ. Сервіс підтримується командою студентів та збирає дані про викладачів кожного факультету.

Головними перевагами цього сервісу є простота використання та анонімність. Для його використання достатньо мати месенджер Telegram. Взаємодія з користувачами відбувається через Telegram бота, а результати ж публікуються після завершення опитування на окремому каналі у вигляді зображень.

Формування результатів опитування не є повністю автоматизованим, і проводиться командою КПІ СОВА і подається у вигляді однієї картинки з усіма даними по конкретному викладачу (рис. 2.6)[9].

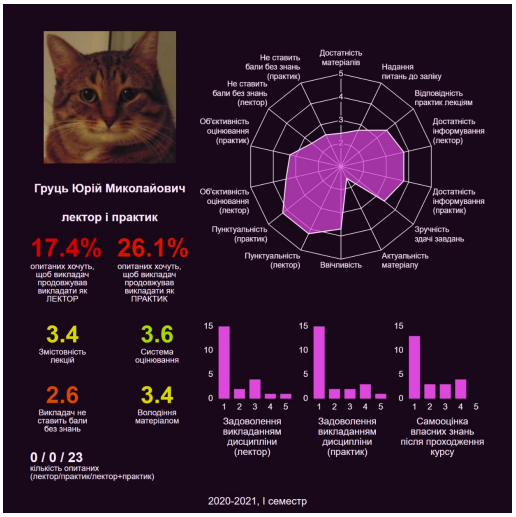

Рисунок 2.6 – Результати опитування про викладача КПІ СОВА (телеграм-канал КПІ СОВА)

Як ми можемо побачити результати включають в себе лише загальний огляд результатів, який представлений у вигляді діаграм, або відсоткових чи числових значень.

Головним оглядом результатів є графік-радар, кожна з осей якого представляє окреме питання, що оцінювалось, і дає змогу оцінити загальну картину опитування.

Є також додаткові дані, які відображені у вигляді стовпчикових діаграм, відсоткових співвідношень чи середньої числової оцінки.

2.5 Результати аналізу

За результатами аналізу цих трьох сервісів ми можемо зробити певні висновки про те, які рішення найчастіше використовуються у подібних додатках та визначити, які рішення є найбільш оптимальними для вирішення нашої задачі.

Всі сервіси мали в собі загальний огляд по опитуванню - TypeForm та Google Forms мали для цього окремі вкладки, коли як КПІ СОВА представляла дані опитування тільки таким чином.

Всі сервіси використовували при представленні інформації про результати графіки. Найрозповсюдженішим графіком була стовпчикова діаграма. Також зустрічались лінійні графіки та радіальні графіки. Окремо варто виділити діаграму-радар, що використовувалась у КПІ СОВА. Такий вид діаграми дозволяє зробити візуальну репрезентацію даних по всім питання щодо оцінювання та дає змогу отримати загальну картину.

Також для інтерфейсу ці сервіси для навігації по різним даним опитування використовують вкладки, а дані про окреме питання зображують у вигляді картки з даними.

Кожне питання містить в собі графічне представлення результатів, саме питання та кількість відповідей на це питання.

Також, для відображення результатів опитування також використовуються відсоткові співвідношення, представлення у вигляді таблиці, або середні значення.

Також окремо варто виділити функцію експорту, яка була помічена у TypeForm. Оскільки результати опитувань по оцінюванню сформовані розробляємим сервісом потім мають використовуватись у документах, то можливість експорту результатів опитування у окремий PDF документ ж досить корисною.

Висновки до розділу

В даному розділі був описаний пошук та вибір програм-аналогів для проведення аналізу. В результаті пошуку були обрані такі сервіси - Google Forms, TypeForm, КПІ СОВА.

Був проведений аналіз кожного з сервісів для визначення того, які рішення найчастіше використовуються в подібних сервісах, які можливості вони надають та які унікальні функції кожен з них має.

На основі проведеного аналізу по кожному з сервісів був проведений загальний аналіз, де були оцінені найкращі рішення та виділені окремі функції, які заслуговують уваги.

3 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка сучасного додатку потребує використання сучасних технологій. В нашому випадку, нам необхідні засоби для самої розробки, засоби розробки клієнтської, серверної частини та бази даних.

3.1 Середовище розробки WebStorm

WebStorm - це інтегроване середовище розробки (IDE), спеціально призначене для веб-розробки на основі мови JavaScript. Розроблене компанією JetBrains, WebStorm надає розширені можливості для побудови високоякісних та ефективних веб-додатків.

Однією з ключових особливостей WebStorm є його потужний редактор коду, який надає широкі можливості для редагування та форматування коду JavaScript, HTML та CSS. Він підтримує автоматичне завершення коду, перевірку синтаксису, рефакторинг, навігацію по проекту та багато іншого, що сприяє підвищенню продуктивності розробника.

WebStorm також має розширені можливості для роботи з фреймворками та бібліотеками JavaScript, такими як React, Angular, Vue.js та інші. Воно підтримує автоматичне виявлення налаштувань та інструментів для цих фреймворків, надає контекстну допомогу, швидкі фіксації помилок та інші корисні функції, що спрощують розробку веб-додатків на їх основі.

Інтеграція з системами контролю версій, такими як Git, є ще однією важливою можливістю WebStorm. Воно надає зручний інтерфейс для роботи з гілками, комітами, злиттями та іншими операціями Git, що полегшує спільну роботу розробників та управління версіями проекту.

Крім того, WebStorm підтримує інтеграцію з іншими інструментами розробки, такими як лінтери, системи збирання проекту, тестування та інші. Воно дозволяє налаштувати різні робочі середовища та використовувати їх залежно від

потреб вашого проекту. Також варто зазначити, що WebStorm має широкі можливості для налаштування, що дозволяє адаптувати його до ваших особистих вимог та робочого процесу.

Окрім цього, WebStorm забезпечує потужні інструменти для відлагодження коду, включаючи можливість ставити точки зупинки, спостерігати за значеннями змінних, використовувати консоль для виводу інформації та аналізу стеку викликів. Це дозволяє розробникам швидко виявляти та виправляти помилки у своєму коді.

Зручний інтерфейс користувача та дружнє середовище WebStorm сприяють полегшенню навігації по проекту, відкриттю та редагуванню файлів, а також взаємодії з іншими членами команди. Він надає різноманітні інструменти для організації проекту, включаючи пошук файлів, навігацію по класам та функціям, рефакторинг та інші корисні функції.

Узагальнюючи, WebStorm є потужним інструментом розробки, який надає розширені можливості для розробки веб-додатків на основі JavaScript. Його функціональність, включаючи редактор коду, підтримку фреймворків, інтеграцію з системами контролю версій та інші інструменти, дозволяють розробникам працювати більш ефективно та продуктивно. Використання WebStorm може сприяти полегшенню розробки, покращенню якості коду та прискоренню розробки веб-додатків.

3.2 Мова програмування JavaScript

JavaScript є високорівневою, інтерпретованою мовою програмування, призначеною для розробки веб-додатків. Вона широко використовується для надання динамічного та інтерактивного змісту на веб-сторінках, а також для розробки серверних додатків, мобільних додатків та додатків для Інтернету речей (IoT).

JavaScript є інтерпретованою мовою, що означає, що вона виконується безпосередньо в середовищі браузера або іншому розподіленому середовищі, без необхідності компіляції. Вона підтримує динамічність, що дає можливість змінювати структуру, типи даних та поведінку об'єктів під час виконання програми.

JavaScript підтримує об'єктно-орієнтовану парадигму програмування, що дозволяє створювати об'єкти, використовувати наслідування, інкапсуляцію та поліморфізм для організації коду. Вона також підтримує функціональну парадигму, де функції можуть бути передані як аргументи, повернуті з інших функцій та замкнуті в змінні, що називається замиканнями.

JavaScript є основною мовою для взаємодії з браузером та зміни вмісту сторінок через використання DOM (Document Object Model). Вона надає можливість маніпулювати DOM-елементами, змінювати їх властивості, структуру та реагувати на події користувача.

JavaScript широко використовується веб-розробниками для створення динамічного веб-змісту. Вона також знаходить застосування у розробці мобільних додатків за допомогою

фреймворків, таких як React Native і NativeScript, що дозволяють розробляти кросплатформні мобільні додатки. Крім того, JavaScript також використовується для розробки серверних додатків за допомогою платформи Node.js, що дозволяє розробникам використовувати JavaScript як мову програмування для створення повноцінних серверних застосунків.

JavaScript має широке співтовариство розробників, що забезпечує доступ до багатьох ресурсів, бібліотек, фреймворків та інструментів, що полегшують розробку. Відкритий характер мови також сприяє створенню активної спільноти, яка активно співпрацює, вносить внески та покращує мову та її екосистему.

JavaScript постійно розвивається, і нові версії стандарту ECMAScript (ES) вводять нові можливості та поліпшення в мову. Останні версії, такі як ECMAScript

6 (ES6) і його наступні ітерації, пропонують сучасні функції, синтаксичні поліпшення та розширені можливості для розробників.

3.3 Мова програмування TypeScript

TypeScript - це мова програмування, яка розширює JavaScript, надаючи розробникам додаткові можливості та інструменти для побудови великих, складних проектів. Однією з найважливіших особливостей TypeScript є статична типізація, яка дозволяє визначати типи даних для змінних, параметрів та повернутих значень функцій. Це допомагає виявляти помилки на етапі компіляції, забезпечуючи більшу надійність та розуміння коду.

Крім того, TypeScript підтримує об'єктно-орієнтовану парадигму програмування. Розробники можуть використовувати класи, успадкування, інтерфейси та інші об'єктно-орієнтовані концепції для створення чистого, модульного коду. Це полегшує розробку, підтримку та розширення проектів, сприяючи організованому та ефективному розробчому процесу.

За допомогою TypeScript також можна використовувати сучасні функціональні можливості мови JavaScript, такі як лямбда-вирази, замикання та виразові функції. Це дозволяє використовувати функціональну парадигму програмування для створення більш елегантного та декларативного коду, що полегшує розуміння та підтримку.

3.4 Фреймворк React

React - це потужна бібліотека JavaScript для розробки інтерфейсу користувача. Вона дозволяє розробникам будувати компонентну, декларативну та масштабовану UI (інтерфейс користувача) для веб-додатків. React створений компанією Facebook і здобув велику популярність завдяки своїм унікальним підходам та перевагам.

Одна з ключових особливостей React - це використання віртуального DOM (Document Object Model). React створює внутрішнє представлення DOM у пам'яті, що дозволяє оптимізувати процеси оновлення та рендерингу елементів інтерфейсу. Це робить React дуже ефективним та швидким для змін у візуалізації даних [10].

Інша важлива особливість React - це компонентна модель. Розробники можуть створювати повторно використовувані компоненти, які представляють окремі частини інтерфейсу. Це дозволяє структурувати код, полегшує підтримку та розширення проекту, а також сприяє зручній співпраці в команді розробників.

React також пропонує одностороннє потокове управління даними. Застосовуючи концепцію "props" (властивостей) та "state" (стану), розробники можуть ефективно керувати даними та оновленнями компонентів. Це сприяє покращенню стану додатків, дозволяє відслідковувати зміни та реагувати на них, забезпечуючи більш динамічний та відзивчивий користувацький інтерфейс.

Нарешті, React має широку спільноту розробників, що допомагає сприяти поширенню знань, надає доступ до багатьох сторонніх бібліотек та інструментів, а також сприяє активному обміну досвідом та підтримці. Це робить React однією з найпопулярніших та найбільш підтримуваних технологій у сфері розробки інтерфейсу користувача.

React також має екосистему розширень, таких як React Router для маршрутизації, Redux для керування станом додатків, а також багато інших сторонніх компонентів та бібліотек. Це дозволяє розробникам використовувати готові рішення та прискорювати процес розробки, забезпечуючи більшу функціональність та гнучкість.

3.5 Фреймворк Fastify

Fastify - це швидкий і ефективний фреймворк для розробки веб-додатків на мові програмування JavaScript. Він побудований на основі Node.js та відомий

своєю високою продуктивністю та низькими накладними витратами. Fastify пропонує простий та елегантний API, який дозволяє розробникам швидко створювати потужні серверні додатки.

Одна з ключових особливостей Fastify - це його швидкодія. Фреймворк використовує оптимізований обробник маршрутів, який дозволяє досягати вражаючої продуктивності та низького рівня навантаження на сервер. Fastify також пропонує асинхронну обробку запитів та підтримку стрімів, що сприяє забезпеченню ефективного використання ресурсів сервера.

Fastify також відзначається своєю гнучкістю та розширюваністю. Він надає широкий спектр плагінів та допоміжних утиліт, що дозволяють розробникам розширювати функціональність фреймворка за потреби. Це дозволяє створювати додатки з різноманітними функціями, такими як автентифікація, журналювання, кешування та багато інших [11].

Fastify має чистий та простий синтаксис, що спрощує розробку та розуміння коду. Він надає розробникам зручні інструменти для роботи з HTTP-запитами, маршрутами, обробниками помилок та іншими аспектами серверної розробки. Крім того, Fastify надає широку документацію та активну спільноту розробників, що забезпечує підтримку та надає доступ до великої кількості ресурсів та прикладів

3.6 Бібліотека Objection JS

Objection.js - це бібліотека JavaScript, яка надає ORM (Object-Relational Mapping) для зручної та ефективної роботи з базами даних в Node.js. Вона дозволяє розробникам взаємодіяти з базами даних за допомогою об'єктно-орієнтованого підходу, що полегшує роботу з даними та спрощує створення складних запитів.

Одна з ключових особливостей Objection.js - це підтримка реляційних баз даних, таких як PostgreSQL, MySQL, SQLite та інші. Бібліотека надає можливість

створювати моделі даних, відображати таблиці баз даних на класи JavaScript і виконувати різноманітні запити, включаючи створення, читання, оновлення та видалення даних.

Бібліотека також пропонує розширені можливості управління відносинами між таблицями баз даних. За допомогою Objection.js розробники можуть використовувати асоціації, включаючи один до одного, один до багатьох та багато до багатьох, для зручної та логічної організації даних та виконання складних операцій з ними.

3.7 PostgreSQL

PostgreSQL - це потужна та розширювана система управління базами даних (СУБД) з відкритим кодом. Вона надає надійне та гнучке середовище для зберігання, організації та маніпулювання структурованою і навіть неструктурованою інформацією. PostgreSQL підтримує багато різних типів даних, включаючи числа, рядки, дати, масиви та інші, а також надає розширені можливості, такі як географічні типи та JSON-операції.

Одна з ключових переваг PostgreSQL - це його висока надійність та стійкість до відмов. Він забезпечує механізми транзакцій, які дозволяють забезпечити цілісність даних та захистити їх від втрати чи пошкодження при непередбачених ситуаціях. PostgreSQL також підтримує механізми резервного копіювання та відновлення, що дозволяють забезпечити надійну та безпечну роботу з даними.

Ще одна вагома перевага PostgreSQL - це його розширюваність та можливості налаштування. Він надає розширену підтримку мови SQL, а також можливості створення функцій та процедур, що дозволяють розробникам виконувати складні операції з даними прямо на рівні бази даних. Більше того, PostgreSQL має велику кількість розширень та модулів, що дозволяють розширити його функціональність і використати специфічні можливості залежно від потреб проекту.

PostgreSQL також має активне співтовариство розробників, яке сприяє підтримці, поширенню знань та вирішенню проблем. Воно надає широкий спектр документації, прикладів використання, форумів та онлайн-ресурсів, що допомагають розробникам отримати необхідну інформацію та розв'язати поточні завдання.

Крім того, PostgreSQL підтримує розподілені бази даних, що дозволяє розміщувати дані на кількох серверах та забезпечує високу доступність та масштабованість. Це важливо для веб-додатків та систем з великим обсягом даних, де вимоги до продуктивності та масштабованості є важливими факторами.

3.8 Material UI

Material-UI - це популярна бібліотека компонентів для розробки інтерфейсу користувача веб-додатків. Вона базується на дизайні Material Design від Google, який пропонує модерні та стильні елементи дизайну, а також принципи розміщення та інтеракції. Material-UI дозволяє розробникам швидко і зручно створювати привабливі та функціональні інтерфейси без необхідності розробляти компоненти з нуля.

Одна з головних переваг Material-UI полягає в його гнучкості та налаштованості. Бібліотека надає велику кількість готових компонентів, таких як кнопки, поля вводу, таблиці, картки тощо, які можна легко інтегрувати у веб-додаток. Крім того, Material-UI дозволяє змінювати стилі компонентів шляхом налаштування теми, дозволяючи адаптувати їх до власних потреб та візуального стилю проекту.

Ще однією перевагою Material-UI є його активна спільнота розробників та документація. Велика кількість ресурсів, прикладів використання та документації дозволяють розробникам швидко зрозуміти та освоїти бібліотеку. Спільнота розробників Material-UI також надає підтримку, відповідає на питання та допомагає вирішувати проблеми, що виникають під час розробки.

Material-UI пропонує високу якість дизайну та виконання компонентів. Вони розроблені з урахуванням кращих практик і забезпечують зручну та приємну взаємодію з користувачем. Компоненти Material-UI мають вбудовану підтримку для адаптивного дизайну, що дозволяє їм працювати ефективно на різних пристроях та розмірах екранів. Вони автоматично адаптуються до мобільних пристроїв, планшетів та настільних комп'ютерів, забезпечуючи однаково зручний інтерфейс для користувачів незалежно від пристрою, на якому вони працюють.

Material-UI також підтримує інтернаціоналізацію та локалізацію, що робить його ідеальним вибором для багатомовних проектів. Бібліотека надає зручні інструменти для перекладу та відображення тексту на різних мовах, що дозволяє створювати міжнародні додатки з легкістю.

3.9 Обгрунтування вибору стеку технологій

В першу чергу було обрано середовище розробки. Оскільки була необхідність працювати з багатьма інструментами для паралельної розробки одночасно і Back-End, і Front-End частини додатку, тому була обрана система WebStorm від JetBrains. На відміну від інших продуктів JetBrains, WebStorm ідеально підходить для написання веб-проектів, оскільки і був для цього створений. Також, всі продукти JetBrains мають інтеграцію з великим переліком технологій, які роблять більш зручним та продуктивним розробку проектів. В цьому випадку були корисними інтеграція з системою контролю версій Git, наявність вбудованого терміналу та ефективна система скорочень та підказок.

JavaScript є мовою вебу і невід'ємною частиною сучасної веб-розробки. Тому без мови програмування JavaScript неможливо розробити динамічний веб-додаток. Також JavaScript є головним рішенням для створення прогресивних веб-інтерфейсів та додатків, а також є популярним рішенням для створення Back-end частини сайту, оскільки це дає можливість писати серверну та клієнтську частину проекту на одній мові програмування. До того ж, згідно в

бенчмарками, JavaScript-фреймворк входить в десятку найшвидших та ефективніших фреймворків для написання Back-end.

Використання TypeScript є обгрунтованим завдяки тому, що TypeScript є підмножиною JavaScript, тобто має його в основі та розширює його функціонал. В нашому випадку використання TypeScript дозволяє замінити динамічну типізацію JavaScript на більш прогнозовану статичну типізацію, що є доволі хорошим рішенням у сучасній розробці, оскільки робить код додатку більш прогнозованим, зрозумілим та дозволяє уникнути великої кількості можливих помилок під час розробки.

Використання React було однією з домовленостей при розробці мікросервісів для спільної системи кафедри. Проте, його використання в сучасній веб-розробці є доволі розповсюдженим рішенням, оскільки React є одним найпопулярніших веб-фреймворків для написання веб-інтерфейсів. Він надає можливість робити односторінкові сайти з динамічними даними, реактивністю, управлінням станом та прогресивним використанням REST API для створення динамічних інтерфейсів.

Оскільки до цього було визначено, що для написання серверної частини проекту буде використовуватись мова програмування JavaScript, тому наступним важливим кроком є обрання JS-фреймворку для написання серверної частини сайту. Обраним фреймворком став Fastify JS, оскільки при його створення головною метою була швидкість роботи, ефективність та продуктивність. Дійсно, згідно з бенчмарками, написані на Fastify JS серверні додатки є швидшими за своїх найпопулярніших конкурентів, і його можна сміливо вважати одним з найшвидших фреймворків, що є важливо для нашого проекту, оскільки ми не оперуємо великою кількістю даних чи складним функціоналом, тому головною ціллю можна зробити швидкість роботи.

Для роботи з БД було обрано саме Objection JS, оскільки він має підтримку багатьох мов управління базами даних і спрощує роботу з ними, оскільки

перетворює все на моделе-орієнтовану взаємодію з БД. Це дає змогу фокусуватись саме на моделях даних, а не на формування запитів.

Використання PostgreSQL було також однією з домовленостей при розробці мікросервісів для загальної системи. PostgreSQL відома своєю надійністю, масштабованістю та стійкістю до навантажень, що робить її відмінним варіантом для роботи з великими обсягами даних та складними запитами.

Останньою складовою необхідною для нашого проекту було обрання бібліотеки для створення інтерфейсу. Це було одним з завдань перед даним проектом, оскільки цей проект буде частиною великої системи, тому було необхідне якесь уніфіковане та універсальне рішення у створенні веб-інтерфейсу. Бібліотека, на відміну від унікального написання стилей робить код інтерфейсу інтуїтивним для кожного розробника, а також спрощує його перевикористання. Саме Material UI була обрана завдяки своїй популярності, оскільки це розробка від Google, а інтерфейси від Google вже стали інтуїтивними для більшості користувачів по всьому світу.

Висновки до розділу

В даному розділі описано стек технологій обраний для розробки даного проекту. В якості мови програмування був обраний JavaScript, як для написанні серверної, так і клієнтської частини додатку. Також була використана надбудова над JavaScript, а саме TypeScript для створення статичної типізації під час розробки.

В якості Front-end фреймворки був обраний React.

Для написання серверної частини, тобто REST API були обрані Fastify JS та Objection JS. В якості мови управління базою даних був обраний PostgreSQL.

Для створення уніфікованого веб-інтерфейсу була обрана UI-бібліотека Material UI від Google.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

За допомогою наявних інструментів та технологій вже можна приступити до виконання самої реалізації додатку.

4.1 Діаграма прецедентів

Однією з форм представлення необхідного функціоналу програмного продукту є діаграма прецедентів, яка визначає сценарії використання програмного продукту їх користувачами (рис. 4.1).

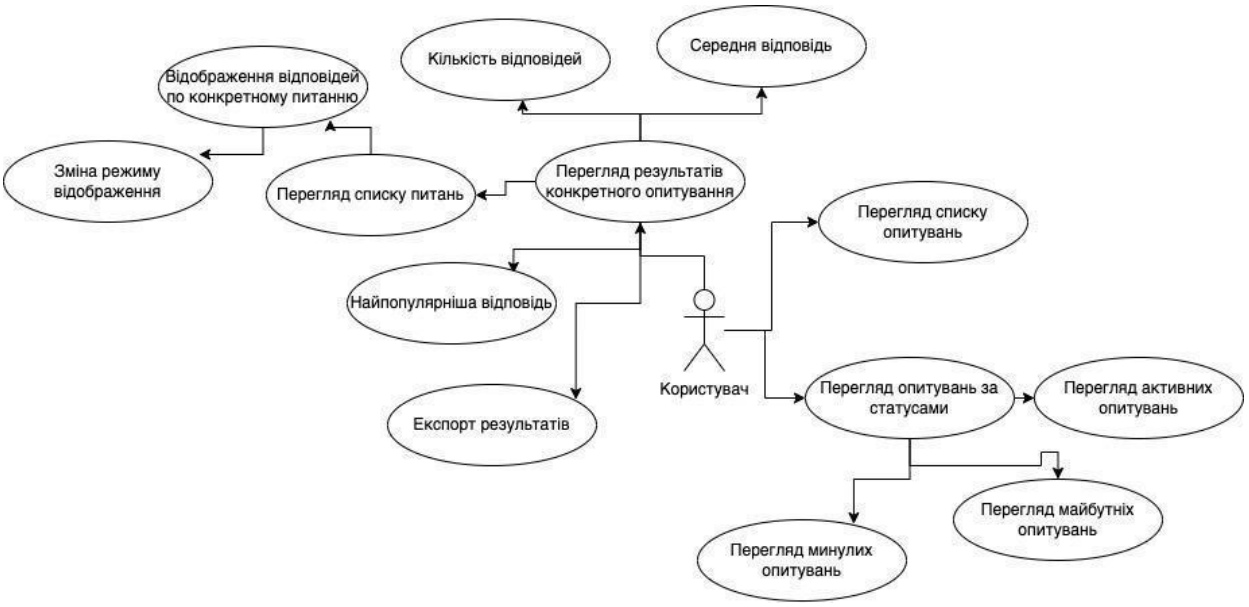

Рисунок 4.1 – Діаграма прецедентів

На даній діаграмі можна побачити основні шляхи використання користувачем цього програмного продукту. В нього є опції перегляду списку опитувань в декількох виглядах, включно з розбиттям по статусам. Також є можливість перегляду результатів по конкретному опитуванню, перегляду питань та відповідей, а також зміни режимів відображення даних.

4.2 Опис вхідних даних

В нашому випадку вхідними даними є дані про користувачів, дані про створені опитування та дані про результати проходження опитувань (рис. 4.2).

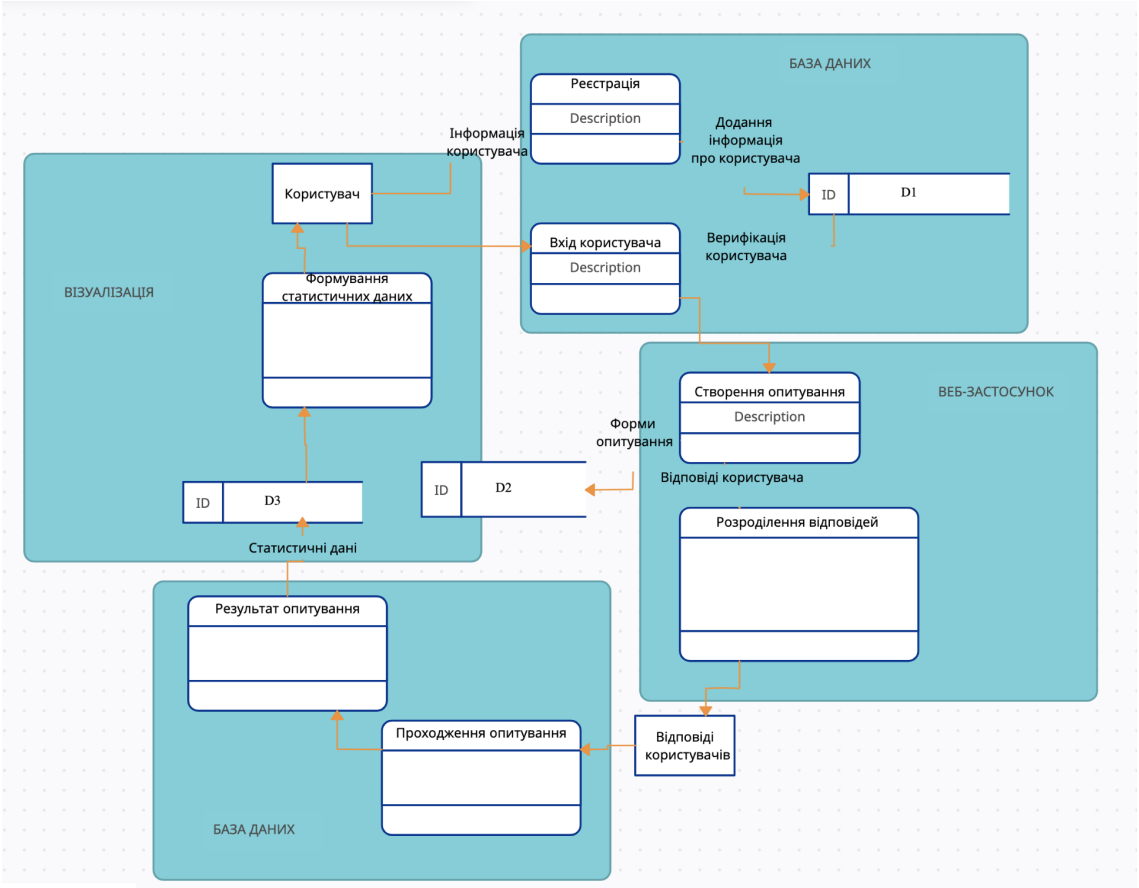

Рисунок 4.2 – Діаграма потоків даних веб-застосунку для роботи з опитуваннями

В даному випадку нас цікавлять лише дані згадані вище, тому головними етапами проходження потоків даних є етап з виведенням даних з БД, а також етап візуалізації у веб-застосунку.

4.3 База Даних

Передбачається, що додаток має в собі містити дані, щодо пройдених чи все ще активних опитувань, в яких варіанти вибору мають в собі декілька варіантів

відповіді, що відображає рівень задоволеності студентами чи викладачами певними аспектами їх освітньої діяльності. Студенти та викладачі також матимуть свої окремі опитування. В кінці кожного опитування також матиметься відкрите поле для написанні додаткового коментаря

Відповідно, нам необхідні такі таблиці даних: таблиця користувачів, таблиця опитувань, таблиця питань, таблиця відкритих відповідей та таблиця навчальних груп (рис. 4.3).

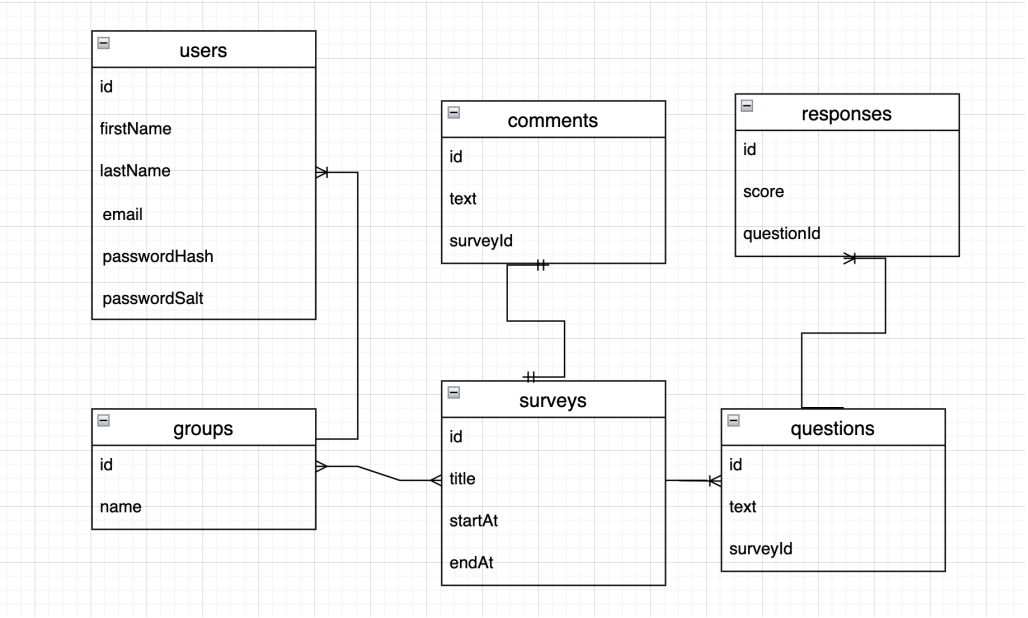

Рисунок 4.3 – Концептуальна модель бази даних

Також, оскільки наш додаток передбачає майбутню інтеграцію з загальною системою, то можна розширити кількість таблиць сформувати концептуальну модель БД в масштабі всієї системи, що буде включати також дані про користувачів (рис. 4.4).

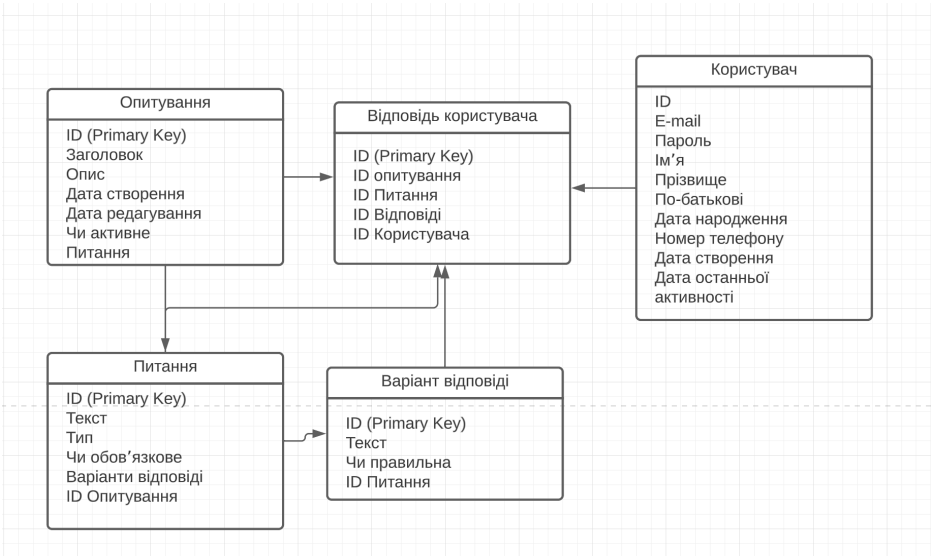

Рисунок 4.4 – Концептуальна модель БД у масштабі загальної системи

На основі концептуальної моделі БД була створена також логічна модель БД (рис. 4.5).

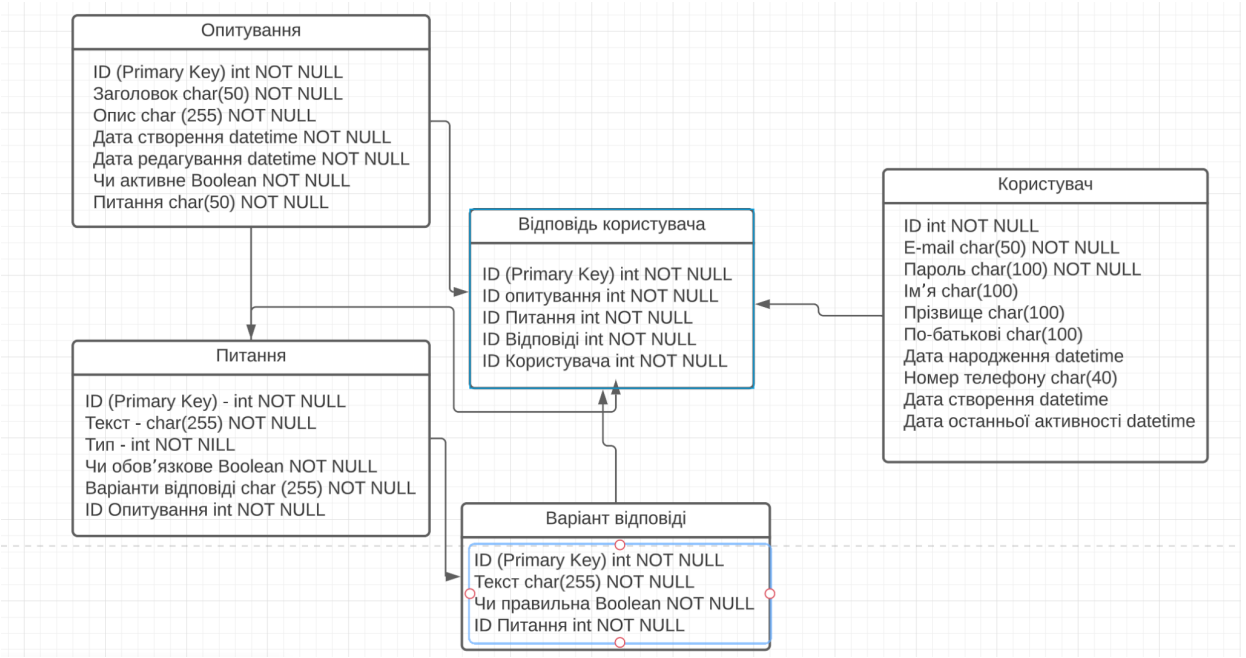

Рисунок 4.5 – Логічна модель БД

Маємо таку модель, яка буде реалізована за допомогою PostgreSQL. Самі таблиці та зв'язки між ними будуть створені за допомогою Objection JS [12]. Ця

бібліотека дає зможу працювати з базою даних об’єктно-орієнтовано. Для цього спочатку необхідно описати всі моделі даних, які у нас є, описуючи дельно типи полів та назви, а також зв’язки між ними для подальшого можливого об’єднання даних при поверненні з бази даних.

4.4 REST API

Оскільки база даних вже готова, наступним кроком є розробка REST API. Раніше ми вже описали моделі даних для БД, які також описувались в рамках нього. Для створення зв’язку з БД був написаний конфігураційний файл з даними для підключення та ініціалізований сам Objection JS всередині API [13]. Далі були створені репозиторії бази даних - це файли в яких саме і будуть зберігатись запити до БД окремо по кожній з моделей [14].

Наступним кроком були створені необхідні сервіси для обгортання цих запитів у необхідні функції, оскільки іноді для виконання однієї функції необхідно зробити не один запит у БД, і окремі функції роблять код більш зрозумілим та інтуїтивним (рис. 4.6).

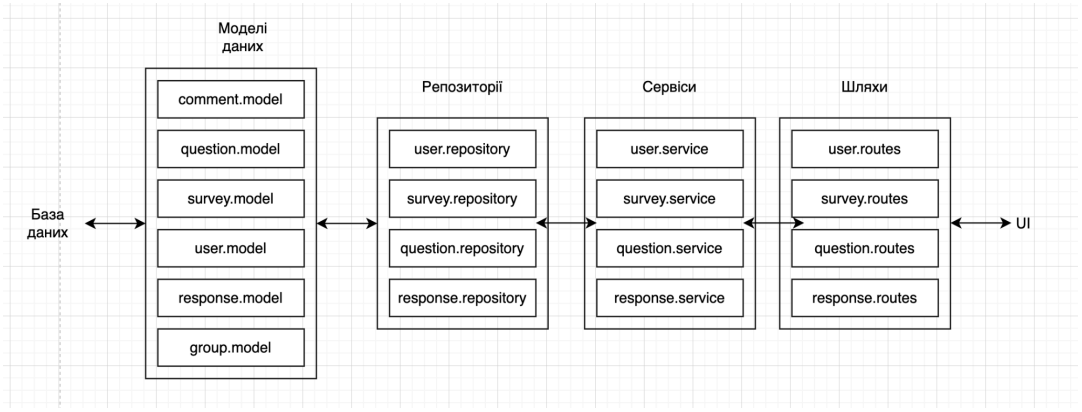

Рисунок 4.6 – Схема роботи REST API

Найважливішим кроком створенні REST API є прописання шляхів. В даному випадку для шляхів по різним моделям були створені окремі файли. В них вже були прописані шляхи для отримання тих чи інших даних [15]. В даному

випадку, були реалізовані шляхи для повернення спису опитувань, повернення даних окремого опитування, повернення списку питань, повернення даних по окремому питанню, повернення списку питань по id опитування, отримання списку відповідей, отримання списку відповідей по масиву id питань, отримання даних по окремій відповіді, повернення всіх користувачів (рис. 4.7).

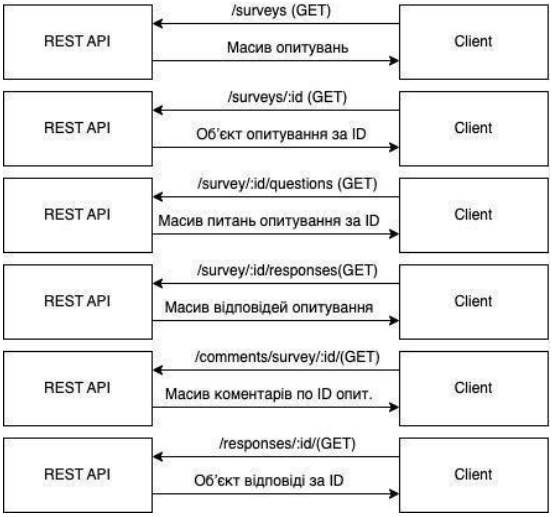

Рисунок 4.7 – Шляхи для отримання даних з REST API

Таким чином була реалізована REST API для отримання даних про опитування, яке може інтегруватись в загальну мікросервісну архітектуру та може мати й інші клієнти в залежності від потреби (рис. 4.8).

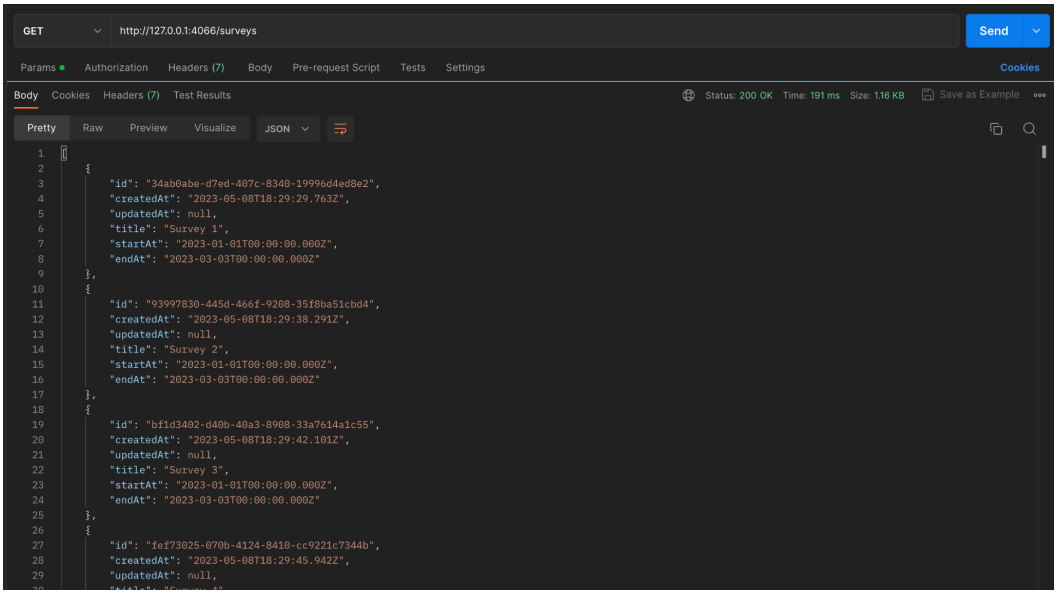

Рисунок 4.8 – Перевірка роботи REST API за допомогою програми Postman

4.5 Веб-клієнт

Для створення клієнтського UI для основи був взятий JavaScript фреймворк React. Тому після ініціалізації проекту були створені спочатку необхідні сторінки - найголовнішими є сторінка списку опитувань та сторінка результатів по окремому опитуванню. Після для реалізації компонентів було встановлено UI бібліотеку Material UI, оскільки однією з вимог до додатку було створення уніфікованого інтерфейсу. Використання UI бібліотеки якраз і вирішить питання уніфікації.

Наступним кроком є отримання даних від сервера. Для отримання даних від серверу була встановлена бібліотека Axios [16]. Вона надасть змогу високоефективно надсилати HTTP-запити з клієнту на сервер. Для збереження даних була також встановлена бібліотека Redux - вона відповідальна за створення глобального сховища даних всередині клієнтської частини, звідки можна взяти дані у будь-якій частині додатку. Після описання всіх необхідних шляхів для отримання даних та створення функцій для збереження даних у сховищі даних Redux, клієнта частина вже мала можливість отримати та зберігати дані з серверу [17].

Під час створення сховища стану за допомогою Redux було використано також методологію, що дозволяє розділити загальне сховище на окремі шматки - slices. В даному випадку створено два slices - один для зберігання даних про опитування для їх використання в додатку, а інший для зберігання даних про помилки, що можуть виникнути під час роботи додатку, і там зберігається масив цих помилок, а інтерфейс виводить їх для користувача.

Після того як з’явилась можливість отримувати дані від серверу був створений компонент для виведення списку опитувань. В ньому також були додані посилання на перегляд результатів по окремому опитуванню, опис з вказаними датами початку та закінчення опитування та статус опитування.

Далі на сторінці результатів по окремому опитуванню були виведені загальні дані про опитування, а також створені такі вкладки: “Загальний огляд”, “Питання”, “Користувачі”.

Оскільки однією з головних складових для відображення результатів є зображення діаграм, то була встановлена бібліотека Charts.js для React, яка підтримує велику кількість різних типів графіків [18].

На сторінці огляду також була виведена загальна статистика по результатам опитування та радарний графік для відображення загальних результатів по питанням.

Наступною була реалізація вкладки з питаннями і однією з головних цілей додатку було надання користувачеві вичерпного списку можливих відображень даних зручних для нього, для цього при рендерингу результатів по кожному з питань були додані кнопки, натисканням на які користувач має можливість змінювати різні види відображення даних, які включають графіки, таблиці та статистику (рис. 4.9).

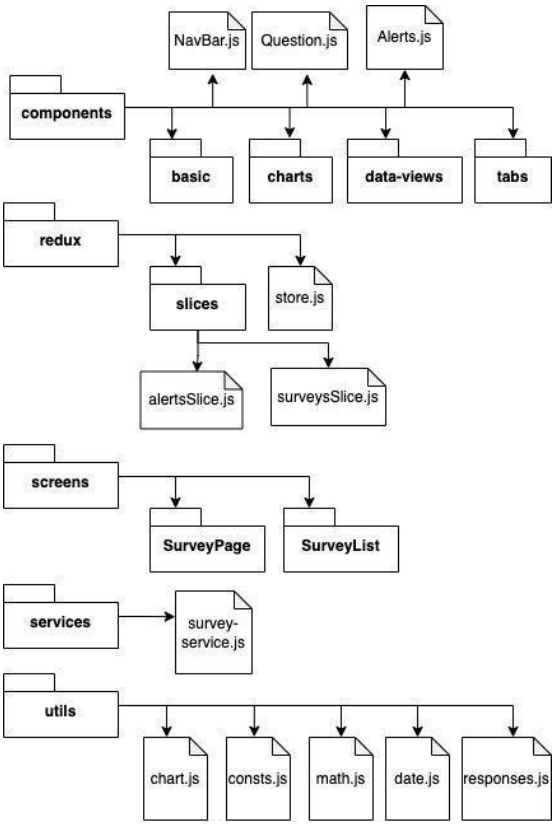

Рисунок 4.9 – Файлова діаграма клієнту

В файловій діаграмі клієнту можна також побачити пакет utils, в якому зберігаються всі додаткові функції необхідні для роботи додатку: функції для конвертації дат, функції для ініціалізації графіків, декомпозиції даних, необхідних математичних функцій та констант.

Окремо також можна виділити структуру компонентів, що використовуються на Front-end.

4.6 Діаграма пакетів

За допомогою діаграми пакетів ми можемо представити декомпозовану архітектуру нашого додатку (рис. 4.10). В першу чергу наш додаток складається з двох головних частин - клієнту та серверу.

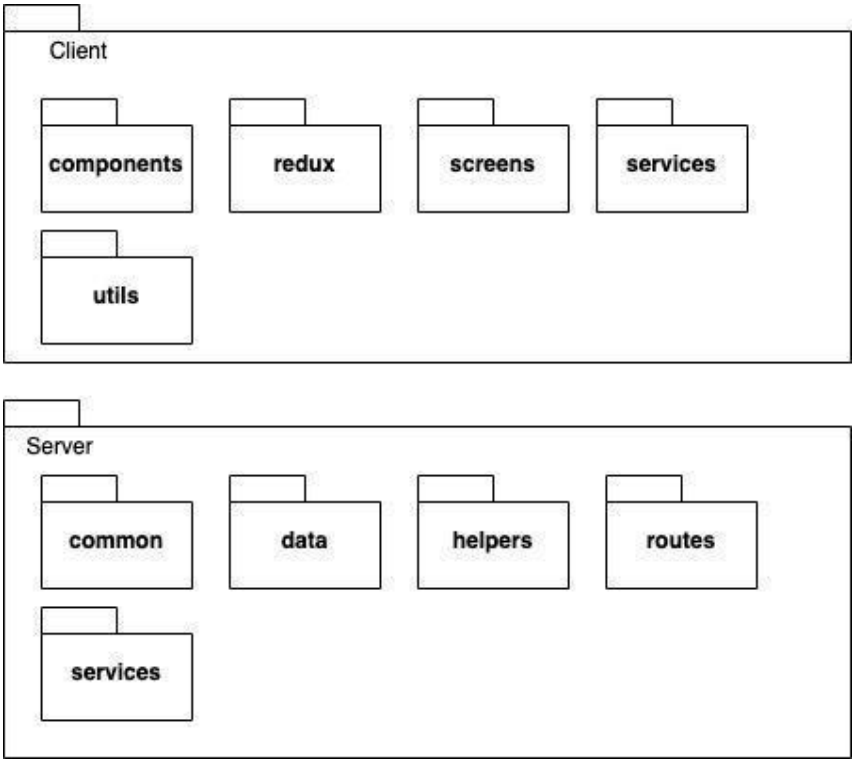

Рисунок 4.10 – Діаграма пакетів проекту

В серверній частині проекту ми можемо побачити п’ять головних пакетів - common, data, helpers, routes та services. Пакет common зберігає в собі загальні константи проекту, дані для підключень, переліки певних даних. Пакет data

зберігає в собі моделі даних нашого додатку - саме в ньому описуються моделі за допомогою Objection JS, що допоможе нам описати самі таблиці БД та зв’язки між ними. Пакет helpers містить в собі додаткові функції написані для спрощення роботи та керування БД. Пакет services містить в собі всі функції-сервіси необхідні для роботи з БД, саме в ньому описується логіка запитів до БД. Пакет routes містить в собі опис API шляхів та такого, які функції мають по ним виконуватись та які дані отримуватись чи повертатись.

Щодо клієнтної частини, то тут також можна побачити п’ять головних пакетів. Пакет components містить в собі окремі компоненти, що використовуються в інтерфейсі, і які можна перевикористовувати в різних частинах інтерфейсу. Пакет redux містить в собі опис та ініціалізацію загального сховища стану, який дає змогу отримувати доступ до даних з будь-якої частини додатку та є головним сховищем даних в клієнтській частині додатку. Пакет screens містить в собі опис окремих сторінок нашого веб-додатку та того, які дані та компоненти на кожній з них мають відображатись. Пакет services містить в собі функції для запиту даних з нашого REST API. Саме ці функції повертають даня з БД і потім зберігаються в нашому загальному сховищі. Останнім пакетом є utils, в ньому зберігаються всі допоміжні методи для реалізації функціоналу клієнтського веб-інтерфейсу.

Висновки до розділу

В цьому розділі було описано отриману реалізацію програмного продукту. Була продемонстрована діаграма прецедентів, що описує функціонал необхідний для роботи користувача з додатком. Також були описані вхідні дані додатку та опис їх проходження. В цьому розділі також описані моделі бази даних, архітектура отриманого REST API та клієнтського веб-додатку. Були продемонстровані діаграма пакетів, файлова діаграма клієнту, діаграма шляхів API та схема функціонування REST API.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

В цьому розділі приведені системні вимоги для роботи з додатком та сценарії роботи користувача з ним.

5.1 Системні вимоги

Користувач повинен мати сумісний веб-браузер, такий як Google Chrome, Mozilla Firefox, Microsoft Edge або Safari. Рекомендується використовувати останню доступну версію браузера для найкращої сумісності з сучасними веб-стандартами.

Для використання веб-додатка користувачеві потрібне активне Інтернет-підключення, оскільки взаємодія з сервером відбувається по мережі.

Оскільки додаток, адаптований під різні розміри девайсу, тому користувач може використовувати не лише десктопні комп’тери чи ноутбуки, а й планшети чи мобільні телефони.

5.2 Користувацький інтерфейс

Головний екран для перегляду результатів щодо опитування це сторінка переліку самих опитувань, що проводяться чи вже завершені (рис. 5.1).

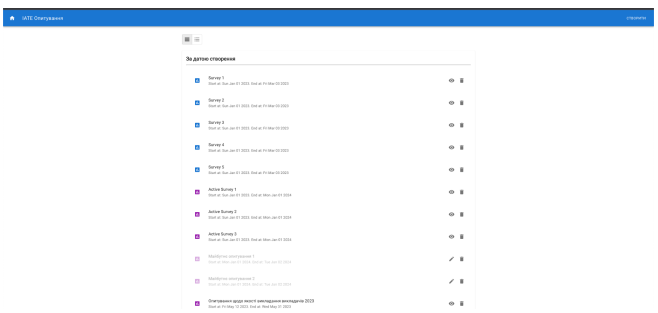

Рисунок 5.1 – Сторінка зі списком опитувань

В цьому списку опитувань користувач може переглянути всі наявні опитування. Також під кожним опитуванням вказані його деталі: в даному випадку дата початку опитування та дата завершення опитування. Індикатором того чи завершене опитування чи воно проводиться є колір іконки опитування праворуч від назви: синій колір - завершено, фіолетовий - проводиться. Також опитування, що ще не почались і будуть проводитись у майбутньому позначені сірим кольором, не є клікабельними і на їх сторінку з результатами неможливо перейти.

Також праворуч від кожного опитування є мобільне меню для базових дій над опитуваннями, а саме: “Переглянути”, “Редагувати”, “Видалити”.

Угорі над списком опитувань також є зміна перегляду списку опитувань. Оскільки перегляд списку опитувань за датами їх створення є зручним для адміністратора цих опитувань, проте для клієнта дата створення опитування не має значення і йому більш важливим є інтуїтивне розділення опитувань - в нашому випадку це було реалізовано за допомогою розділень на статуси (рис. 5.2).

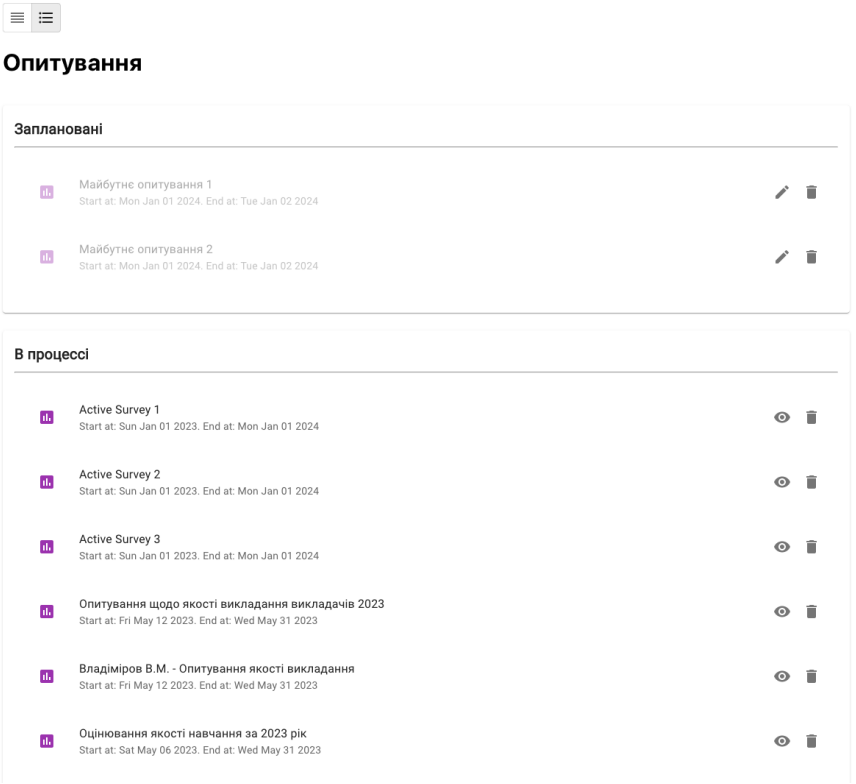

Рисунок 5.2 – Розділення списку опитувань на статуси

Натиснувши на кнопку “Переглянути” користувача переносить до сторінки результатів по окремому опитуванню (рис. 5.3).

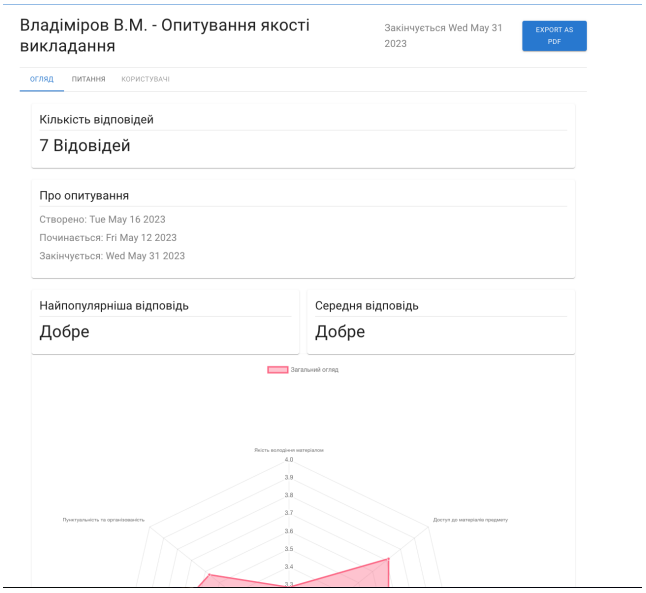

Рисунок 5.3 – Сторінка з загальним оглядом результатів опитування

На даній сторінці вказані також базова інформація щодо опитування. А також загальний огляд результатів. Огляд результатів включає: дату створення, дату початку та дату завершення, кількість відповідей користувачів, найпопулярнішу оцінку та середню оцінку по всім відповідям. Нижче також рендериться радарна-діаграма, що відображає загальні оцінки по всім питанням в опитуванні (рис. 5.4).

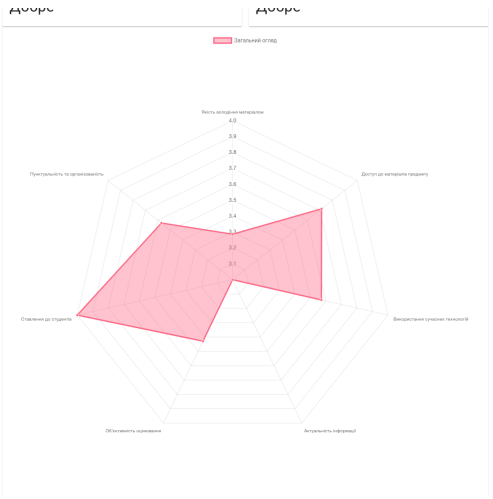

Рисунок 5.4 - Радарна діаграма, що відображає загальну оцінку опитування

Вище над результатами є також головна навігація по результатам цього опитування це є кнопки для переключення на такі вкладки: “Загальний огляд” та “Результати по питанням”.

При переході на вкладку результатів по питанням користувач потрапляє на сторінку з результатами по всім питанням (рис. 5.5).

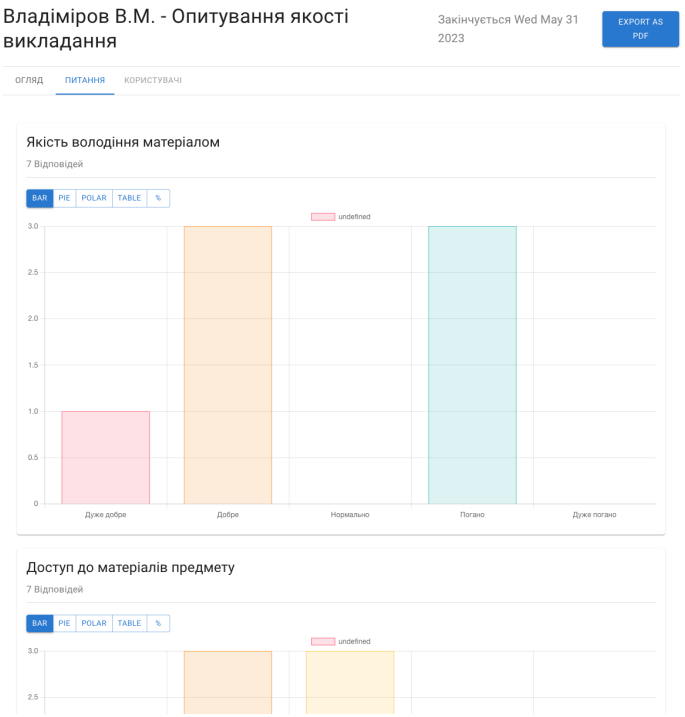

Рисунок 5.5 – Сторінка результатів опитування по питанням

Потрапляючи на сторінку результатів по питанням користувач бачить перелік усіх питань та відображення усіх результатів користувачів по кожному з питань. Користувач має можливість змінювати режим відображення результатів по кожному з питань окремо завдячи кнопкам у верхній частині питання: режим відображення за замовчуванням це графік у вигляді колонок.

Користувачу також доступні такі види відображення даних: радіальний графік і вигляді кола поділеного на частини, полярний графік, де коло поділене на рівні частини з різною довжиною в залежності від кількості відповідей певного типу, таблиця кількості відповідей по кожній з оцінок, а також таблиця статистики по відсотковому співвідношення кожної з оцінок до загальної кількості відповідей.

Вид відображення у вигляді діаграми колонок відображається за замовчуванням, оскільки такий вид діаграм є одним з найрозповсюдженіших, він є дуже інтуїтивним для будь-якого користувача, який хоче отримати якнайшвидше та якнайпростіше дані щодо результатів певного опитування і цей вид відображення дає йому змогу це зробити (рис. 5.6).

Вид відображення у вигляді радіального графіку, який також називається діаграма “пиріг” є також доволі корисним для використання. Він стоїть на одному рівні з стовпчиковою діаграмою по рівню розповсюдженості та інтуїтивності для користувачів. Він дає змогу отримати візуальне представлення частки кожної оцінки в загальній кількості відповідей (рис. 5.7).

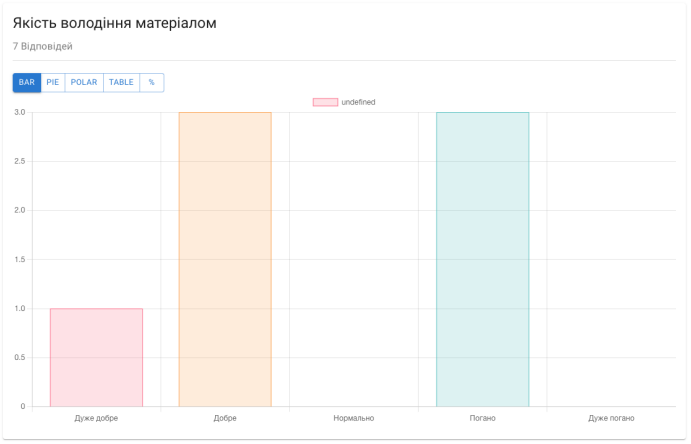

Рисунок 5.6 – Вид відображення “Стовпчикова діаграма”

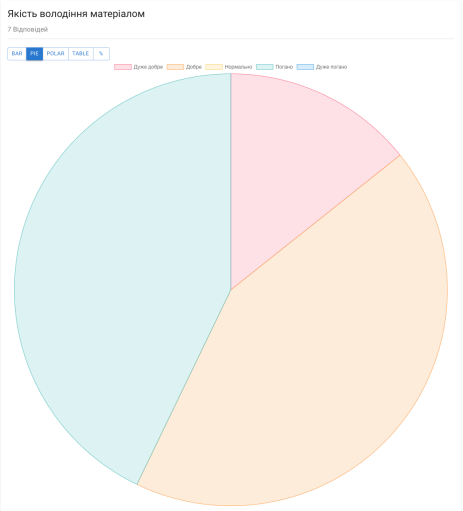

Рисунок 5.7 Вид відображення “Діаграма пиріг”

Вид відображення з полярною діаграмою дуже схожий на минулу діаграму “пиріг” через свій радіальний вид відображення, проте має суттєву різницю, оскільки якщо діаграма “пиріг” відображає лише частку від загальної кількості, то полярна діаграма має за мету показати саме кількісну різницю між кожною оцінкою, це досягається завдяки візуальному відображенню завдяки довжині часток, а також числовими направляючими, що дають змогу орієнтуватись на числовій діаграмі (рис. 5.8).

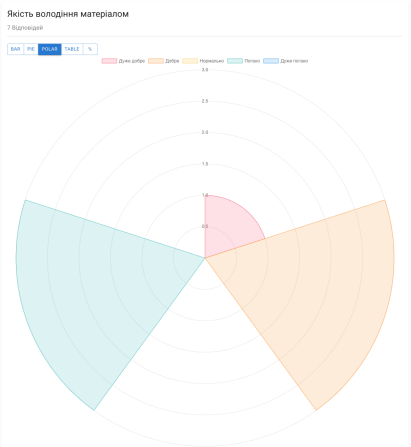

Рисунок 5.8 – Вид відображення “Полярна діаграма”

Вид відображення зі звичайною таблицею є найпростішим видом відображення (рис. 5.9). Такий вид відображення є в будь-якій подібній системі, також так дані представляють у звітах та документах, оскільки це є найбільш точним видом подання інформації і він ґрунтується на фактичних числах відповідей для кожної оцінки.

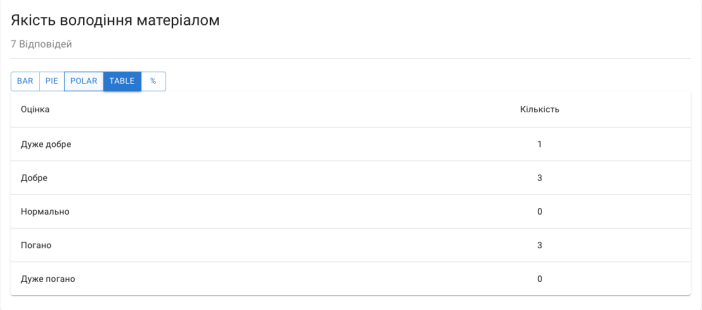

Рисунок 5.9 – Вид відображення “Таблиця”

Вид відображення таблиці з відсотковим співвідношенням відрізняється від минулого лише способом подачі даних, оскільки якщо минулий відображав просто кількість відповідей по певній оцінці, то цей відображає частку, що є також корисно при формуванні результатів будь-якого опитування, оскільки саме на частках ґрунтуються кожні результати опитування (рис. 5.10).

Рисунок 5.10 – Результати щодо окремого опитування у вигляді відсоткового співвідношення

Таким чином, у користувача є доволі великий перелік можливостей для перегляду то відображення даних щодо опитування. Користувачу надається динамічний інтерфейс, в якому він може сам обирати, яким способом він може переглядати дані включно з графіками.

Також, була створена функція експорту даних. В даному випадку кнопку “Export to PDF” можна побачити у верхньому правому куті сторінки перегляду результатів опитування на будь-якій з вкладок. При її натисканні користувачеві запропонується обрати шлях та назву для файлу, що збережеться на його комп’ютері. Експортований файл міститиме загальні відомості про дане опитування, а також середню відповідь по кожному з питань (рис. 5.11).

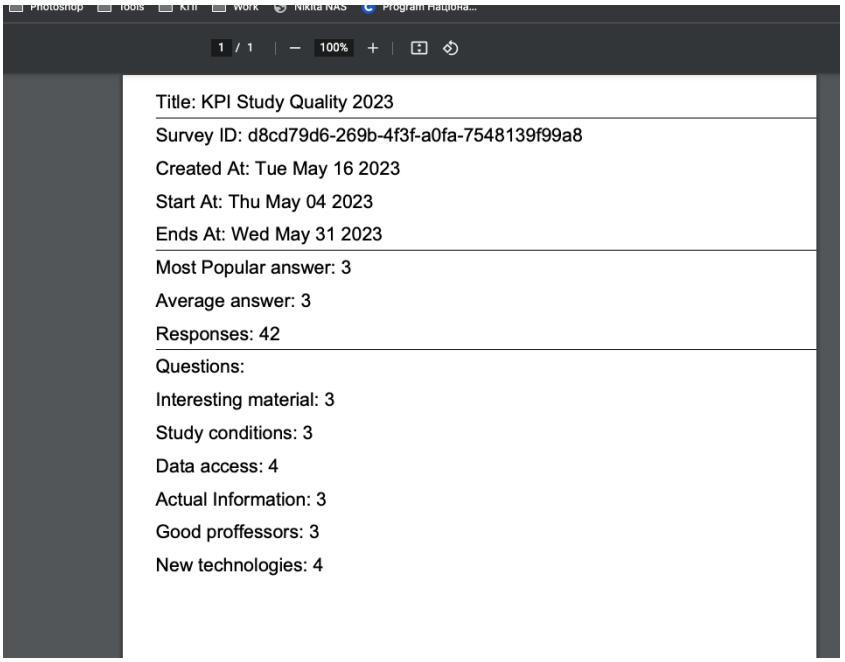

Рисунок 5.11 – Експортований PDF файл.

5.3 Інтеграція з загальною системою

Розділ інтеграції мікросервісу проведення опитувань у загальну систему є важливою складовою вашої дипломної роботи. Цей мікросервіс складається з серверної частини REST API, бази даних PostgreSQL і веб-клієнта на React. Основна мета цього підрозділу - описати ефективну інтеграцію вашого мікросервісу в загальну систему.

В першу чергу, при інтеграції цього додатку з загальною системою необхідно знайти місце в загальній архітектурі системи, а також вирішити проблему з тим, де цей мікросервіс буде хоститись. В даному випадку, наш мікросервіс має бути розгорнутий в окремому докер-файлі у шарі з усіма іншими мікросервісами, що входять до складу загальної системи. Проте, це стосується лише Back-end частини додатку, тобто REST API (рис. 5.12).

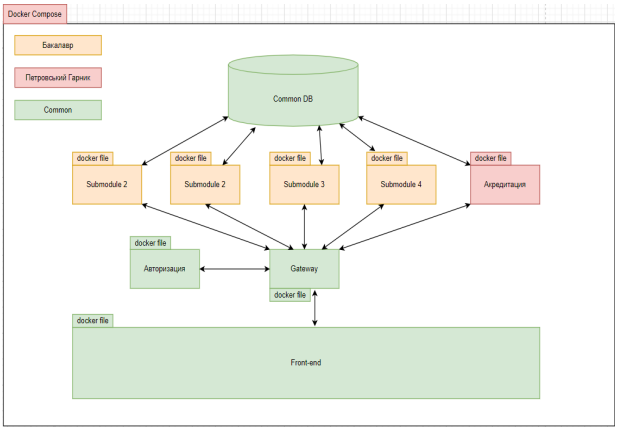

Рисунок 5.12 – Загальна архітектура системи

Наступним важливим кроком, є налагодження зв’язку з центральною базою даних, перевірку з’єднання з нею. Необхідно перевірити чи всі таблиці необхідні для роботи мікросервісу були коректно створені та працюють без помилок.

Наступним, проте не менш важливим кроком, є інтеграція з роботою загальної авторизації, оскільки використання мікросервісу передбачає ролі користувачів та різні юзер-кейси, тому необхідно перевірити правильну роботу системи авторизації, а також інтегруватись з таблицею користувачів загальною системи та створити з нею зв’язки з таблиць даного мікросервісу.

Останнім важливим кроком є інтеграція фронт-енд частини мікросервісу. Цей етап є одним з фінальних, оскільки спочатку необхідно налагодити комунікацію мікросервісів та бази даних. Інтерфейсна частина цього додатку є найважливішою його частиною, оскільки саме вона відповідає за формування даних щодо опитування. Тому, тут є дві варіації щодо вирішення цього завдання - інтеграція в загальний фронт-енд системи, чи створення окремого під-клієнту.

Перший варіант є найбільш правильним, оскільки не ускладнює архітектуру системи, роблячи її структуру зрозумілою та легкою в підтримці, хоча він має більші часові витрати в реалізації. Оскільки перед розробкою мікросервісів були досягнуті певні домовленості з приводу використання технологій інтерфейсу, а саме написання інтерфейсу на React, суттєва спрощує цю інтеграцію оскільки мікросервіс написаний на тій самій технології, що і загальна система.

Інше рішення, яке було прийняте для майбутньої інтеграції з загальним фронт-енд - це використання UI бібліотеки, замість написанні унікальних стилів. Хоча, написання унікальних стилів є простішим у реалізації, проте якщо кожна з частин інтерфейсу буде написана з використанням унікальних стилів, це зробить її абсолютно не гнучкою, важкою в підтримці та різною за виглядом та методологіям. По-перше, використання бібліотеки вирішує це питання, оскільки їх використання є інтуїтивним для будь-якого розробника, що спрощує підтримку. По-друге, більшість бібліотек використовує схожу систему компонентів, що дає змогу при мінімальних витратах замінить компоненти однієї бібліотеки на компоненти іншої. По-третє, була використана бібліотека Material UI, що є однією з найпопулярніших, і є інтуїтивною як для користувачів, так і для розробників і вона не має агресивних стилів що занадто сильно виділяються з інших інтерфейсів, що дозволяє лишить ці компоненти в загальному інтерфейсі.

Проте, існує також і інший підхід, це створення окремого під-клієнту, який буде окремо запущений від загального інтерфейсу, проте може мати загальні його частини, як наприклад навігаційну панель чи авторизацію. Він може бути на піддомені, і бути окремою підсистемою, загальної системи, як наприклад це зроблено в мікросервісах від Google.

Висновки до розділу

В цьому розділі були описані системні вимоги для використання додатку. Також тут був описаний інтерфейс додатку, способи його використання та взаємодії з ним. Були детально описані всі види відображення даних та спосіб взаємодії з ними, а також переваги використання тих чи інших видів відображення даних.

Також, в цьому розділі була описана стратегія інтеграції цього додатку, як мікросервісу в загальній системі. Тут були надані рекомендації з інтеграції та можливі її варіанти.

ВИСНОВКИ

У ході виконання дипломної роботи було розроблено клієнт-серверний додаток для формування матеріалів щодо опитувань та візуалізації їх результатів в удобному веб-інтерфейсі. Робота включає комплексне дослідження, в якому були проаналізовані вимоги до системи, вивчені сучасні технології та інструменти, розроблена архітектура та дизайн системи, реалізована система, проведене тестування та валідація.

Додаток, що було створено в рамках цієї роботи, демонструє значну практичну корисність, оскільки він забезпечує ефективний збір і аналіз даних опитувань, що є важливим інструментом для забезпечення якості роботи кафедри.

Робота також включає оцінку ефективності розробленого додатка і визначення напрямків його подальшої інтеграції. Незважаючи на те, що додаток вже демонструє високу продуктивність і зручність використання, було визначено декілька можливих напрямків для подальшого удосконалення, включаючи додавання нових функцій, покращення інтерфейсу користувача та оптимізація продуктивності.

Ця робота підтверджує важливість використання сучасних інформаційних технологій для вирішення практичних задач і викликів, пов'язаних з опитуваннями. Результати, отримані в рамках цієї роботи, можуть бути використані для розвитку подібних систем у майбутньому, а також можуть служити основою для технічних досліджень у сфері використання технологій для збору і аналізу даних.

Під час практики успішно виконано всі поставлені завдання, отримано досвід розробки програмного забезпечення і глибоке розуміння розробки клієнт-серверних веб-додатків та роботою з даними щодо опитування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Паніотто В., Харченко Н. . Методи опитування. Київ: КМА, 2017. 342 с.
2. Річардсон Л., Ам’юндсен М., Рубі С. REST-фул Веб API: Сервіси для зміни світу. Ньютон: O'Reilly Media, 2013. 406 с.
3. What Is A RESTful API? URL: https://aws.amazon.com/what-is/restful-api/?nc1=h_ls (дата звернення: 12.05.23)
4. Patterns for API Design: Simplifying Integration with Loosely Coupled Message Exchanges. / Zimmermann O., Stocker M., Lubke D. (2023). Контрада Ротолі: Print2print, 2023. 544 с.
5. Zammetti F. Modern Full-Stack Development.. Нью-йорк: Apress, 2020. 373 с.
6. Роберт С. М. Чиста архітектура. Київ: Фабула, 2019. 368 с.
7. Best survey tools of 2023. URL: https://www.techradar.com/best/best-survey-tools (дата звернення: 20.05.23)
8. TypeForm Documentation. URL: https://www.typeform.com/developers/get-started/ (дата звернення: 20.05.23)
9. КПІ СОВА. URL: https://t.me/analyticsKPI (дата звернення: 20.05.23)
10. Stefanov S. React: Up & Running: Building Web Applications. Ньютон: O’Reilly Media, 2021. 222 с.
11. Fastify Getting started. URL: https://www.fastify.io/docs/latest/Guides/Getting-Started/ (дата звернення: 19.05.23)
12. Designing Interfaces. / Tidwell J., Brewer C., Valencia A. (2020).. Ньютон: O’Reilly Media, 2020. 600 с.
13. Сучасні клієнт-серверні технології та їх застосування при вивченні систем управління базами даних. URL: https://fi.npu.edu.ua/files/Zbirnik_KOSN/12/29.pdf (дата звернення: 10.05.23)

14. Objection JS Models. URL: https://vincit.github.io/objection.js/guide/models.html (дата звернення: 20.05.23)
15. Node.js в дії. / Кантелон М., Хартер М., Головайчук Т., Райлих Н. (2018).. Нью-Йорк: Manning Publications, 2013. 416 с.
16. Axios API. URL: https://axios-http.com/uk/docs/api_intro (дата звернення: 19.05.23)
17. Redux Overview and Concepts. URL: https://redux.js.org/tutorials/essentials/part-1-overview-concepts (дата звернення: 18.05.23)
18. Charts.js Configuration. URL: https://www.chartjs.org/docs/latest/configuration/ (дата звернення: 19.05.23)

ДОДАТОК А

Формування даних щодо опитування

Текст програми

Аркушів 20

Київ - 2023

index.ts

```
import FastifyCors from '@fastify/cors';
import FastifyFormBody from '@fastify/formbody';
import fastify, { FastifyPluginAsync } from 'fastify';
import Knex from 'knex';
import { Model } from 'objection';
import knexConfig from '../knexfile';
import { initRoutes } from './routes';


Model.knex(Knex(knexConfig));

const app: FastifyPluginAsync = async (fastify, opts): Promise<void> => {
  await fastify.register(FastifyCors, {});

  await fastify.register(FastifyFormBody);

  await fastify.register(initRoutes, opts);
};

const instance = fastify({
  logger: {
    level: 'debug',
  },
});

instance.register(app);

export default app;
export { app, instance };
```

comment.routes.ts

```
import {FastifyPluginAsync, FastifyRequest} from 'fastify';
import { HttpCode, HttpMethod } from '@/common/enums';
import { commentService } from '@/services';

type Options = {
  services: {
    comment: typeof commentService;
  };
};

export const initCommentRoutes: FastifyPluginAsync<Options> = async (fastify, options) => {
  const { comment } = options.services;

  fastify.route({
    method: HttpMethod.GET,
    url: '/',
    handler: async (req, rep) => {
      const result = await comment.getAll();
      rep.status(HttpCode.OK).send(result);
    },
```

```
  });

  fastify.route({
    method: HttpMethod.GET,
    url: '/:id',
    handler: async (req: FastifyRequest<{ Params: { id: string } }>, rep) => {
      const id = req.params.id
      const result = await comment.getOne(id);
      rep.status(HttpCode.OK).send(result);
    },
  });

  fastify.route({
    method: HttpMethod.GET,
    url: '/survey/:id',
    handler: async (req: FastifyRequest<{ Params: { id: string } }>, rep) => {
      const id = req.params.id
      const result = await comment.getBySurveyId(id);
      rep.status(HttpCode.OK).send(result);
    },
  });

  fastify.route({
    method: HttpMethod.POST,
    url: '/',
    handler: async (req, rep) => {
      const result = await comment.create(req.body as any);
      rep.status(HttpCode.OK).send(result);
    },
  });

};
```

question.routes.ts

```
import {questionService} from "@/services";
import {FastifyPluginAsync, FastifyRequest} from "fastify";
import {HttpCode, HttpMethod} from "@/common/enums";

type Options = {
    services: {
        question: typeof questionService;
    };
};

export const initQuestionRoutes: FastifyPluginAsync<Options> = async (fastify, options) => {
    const { question } = options.services;

    fastify.route({
        method: HttpMethod.GET,
        url: '/',
        handler: async (req: FastifyRequest<{ Params: { id: string } }>, rep) => {
            const result = await question.getAll();
            rep.status(HttpCode.OK).send(result);
```

```
        },
    });

    fastify.route({
        method: HttpMethod.GET,
        url: '/:id',
        handler: async (req: FastifyRequest<{ Params: { id: string } }>, rep) => {
            const questionId = req.params.id;

            const result = await question.getOne(questionId);
            rep.status(HttpCode.OK).send(result);
        },
    });

};
```

response.routes.ts

```
import {FastifyPluginAsync, FastifyRequest} from 'fastify';
import { HttpCode, HttpMethod } from '@/common/enums';
import { responseService } from '@/services';

type Options = {
  services: {
    response: typeof responseService;
  };
};

export const initResponseRoutes: FastifyPluginAsync<Options> = async (fastify, options) => {
  const { response } = options.services;

  fastify.route({
    method: HttpMethod.GET,
    url: '/',
    handler: async (req, rep) => {
      const result = await response.getAll();
      rep.status(HttpCode.OK).send(result);
    },
  });

  fastify.route({
    method: HttpMethod.GET,
    url: '/:id',
    handler: async (req: FastifyRequest<{ Params: { id: string } }>, rep) => {
      const id = req.params.id
      const result = await response.getOne(id);
      rep.status(HttpCode.OK).send(result);
    },
  });

};
```

survey.routes.ts

```
import {FastifyPluginAsync, FastifyRequest} from 'fastify';
import { HttpCode, HttpMethod } from '@/common/enums';
import {questionService, surveyService} from '@/services';

type Options = {
  services: {
    survey: typeof surveyService;
    question: typeof questionService;
  };
};

export const initSurveyRoutes: FastifyPluginAsync<Options> = async (fastify, options) => {
  const { survey } = options.services;
  const { question } = options.services

  fastify.route({
    method: HttpMethod.GET,
    url: '/',
    handler: async (req, rep) => {
      const result = await survey.getAll();
      rep.status(HttpCode.OK).send(result);
    },
  });

  fastify.route({
    method: HttpMethod.GET,
    url: '/:id',
    handler: async (req: FastifyRequest<{ Params: { id: string } }>, rep) => {
      const id = req.params.id;

      const result = await survey.getOne(id);
      rep.status(HttpCode.OK).send(result);
    },
  });

  fastify.route({
    method: HttpMethod.GET,
    url: '/:id/questions',
    handler: async (req: FastifyRequest<{ Params: { id: string } }>, rep) => {
      const id = req.params.id;


      const result = await question.getBySurveyId(id);
      rep.status(HttpCode.OK).send(result);
    },
  });

  fastify.route({
    method: HttpMethod.GET,
    url: '/:id/responses',
    handler: async (req: FastifyRequest<{ Params: { id: string } }>, rep) => {
      // const id = req.params.id;
```

```
      const result = "Responses";
      rep.status(HttpCode.OK).send(result);
    },
  });

};
```

survey.model.ts

```
import { WithUpdatedAtModel } from './with-updated-at.model';
import {DbTableName} from "@/common/enums";
import {Model, RelationMappings} from "objection";
import {QuestionModel} from "@/data/models/question.model";

export class SurveyModel extends WithUpdatedAtModel {
  'title': string;
  'startAt': string;
  'endAt': string;

  public static override get tableName(): string {
    return DbTableName.SURVEYS;
  }

  public static override get relationMappings(): RelationMappings {
    return {
      questions: {
        relation: Model.HasManyRelation,
        modelClass: QuestionModel,
        join: {
          from: `${DbTableName.SURVEYS}.id`,
          to: `${DbTableName.QUESTIONS}.surveyId`,
        },
      },
    };
  }


}
```

question.model.ts

```
import { WithUpdatedAtModel } from './with-updated-at.model';
import {DbTableName} from "@/common/enums";
import {Model, RelationMappings} from "objection";
import {SurveyModel} from "@/data/models/survey.model";
import {ResponseModel} from "@/data/models/response.model";

export class QuestionModel extends WithUpdatedAtModel {
  'text': string;
  'surveyId': string;

  public static override get tableName(): string {
```

```
    return DbTableName.QUESTIONS;
  }

  public static override get relationMappings(): RelationMappings {
    return {
      user: {
        relation: Model.BelongsToOneRelation,
        modelClass: SurveyModel,
        join: {
          from: `${DbTableName.QUESTIONS}.survey_id`,
          to: `${DbTableName.SURVEYS}.id`,
        },
      },
      responses: {
        relation: Model.HasManyRelation,
        modelClass: ResponseModel,
        join: {
          from: `${DbTableName.QUESTIONS}.id`,
          to: `${DbTableName.RESPONSES}.questionId`
        }

      }
    };
  }


}
```

alertsSlice.js

```
import { createSlice } from '@reduxjs/toolkit';
import {v4 as uuid} from 'uuid';

const alertSlice = createSlice({
    name: 'alerts',
    initialState: [],
    reducers: {
        setAlert: (state, action) => {
            const { message, alertType } = action.payload;
            const id = uuid()
            state.push({ message, alertType, id});
        },
        removeAlert: (state, action) => {
            const id = action.payload;
            return state.filter((alert) => alert.id !== id);
        },
    },
});

export const { setAlert, removeAlert } = alertSlice.actions;
export default alertSlice.reducer;
```

surveysSlice.js

```
import { createSlice } from '@reduxjs/toolkit';

export const initialState = {
  surveys: [],
  survey: {},
  questions: [],
  comments: {},
};

export const surveysSlice = createSlice({
  name: 'surveys',
  initialState,
  reducers: {
    setSurveys: (state, { payload }) => ({ ...state, surveys: payload }),
    setSurvey: (state, { payload }) => ({...state, survey: payload}),
    setQuestions: (state, { payload }) => ({...state, questions: payload}),
    setComments: (state, { payload }) => ({...state, comments: payload}),
  },
});

export const { setSurveys, setSurvey, setQuestions, setComments } = surveysSlice.actions;
export default surveysSlice.reducer;
```

survey-service.js

```
import axios from "axios";
import {setComments, setQuestions, setSurvey, setSurveys} from "../redux/slices/surveysSlice";
import {setAlert} from "../redux/slices/alertsSlice";

axios.defaults.baseURL = 'http://localhost:4066'

export const fetchSurveys = () => async (dispatch) => {
  try {
    const { data } = await axios.get('/surveys/')
    dispatch(setSurveys(data));
  } catch (err) {
    dispatch(setAlert({ message: 'Server error! Cannot get list of surveys.', alertType:
'error'}))
  }
};

export const getSurveyById = (payload) => async (dispatch) => {
  try {
    const { data } = await axios.get(`/surveys/${payload}`)
    dispatch(setSurvey(data))
  } catch (err) {
    dispatch(setAlert({ message: 'Server error! Cannot get survey info.', alertType: 'error'}))
  }
}

export const getSurveyQuestions = (payload) => async (dispatch) => {
```

```
  try {
    const { data } = await axios.get(`/surveys/${payload}/questions`)
    dispatch(setQuestions(data))
  } catch (err) {
    dispatch(setAlert({ message: 'Server error! Cannot get list of survey questions.',
alertType: 'error'}))
  }
}

export const createSurveyComment = (payload) => async (dispatch) =>  {
  try {
    await axios.post('/comments', payload)
    return true;
  } catch (err) {
    alert(err.response?.data.message)
    console.error(err);
  }
};

export const getSurveyComment = (payload) => async (dispatch) => {
  try {
    const { data } = await axios.get(`/comments/survey/${payload}`)
    dispatch(setComments(data))
  } catch (err) {
    dispatch(setAlert({ message: 'Server error! Cannot get survey comments.', alertType:
'error'}))
  }
}

export const createSurveys = (payload) => async (dispatch) => {
  try {
    await axios.post('/surveys/', payload)
    return true;
  } catch (err) {
    alert(err.response?.data.message)
    console.error(err);
  }
};

export const editSurvey = (payload) => async (dispatch) => {
  try {
    await axios.put('/surveys/', payload)
    return true;
  } catch (err) {
    alert(err.response?.data.message)
    console.error(err);
  }
};

export const deleteSurvey = (payload) => async (dispatch) => {
  try {
    await axios.delete(`/surveys/${payload}`)
    return true;
  } catch (err) {
```

```
    alert(err.response?.data.message)
    console.error(err);
  }
};

export const createSurveyResponses = (payload) => async (dispatch) =>  {
  try {
    await axios.post('/responses', payload)
    return true;
  } catch (err) {
    alert(err.response?.data.message)
    console.error(err);
  }
};
```

chart.js

```
export const getChartConfig = (scores) => {
    return {
        labels: ['Дуже добре', 'Добре', 'Нормально', 'Погано', 'Дуже погано'],
        datasets: [{
            data: [
                scores.reduce((count, value) => (value === 5 ? count + 1 : count), 0),
                scores.reduce((count, value) => (value === 4 ? count + 1 : count), 0),
                scores.reduce((count, value) => (value === 3 ? count + 1 : count), 0),
                scores.reduce((count, value) => (value === 2 ? count + 1 : count), 0),
                scores.reduce((count, value) => (value === 1 ? count + 1 : count), 0)
            ],
            backgroundColor: [
                'rgba(255, 99, 132, 0.2)',
                'rgba(255, 159, 64, 0.2)',
                'rgba(255, 205, 86, 0.2)',
                'rgba(75, 192, 192, 0.2)',
                'rgba(54, 162, 235, 0.2)'
            ],
            borderColor: [
                'rgb(255, 99, 132)',
                'rgb(255, 159, 64)',
                'rgb(255, 205, 86)',
                'rgb(75, 192, 192)',
                'rgb(54, 162, 235)'
            ],
            borderWidth: 1
        }]
    }
}

export const getRadarConfig = (labels, data) => {
    return {
        labels: labels,
        datasets: [{
            label: 'Загальний огляд',
            data: data,
```

```
            fill: true,
            backgroundColor: 'rgba(255, 99, 132, 0.4)',
            borderColor: 'rgb(255, 99, 132)',
            pointBackgroundColor: 'rgb(255, 99, 132)',
            pointBorderColor: '#fff',
            pointHoverBackgroundColor: '#fff',
            pointHoverBorderColor: 'rgb(255, 99, 132)'
        }]
    }
}
```

math.js

```
export const countElement = (array, element) =>  {
    return array.reduce((count, value) => (value === element ? count + 1 : count), 0)
}

export const getPercent = (array, element) => {
    return Math.round( countElement(array, element)/ array.length * 100) + ' %'
}

export const getAverage = (array) => {
    return Math.round(array.reduce((partialSum, a) => partialSum + a, 0) / array.length)
}

export const getAverageNotRounded = (array) => {
    return array.reduce((partialSum, a) => partialSum + a, 0) / array.length
}

export const getMostPopular = (array) => {
    return array.sort((a,b) =>
        array.filter(v => v===a).length
        - array.filter(v => v===b).length
    ).pop();
}
```

responses.js

```
export const getScores = (responses) => {
    return responses.map(response => response.score)
}

export const getTextResponse = (response_number) => {
    switch (response_number) {
        case 1:
            return 'Дуже погано'
        case 2:
            return 'Погано'
        case 3:
            return 'Нормально'
        case 4:
            return 'Добре'
        case 5:
```

```
            return 'Дуже добре'
        default:
            return ''
    }
}

export const responsesList = [
    { score: 5, label: 'Дуже добре' },
    { score: 4, label: 'Добре' },
    { score: 3, label: 'Нормально' },
    { score: 2, label: 'Погано' },
    { score: 1, label: 'Дуже погано' },
]
```

ChartsView.js

```
import React from 'react';
import {Bar, Pie, PolarArea} from "react-chartjs-2";
import {
    Chart as ChartJS,
    ArcElement,
    Tooltip,
    Legend,
    CategoryScale,
    LinearScale,
    PointElement,
    LineElement,
    BarElement,
    RadialLinearScale,
    Filler
} from "chart.js";
import {getChartConfig} from "../../utils/chart";
ChartJS.register(ArcElement, Tooltip, Legend, CategoryScale, LinearScale, PointElement,
LineElement, BarElement,    RadialLinearScale, Filler);

const ChartView = ({responses, type}) => {

    const getScores = (responses) => {
        return responses.map(response => response.score)
    }

    const scores = getScores(responses)

    const resultData = getChartConfig(scores)

    return (
        <div>
            {type === 'bar' && (
                <Bar data={resultData} />
            )}
            {type === 'pie' && (
                <Pie data={resultData} />
            )}
            {type === 'polar' && (
```

```
                <PolarArea data={resultData} />
            )}
        </div>
    );
};

export default ChartView;
```

TableView.js

```
import React from 'react';
import {Paper, Table, TableBody, TableCell, TableContainer, TableHead, TableRow} from
"@mui/material";
import {countElement} from "../../utils/math";

const TableView = ({responses, func, columnName}) => {

    const getScores = (responses) => {
        return responses.map(response => response.score)
    }

    const scores = getScores(responses)

    function createData(label, value) {
        return { label, value };
    }

    const rows = [
        createData('Дуже добре', func(scores, 5)),
        createData('Добре', func(scores, 4)),
        createData('Нормально', func(scores, 3)),
        createData('Погано', func(scores, 2)),
        createData('Дуже погано', func(scores, 1))
    ];

    return (
        <TableContainer component={Paper}>
            <Table sx={{ minWidth: 650 }} aria-label="simple table">
                <TableHead>
                    <TableRow>
                        <TableCell>Оцінка</TableCell>
                        <TableCell align="center">{columnName}</TableCell>
                    </TableRow>
                </TableHead>
                <TableBody>
                    {rows.map((row) => (
                        <TableRow
                            key={row.label}
                            sx={{ '&:last-child td, &:last-child th': { border: 0 } }}
                        >
                            <TableCell component="th" scope="row">
                                {row.label}
                            </TableCell>
                            <TableCell align="center">{row.value}</TableCell>
```

```
                        </TableRow>
                    ))}
                </TableBody>
            </Table>
        </TableContainer>
    );
};

export default TableView;
```

OverviewTab.js

```
import React from 'react';
import {Card, CardContent, Divider, Grid, Typography} from "@mui/material";
import {getAverage, getMostPopular} from "../../utils/math";
import RadarChart from "../charts/RadarChart";
import {getScores, getTextResponse} from "../../utils/responses";
import { getDate } from "../../utils/date";


const OverviewTab = ({questions, survey}) => {

    const responses = questions.map(question => question.responses).flat();
    const scores = getScores(responses)

    return (
        <div>
            <Card>
                <CardContent>
                    <Typography variant="h5" component="div" style={{marginBottom: 5}}>
                        Кількість відповідей
                    </Typography>
                    <Divider />
                    <Typography variant="h4" component="div" style={{marginTop: 10}}>
                        {responses.length === 0 ? 0 : Math.round(responses.length /
questions.length)} Відовідей
                    </Typography>
                </CardContent>
            </Card>
            <Card style={{marginTop: 24}}>
                <CardContent>
                    <Typography variant="h5" component="div" style={{marginBottom: 5}}>
                      Про опитування
                    </Typography>
                    <Divider />
                    <Typography variant="h6" color="grey" component="div" style={{marginBottom:
5, marginTop: 10}}>
                        Створено: {getDate(survey.createdAt)}
                    </Typography>
                    <Typography variant="h6" color="grey" component="div" style={{marginBottom:
5}}>
                        Починається: {getDate(survey.startAt)}
                    </Typography>
```

```
                    <Typography variant="h6" color="grey" component="div" style={{marginBottom:
5}}>
                        Закінчується: {getDate(survey.endAt)}
                    </Typography>
                    {survey.updatedAt && (
                        <Typography variant="h6" color="grey" component="div"
style={{marginBottom: 5}}>
                            Редаговано: {getDate(survey.updatedAt)}
                        </Typography>
                    )}
                </CardContent>
            </Card>
            <Grid container spacing={2}>
                <Grid item xs={6}>
                    <Card style={{marginTop: 24}}>
                        <CardContent>
                            <Typography variant="h5" component="div" style={{marginBottom: 5}}>
                                Найпопулярніша відповідь
                            </Typography>
                            <Divider />
                            <Typography variant="h4" component="div" style={{marginTop: 10}}>
                                {responses.length > 0 ? getTextResponse(getMostPopular(scores))
: 'Нема відповідей'}
                            </Typography>
                        </CardContent>
                    </Card>
                </Grid>
                <Grid item xs={6}>
                    <Card style={{marginTop: 24}}>
                        <CardContent>
                            <Typography variant="h5" component="div" style={{marginBottom: 5}}>
                                Середня відповідь
                            </Typography>
                            <Divider />
                            <Typography variant="h4" component="div" style={{marginTop: 10}}>
                                {responses.length > 0 ? getTextResponse(getAverage(scores)) :
'Нема відповідей'}
                            </Typography>
                        </CardContent>
                    </Card>
                </Grid>

            </Grid>
            <Card style={{marginBottom: 24}}>
                <CardContent>
                    <RadarChart questions={questions} />
                </CardContent>
            </Card>
        </div>
    );
};

export default OverviewTab;
```

QuestionsTab.js

```
import React from 'react';
import Question from "../Question";
import {Card, CardContent, Divider, Typography} from "@mui/material";

const QuestionsTab = ({questions, comments}) => {

    return (
        <div>
            {questions && questions.length > 0 && (
                <div>
                {questions.map(question => (
                    <Question question={question} />
                ))}
                {comments.map(comment => (
                    <Card style={{marginTop: 24,marginBottom: 50}}>
                        <CardContent>
                            <Typography variant="h5" component="div" style={{marginBottom:
10}}>
                                Коментар
                            </Typography>
                            <Divider />
                            <Typography variant="p" component="p" style={{padding: 10}}>
                                {comment.text}
                            </Typography>
                        </CardContent>
                    </Card>
                ))}
                </div>
            )}
            {!(questions && questions.length) && (
                <Typography variant="h5" component="div" style={{marginBottom: 10}}>
                    Нема питань
                </Typography>
            )}

        </div>
    );
};

export default QuestionsTab;
```

Question.js

```
import React, {useState} from 'react';
import {Button, ButtonGroup, Card, CardContent, Divider, Typography} from "@mui/material";
import ChartView from "./data-views/ChartView";
import {countElement, getPercent} from "../utils/math";
import TableView from "./data-views/TableView";

const Question = ({question}) => {
    const [resultView, setResultView] = useState('default')
```

```
    return (
        <Card style={{marginTop: 24}}>
            <CardContent>
                <Typography variant="h5" component="div" style={{marginBottom: 10}}>
                    {question.text}
                </Typography>
                <Typography variant="p" color="grey" component="div" style={{marginBottom:
10}}>
                    {question.responses.length} Відповідей
                </Typography>
                <Divider />
                {(question.responses && question.responses.length > 0) && (
                <div>
                    <ButtonGroup
                        size="small"
                        variant="outlined"
                        aria-label="outlined primary button group"
                        style={{marginTop: 20}}
                    >
                        <Button
                            onClick={() => setResultView('bar')}
                            variant={(resultView === 'default' || resultView === 'bar') ?
'contained' : 'outlined'}
                        >Bar</Button>
                        <Button
                            onClick={() => setResultView('pie')}
                            variant={resultView === 'pie' ? 'contained' : 'outlined'}
                        >Pie</Button>
                        <Button
                            onClick={() => setResultView('polar')}
                            variant={resultView === 'polar' ? 'contained' : 'outlined'}
                        >Polar</Button>
                        <Button
                            onClick={() => setResultView('table')}
                            variant={resultView === 'table' ? 'contained' : 'outlined'}
                        >Table</Button>
                        <Button
                            onClick={() => setResultView('percent')}
                            variant={resultView === 'percent' ? 'contained' : 'outlined'}
                        >%</Button>
                    </ButtonGroup>
                    {(resultView === 'bar' || resultView === 'default') && (
                        <ChartView responses={question.responses} type='bar' />
                    )}
                    {resultView === 'pie' && (
                        <ChartView responses={question.responses} type={resultView} />
                    )}
                    {resultView === 'polar' && (
                        <ChartView responses={question.responses} type={resultView} />
                    )}
                    {resultView === 'table' && (
                        <TableView responses={question.responses} func={countElement}
columnName="Кількість"/>
```

```
                    )}
                    {resultView === 'percent' && (
                        <TableView responses={question.responses} func={getPercent}
columnName="Відсоток" />
                    )}
                    </div>
                )}
                {!(question.responses && question.responses.length > 0) && (
                    <Typography component="p" color="grey" style={{paddingInline: 20,
paddingBlock: 30 }}>
                        Немає відповідей
                    </Typography>
                )}

            </CardContent>

        </Card>
    );
};
```

SurveyList.js

```
import React, {useEffect, useState} from 'react';
import styles from './styles.module.sass';
import {useDispatch, useSelector} from "react-redux";
import {fetchSurveys} from "../../services/survey-service";
import {Container, Skeleton, ToggleButton, ToggleButtonGroup, Typography} from "@mui/material";
import FormatListBulletedIcon from '@mui/icons-material/FormatListBulleted';
import ReorderIcon from '@mui/icons-material/Reorder';
import DefaultListView from "./list-views/DefaultListView";
import SortedListView from "./list-views/SortedListView";

function SurveysList() {
  const dispatch = useDispatch();
  const { surveys } = useSelector((state) => state.surveysReducer);

    const [listView, setListView] = useState('default')

    const handleListView = (event, newView) => {
        if (newView)
            setListView(newView)
    }

  useEffect(() => {
    dispatch(fetchSurveys());
  }, [])

  return (
    <div className={styles.container}>
        <Container>
            <ToggleButtonGroup
                value={listView}
                exclusive
                onChange={handleListView}
```

```
                size="small"
                aria-label="list view"
            >
                <ToggleButton value="default" aria-label="default list">
                    <ReorderIcon />
                </ToggleButton>
                <ToggleButton value="sorted" aria-label="sorted-list">
                    <FormatListBulletedIcon />
                </ToggleButton>
            </ToggleButtonGroup>
        </Container>
        {surveys && surveys.length > 0 && (
            <div>
                {listView === 'default' && <DefaultListView surveys={surveys} />}
                {listView === 'sorted' && <SortedListView surveys={surveys} />}
            </div>
        )}
        {!(surveys && surveys.length > 0) && (
            <Typography variant="h6" component="h1" style={{paddingInline: 0, fontWeight:
700}}>
                Нема опитувань
            </Typography>
        )}


    </div>
  );
}

export default SurveysList;
```

SortedListView.js

```
import React from 'react';
import { useNavigate} from "react-router-dom";
import { isPast} from "../../../utils/date";
import {
    Container,
    Card,
    CardContent,
    Typography,
} from "@mui/material";
import MainList from "../../../components/basic/MainList";

const SortedListView = ({surveys}) => {

    const finished = surveys.filter(survey => isPast(survey.endAt))
    const planned = surveys.filter(survey => !isPast(survey.startAt) && !isPast(survey.endAt))
    const current = surveys.filter(survey => !isPast(survey.endAt) && isPast(survey.startAt))

    const navigate = useNavigate()

    return (
```

```
        <Container>
            <h1>Опитування</h1>
            <br/>
            <Card>
                <CardContent>
                    <Typography variant="h6" component="h1" style={{paddingInline: 0,
fontWeight: 700}}>
                        Заплановані
                    </Typography>
                    <hr/>
                    <MainList surveys={planned}/>
                </CardContent>
            </Card>
            <Card style={{marginTop: 24}}>
                <CardContent>
                    <Typography variant="h6" component="h1" style={{paddingInline: 0,
fontWeight: 700}}>
                        В процессі
                    </Typography>
                    <hr/>
                    <MainList surveys={current}/>
                </CardContent>
            </Card>
            <Card style={{marginTop: 24}}>
                <CardContent>
                    <Typography variant="h6" component="h1" style={{paddingInline: 0,
fontWeight: 700}}>
                        Завершені
                    </Typography>
                    <hr/>
                    <MainList surveys={finished}/>
                </CardContent>
            </Card>
        </Container>
    );
};

export default SortedListView;
```

DefaultListView.js

```
import React from 'react';
import {
    Card,
    Container,
    Typography,
    CardContent
} from "@mui/material";
import MainList from "../../../components/basic/MainList";

const DefaultListView = ({surveys}) => {

    return (
        <Container>
```

```
            <Card style={{marginTop: 24}}>
                <CardContent>
                    <Typography variant="h6" component="h1" style={{paddingInline: 0,
fontWeight: 700}}>
                        За датою створення
                    </Typography>
                    <hr/>
                    <MainList surveys={surveys} />
                </CardContent>
            </Card>
        </Container>
    );
};

export default DefaultListView
```