

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА СИСТЕМИ ОЦІНЮВАННЯ ПРОФЕСІЙНИХ ТА
КОМУНІКАТИВНИХ НАВИЧОК ПРАЦІВНИКІВ
DEVELOPMENT OF A SYSTEM FOR ASSESSING THE PROFESSIONAL
AND COMMUNICATION SKILLS OF EMPLOYEES

Виконав(ла): студент(ка) 4 курсу, групи КНТ-
121сп

Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ПАЛІЄНКО Д.С.

(ПРИЗВИЩЕ та ініціали)

Керівник КАПЛІЄНКО Т.І.

(ПРИЗВИЩЕ та ініціали)

Рецензент ШИТІКОВА О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ПАЛІЄНКО Данила Сергійовича
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка системи оцінювання професійних та комунікативних навичок працівників.
Development of a System for Assessing the Professional and Communication Skills of Employees

керівник проєкту (роботи) к.т.н., доцент, КАПІЛІЄНКО Тетяна Ігорівна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 24 ” квітня 2024 року № 168

2. Строк подання студентом проєкту (роботи) 10 червня 2024 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Характеристика об'єкта проєктування.
2. Вибір і обґрунтування структури системи. 3. Основні рішення щодо реалізації компонентів системи. 4. Керівництво програміста. 5. Керівництво системного адміністратора. 6. Керівництво користувача.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-6 Основна частина	КАПЛІЄНКО Т.І., доцент		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання “ 15 ” квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

_____ Данило ПАЛІЄНКО
(підпис) (Імя ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Тетяна Каплієнко
(підпис) (Імя ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 110 с., 6 табл., 44 рис., 2 дод., 17 джерел.

ВЕБЗАСТОСУНОК, САЙТ, НАВЧАННЯ, ТЕСТУВАННЯ, ОЦІНКА СПІВРОБІТНИКІВ.

Об'єкт дослідження – програмні засоби для оцінки співробітників.

Предмет дослідження – вебзастосунок для оцінки комунікативних та професійних навичок співробітників.

Мета роботи – створення вебзастосунку для автоматизації та прискорення навчання співробітників та оцінки їх комунікативних та професійних навичок.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, мови програмування JavaScript, TypeScript, персональний комп'ютер.

Результати. Створено вебзастосунок, у якому співробітники можуть навчатися, проходити тестування та оцінювати себе і інших співробітників.

Висновки. Розроблено програмну систему для навчання, тестування та оцінки комунікативних та професійних навичок.

Галузь використання – навчання та тестування співробітників в корпоративному середовищі.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 110 pages, 6 tables, 44 figures, 2 appendixes, 17 sources.

WEB APPLICATION, WEBSITE, TRAINING, TESTING, EMPLOYEE EVALUATION.

Object of study is a software tools for employee evaluation.

Subject of study are web applications for evaluating employees' communication and professional skills.

The purpose of the work is to create a web application for automating and accelerating employee training and the evaluation of their communication and professional skills.

Materials, methods, and tools: structural and object-oriented programming, JavaScript, TypeScript programming languages, personal computer.

Results. a web application was created where employees can train, undergo testing, and evaluate other employees.

Conclusions a software system for training, testing, and evaluating communication and professional skills has been developed.

The field of use is employee training and testing.

ЗМІСТ

С.

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК.....	8
ВСТУП.....	9
1 ХАРАКТЕРИСТИКА ОБ’ЄКТА ПРОЄКТУВАННЯ	10
1.1 Поняття навчального центра	10
1.2 Огляд аналогів системи оцінювання	11
1.3 Основні способи використання системи оцінювання в корпоративному середовищі.....	13
1.4 Основні функціональні можливості системи оцінювання	15
1.5 Висновки за розділом	16
2 ВИБІР І ОБҐРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ.....	17
2.1 Вибір середовища програмування для серверної частини сайту.....	17
2.2 Вибір середовища програмування для клієнтської частини сайту.....	20
2.3 Вибір середовища програмування для клієнтської частини сайту.....	22
2.4 Бази даних	25
2.5 Функціональна схема роботи	28
2.6 Висновки за розділом	29
3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ.....	31
3.1 Основні відомості про розробку системи управління навчальним центром	31
3.2 Принцип роботи лекційної та тестувальної частини застосунка.....	33
3.3 Принцип роботи частини оцінювання методом «360 градусів»	34
3.4 Алгоритм функціонування адміністративної частини.....	36
3.4 Висновки за розділом.....	37

4 КЕРІВНИЦТВО ПРОГРАМІСТА	39
4.1 Призначення й умови використання навчального центра	39
4.2 Характеристика продукту	39
4.3 Структура бази даних	40
4.4 Структура застосунку	43
4.5 Висновки за розділом	44
5 КЕРІВНИЦТВО СИСТЕМНОГО АДМІНІСТРАТОРА	45
5.1 Розгортання навчального центра	45
5.2 Розгортання навчального центра	45
5.3 Розгортання бази даних	45
5.4 Розгортання серверної частини	46
5.5 Розгортання клієнтської частини	46
5.6 Адміністративна панель навчального центра	47
5.7 Висновки за розділом	51
6 КЕРІВНИЦТВО КОРИСТУВАЧА	52
6.1 Призначення вебзастосунку	52
6.2 Авторизація, процес навчання та проведення тестувань	52
6.3 Висновки за розділом	58
ВИСНОВКИ	59
ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	60
ДОДАТОК А ТЕКСТ ПРОГРАМИ	62
ДОДАТОК Б СЛАЙДИ ПРЕЗЕНТАЦІЇ	100

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

Онбордінг (англ. onboarding) - це процес введення в посаду та адаптації нових співробітників, який допомагає їм швидко ознайомитися з компанією, її культурою, процесами та почати ефективно виконувати свої робочі обов'язки.

СКБД (Система керування базами даних) - це спеціальне програмне забезпечення, яке призначене для створення, управління та забезпечення ефективного доступу до баз даних.

ВСТУП

У сучасному цифровізованому бізнес-середовищі, автоматизація стає вирішальним елементом для підвищення продуктивності та ефективності організацій. Завдяки впровадженню новітніх електронно-обчислювальних технологій, компанії можуть значно оптимізувати свої процеси, підвищити прибутковість і продуктивність.

В освітній сфері, особливо в контексті корпоративного навчання, автоматизація відіграє ключову роль, дозволяючи навчальним центрам ефективно реагувати на потреби динамічного ринку. Була розроблена автоматизована система управління для навчального центра, яка зосереджена на компаніях, метою якої є навчання співробітників та аналіз їхньої взаємодії за допомогою оцінки «360 градусів» [1]. Метод «360 градусів» - це підхід до оцінювання, який передбачає отримання зворотного зв'язку про працівника від його керівника, колег, підлеглих та самооцінки, що дозволяє отримати всебічне уявлення про компетенції та ефективність співробітника.

Ця система призначена для вдосконалення навчального процесу, поліпшення управління курсами, забезпечення ефективної взаємодії між співробітниками, а також надає комплексний аналіз навичок і взаємодії персоналу в межах компанії. Автоматизація адміністративних завдань та забезпечення доступності навчальних матеріалів для співробітників незалежно від їхнього місцезнаходження робить цю систему важливим інструментом для розвитку корпоративних команд.

Система буде будуватись на перевірених технологіях, що гарантує її надійність та ефективність. Це дозволить створити масштабоване рішення, готове до адаптації до мінливих потреб компанії та її співробітників, забезпечуючи ним потужний ресурс для неперервного професійного розвитку.

1 ХАРАКТЕРИСТИКА ОБ'ЄКТА ПРОЄКТУВАННЯ

1.1 Поняття навчального центра

Навчальний центр – це вебзастосунок, для автоматизації процесів навчання та розвитку персоналу в корпоративному середовищі. Це комплексна система, яка дозволяє проводити оцінювання професійних та комунікативних навичок працівників, призначати навчальні матеріали, відстежувати прогрес та аналізувати результати.

Основні функції та особливості навчального центра:

- централізація навчального процесу - вебзастосунок слугує єдиною платформою для управління всіма аспектами корпоративного навчання. Він дозволяє призначати курси, тести та інші навчальні активності, забезпечуючи єдиний простір для взаємодії між працівниками, менеджерами та HR-фахівцями;
- оцінювання навичок - "навчальний центр" включає в себе функціонал для проведення різноманітних оцінювань, таких як тести, опитування та метод 360 градусів. Це дозволяє визначити поточний рівень компетенцій працівників, виявити сильні сторони та зони для розвитку;
- призначення навчальних матеріалів - система дозволяє призначати співробітникам відповідні навчальні матеріали, такі як електронні курси, вебінари, відео та документи. Працівники можуть отримувати доступ до цих матеріалів у будь-якому місці та в будь-який час, забезпечуючи гнучкість навчального процесу;
- відстеження прогресу та результатів - "навчальний центр" надає інструменти для відстеження прогресу співробітників у проходженні призначених навчальних активностей. Менеджери та HR-фахівці можуть переглядати статистику та звіти, що відображають рівень залученості, результати оцінювань та завершення навчальних програм;
- аналітика та звітність - вебзастосунок генерує різноманітні аналітичні звіти, що дозволяють оцінити ефективність навчальних ініціатив,

виявити тренди та приймати обґрунтовані рішення щодо розвитку персоналу. Це допомагає оптимізувати інвестиції в навчання та підвищити віддачу від навчальних програм.

Таким чином, "Навчальний центр" - це комплексний вебзастосунок, який автоматизує та централізує процеси корпоративного навчання та розвитку. Він дозволяє ефективно оцінювати навички працівників, призначати релевантні навчальні матеріали, відстежувати прогрес та аналізувати результати. Використання такої системи допомагає компаніям розвивати людський капітал, підвищувати продуктивність та досягати стратегічних цілей організації.

1.2 Огляд аналогів системи оцінювання

Для кращого розуміння переваг та недоліків розробленої системи оцінювання професійних та комунікативних навичок працівників, доцільно проаналізувати існуючі аналоги на ринку. У цьому огляді будуть розглянуті три популярні системи: Moodle [2], TalentLMS [3] та Qualtrics 360 [4].

Moodle (рис. 1.1) – це безкоштовна відкрита платформа для створення та управління онлайн-курсами, яка надає широкі можливості для кастомізації та розширення функціоналу за допомогою розширень. Вона підтримує різні типи контенту та активностей, як-от тести, завдання та форуми. Однак Moodle вимагає технічних знань для налаштування та адміністрування системи, має обмежені вбудовані можливості для оцінювання та зворотного зв'язку, а при значному збільшенні кількості користувачів та курсів можуть виникати проблеми з масштабованістю. Інтерфейс Moodle має класичний мінімалістичний дизайн з можливістю зміни теми оформлення та розташування елементів.

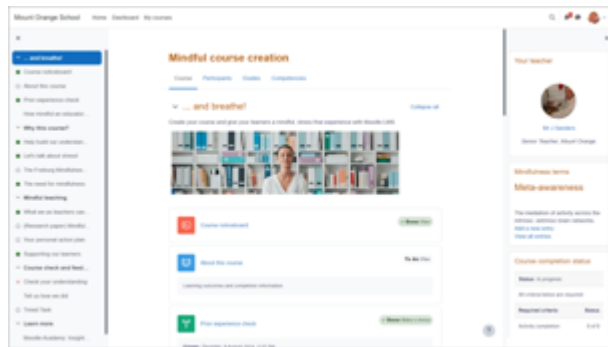


Рисунок 1.1 – Зовнішній вигляд Moodle [2]

TalentLMS (рис. 1.2) – це хмарна платформа для створення та проведення онлайн-навчання з інтуїтивно зрозумілим інтерфейсом для адміністраторів та користувачів. Вона містить вбудовані інструменти для гейміфікації навчального процесу, такі як бейджі та сертифікати. Проте у безкоштовному тарифі TalentLMS функціонал обмежений, платформа не має вбудованих інструментів для оцінювання за методом 360 градусів, а також надає менші можливості для кастомізації порівняно з Moodle. Інтерфейс TalentLMS має сучасний привабливий дизайн з адаптивною версткою, що забезпечує зручне використання на різних пристроях.



Рисунок 1.2 – Зовнішній вигляд TalentLMS [3]

Qualtrics 360 (рис. 1.3) – спеціалізована платформа для проведення оцінювання за методом 360 градусів, яка надає гнучкі можливості для створення та налаштування опитувань і звітів. Вона інтегрується з популярними системами управління персоналом (HRIS). Однак ліцензії Qualtrics 360 мають високу вартість, особливо для великих організацій.

Платформа не має вбудованих інструментів для навчання та розвитку персоналу, а для налаштування й аналізу результатів оцінювання потрібні кваліфіковані фахівці. Інтерфейс Qualtrics 360 має професійний лаконічний дизайн та надає шаблони опитувань і звітів для швидкого старту роботи з платформою.

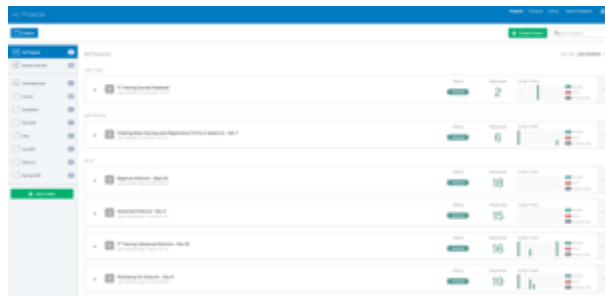


Рисунок 1.3 – Зовнішній вигляд Qualtrics [4]

Проаналізувавши переваги, недоліки й особливості інтерфейсу, можна зробити висновок, що розроблювана система оцінювання має потенціал для успішної конкуренції на ринку за умови вдалого поєднання функцій навчання та оцінювання за методом 360 градусів.

1.3 Основні способи використання системи оцінювання в корпоративному середовищі

Розроблена система оцінювання професійних та комунікативних навичок працівників може бути використана в корпоративному середовищі наступними способами [5]:

- регулярна оцінка персоналу - компанії можуть використовувати систему для проведення регулярних оцінювань співробітників, наприклад, щоквартально або щорічно. Це дозволяє відстежувати прогрес, визначати сильні сторони та зони для розвитку, а також приймати обґрунтовані рішення щодо кар'єрного зростання та розвитку персоналу;

- онбордінг та адаптація - система може бути інтегрована в процес онбордінгу нових співробітників. Оцінка навичок на початку роботи допомагає визначити початковий рівень компетенцій та створити індивідуальний план розвитку для кожного нового працівника;

- формування команд - результати оцінювань можуть бути використані для формування ефективних команд всередині організації. Розуміння сильних сторін та областей для розвитку кожного члена команди дозволяє створювати збалансовані та продуктивні команди;

- планування навчання та розвитку - на основі результатів оцінювань, компанії можуть розробляти цільові навчальні програми та ініціативи розвитку, що відповідають потребам як окремих співробітників, так і організації в цілому. Це дозволяє оптимізувати інвестиції в навчання та розвиток персоналу;

- прийняття кадрових рішень - результати оцінювань можуть бути враховані при прийнятті різних кадрових рішень, таких як підвищення, переведення на іншу посаду, зміна компенсації тощо. Це забезпечує об'єктивність та обґрунтованість таких рішень;

- покращення комунікації та взаємодії - використання системи оцінки 360 градусів сприяє відкритому зворотному зв'язку та покращує комунікацію між співробітниками та керівництвом. Це допомагає створити культуру взаємної підтримки та розвитку в організації;

Таким чином, система оцінювання професійних та комунікативних навичок може бути ефективно інтегрована в різні аспекти управління персоналом в корпоративному середовищі, сприяючи розвитку співробітників, підвищенню продуктивності та досягненню стратегічних цілей організації.

1.4 Основні функціональні можливості системи оцінювання

Система оцінювання професійних та комунікативних навичок працівників повинна мати ключові функціональні можливості, що включають управління користувачами, управління навчальними матеріалами, оцінювання навичок, а також аналітику та звітність.

Управління користувачами передбачатиме реєстрацію нових користувачів з різними ролями, такими як адміністратор та працівник, редагування та оновлення профілів користувачів, а також призначення користувачів до відповідних команд або відділів.

Управління навчальними матеріалами включатиме додавання, редагування та видалення навчальних матеріалів, таких як курси, модулі та лекції, призначення навчальних матеріалів відповідним користувачам або групам, а також відстеження прогресу користувачів у проходженні призначених матеріалів.

Оцінювання навичок передбачатиме створення та управління різними типами оцінювань, такими як тести, опитування та метод 360 градусів, призначення оцінювань відповідним користувачам або групам, автоматичне оцінювання результатів тестів та генерацію звітів, а також можливість проведення самооцінки, оцінки керівником, колегами та підлеглими за методом 360 градусів.

Аналітика та звітність включатимуть генерацію звітів щодо результатів оцінювань та прогресу користувачів, інтеграцію з Power BI для створення інтерактивних та візуально привабливих звітів, а також можливість експорту звітів у різних форматах, таких як PDF, Excel та CSV.

Завдяки цим функціональним можливостям система оцінювання забезпечуватиме повний цикл управління процесом розвитку та оцінки навичок працівників. Система надаватиме гнучкі інструменти для управління користувачами, навчальними матеріалами та проведення різних типів оцінювань. Це дозволить організаціям ефективно керувати процесом розвитку

персоналу та отримувати цінну інформацію про рівень компетенцій своїх співробітників. Можливість генерації звітів та візуалізації даних допомагатиме керівництву аналізувати результати оцінювань та приймати обґрунтовані рішення щодо подальшого розвитку та навчання працівників. Завдяки своїй функціональності та можливостям система оцінювання стане ефективним інструментом для управління людськими ресурсами та розвитку персоналу в різних компаніях та організаціях.

1.5 Висновки за розділом

У даному розділі було проведено ґрунтовне дослідження об'єкта проєктування - системи оцінювання професійних та комунікативних навичок працівників. Були проаналізовані конкуренти та визначені ключові поняття і терміни, що характеризують основні процеси та принципи функціонування такої системи в контексті навчального центру. Також було розглянуто розглянули поняття "Навчальний центр" як вебзастосунок, що автоматизує процеси навчання та розвитку персоналу в корпоративному середовищі, а також детально проаналізували основні функціональні можливості розробленої системи оцінювання.

На основі проведеного дослідження, було визначено, що розробка системи оцінювання професійних та комунікативних навичок працівників є актуальною та необхідною в сучасних умовах. Така система надає значні переваги в порівнянні з традиційними методами оцінювання та розвитку персоналу, забезпечуючи автоматизацію, ефективність та можливість масштабування процесів навчання та оцінювання. Реалізація такої системи дозволить компаніям підвищити продуктивність, конкурентоспроможність та забезпечити безперервний розвиток своїх співробітників.

2 ВИБІР І ОБҐРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ

2.1 Вибір середовища програмування для серверної частини сайту

Для розробки серверної частини системи оцінювання професійних та комунікативних навичок працівників було обрано використовувати мову програмування TypeScript [6] разом із фреймворком NestJS та бібліотекою TypeORM.

TypeScript є розширенням JavaScript, що додає статичну типізацію, класи, інтерфейси та інші функції, підвищуючи надійність, зручність та масштабованість коду. На відміну від JavaScript, який є динамічно типізованою мовою, TypeScript забезпечує перевірку типів на етапі компіляції, допомагаючи виявляти та уникати помилок ще до запуску програми. Код TypeScript компілюється в звичайний JavaScript, що дозволяє виконувати його на будь-якій платформі, яка підтримує JavaScript, включаючи Node.js на серверній стороні.

NestJS [7] - це прогресивний фреймворк для створення ефективних, масштабованих та тестованих серверних застосунків на основі Node.js. Він забезпечує структурований підхід до розробки з використанням шаблонів проєктування та принципів модульності. NestJS використовує концепції залежностей та декораторів, що робить код більш читабельним та зрозумілим. Він також має вбудовану підтримку TypeScript, що дозволяє писати типізований код та користуватися перевагами інтегрованого середовища розробки (IDE).

TypeORM [8] - це бібліотека Object-Relational Mapping (ORM) для TypeScript та JavaScript, що дозволяє працювати з базами даних, використовуючи об'єктно-орієнтований підхід. Вона підтримує різні види баз даних, включаючи PostgreSQL, яка була обрана для цього проєкту. TypeORM дозволяє визначати сутності бази даних у вигляді класів TypeScript та надає зручний API для виконання запитів, обробки відношень між таблицями та керування міграціями бази даних. Використання TypeORM полегшує

взаємодію з базою даних та забезпечує абстракцію над SQL, дозволяючи писати більш чистий та підтримуваний код.

Порівняльні таблиці 2.1, 2.2 та 2.3 демонструють, що комбінація TypeScript, NestJS та TypeORM пропонує ряд істотних переваг перед альтернативними мовами, фреймворками та ORM відповідно.

Таблиця 2.1 – Порівняння мов програмування для серверної розробки

Критерій	TypeScript	Python [9]	Java [10]
Тип	Об'єктно-орієнтована	Об'єктно-орієнтована, процедурна, функціональна	Об'єктно-орієнтована
Типізація	Статична, необов'язкова	Динамічна	Статична
Область застосування	Веб, Node.js, великі застосунки	Веб, наука, скриптинг	Корпоративні застосунки, мобільні застосунки, веб
Продуктивність	Висока	Середня	Висока
Розширення та бібліотеки	Великий вибір розширень	Великий вибір розширень	Обмежений вибір розширень

Таблиця 2.2 – Порівняння фреймворків для Node.js

Критерій	NestJS	Express	Koa	Нарі
Архітектурний підхід	Модульний	Мінімалістичний	Мінімалістичний	Конфігураційний
Типова система	TypeScript	JavaScript	JavaScript	TypeScript, JavaScript

Продовження таблиці 2.2

Критерій	NestJS	Express	Koa	Hapi
Шаблони проектування	Dependency Injection, модулі, декоратори	Відсутні	Відсутні	Плагіни, хуки

Таблиця 2.3 – Порівняння ORM

Критерій	TypeORM	Sequelize	Mongoose	Objection.js
Мова програмування	TypeScript, JavaScript	JavaScript	JavaScript	JavaScript
Типи баз даних	PostgreSQL, MySQL, SQLite, Oracle та ін.	PostgreSQL, MySQL, SQLite, Oracle та ін.	MongoDB	PostgreSQL, MySQL, SQLite, Oracle та ін.

По-перше, TypeScript як мова програмування забезпечує статичну типізацію, що допомагає виявляти та попереджати помилки на етапі компіляції, покращуючи надійність та підтримуваність коду. На відміну від динамічно типізованих мов, таких як JavaScript чи Python, TypeScript дозволяє писати більш структурований та масштабований код для великих застосунків, полегшуючи рефакторинг та розширення функціоналу.

По-друге, NestJS як фреймворк для Node.js пропонує модульну архітектуру, засновану на концепціях Angular, що сприяє кращій організації коду та полегшує тестування і розширення застосунку. Використання шаблонів проектування, таких як Dependency Injection, модулі та декоратори, робить код більш читабельним та зрозумілим. На відміну від мінімалістичних фреймворків, як Express чи Koa, NestJS забезпечує структурований підхід до розробки, що є особливо корисним для великих проєктів.

Нарешті, TypeORM спрощує взаємодію з базою даних, надаючи об'єктно-орієнтований підхід до роботи з даними. Порівняно з альтернативами, такими як Sequelize чи Mongoose, TypeORM пропонує кращу

підтримку TypeScript та широкий вибір підтримуваних баз даних, включаючи PostgreSQL. Використання TypeORM дозволяє писати більш чистий та підтримуваний код, абстрагуючись від SQL та концентруючись на об'єктній моделі даних.

Отже, поєднання TypeScript, NestJS та TypeORM забезпечує надійну, структуровану та масштабовану основу для розробки серверної частини системи оцінювання професійних та комунікативних навичок працівників. Ця комбінація технологій полегшує розробку, підтримку та розширення застосунка, а також сприяє кращій продуктивності та якості коду.

2.2 Вибір середовища програмування для клієнтської частини сайту

Після ретельного аналізу різноманітних технологій та підходів для створення сучасних вебзастосунків, було обрано бібліотеку React у поєднанні з бібліотекою компонентів користувацького інтерфейсу Material-UI для розробки клієнтської частини.

React – це JavaScript бібліотека, яка стала індустріальним стандартом для побудови користувацьких інтерфейсів. Вона базується на концепції компонентів, які можна вважати невеликими автономними блоками коду, відповідальними за певну частину інтерфейсу та його поведінки. Компоненти React легко повторно використовувати та композитувати для створення складніших інтерфейсів, підвищуючи модульність та зменшуючи дублювання коду.

Прикладом компонента у системі оцінювання може бути компонент "Курс", який відображає інформацію про навчальний курс, такий як назва, опис, прогрес проходження та кнопки для початку чи продовження курсу. Цей компонент можна багаторазово використовувати для відображення різних курсів у різних частинах застосунку.

React використовує декларативний підхід до створення інтерфейсів, що робить код більш передбачуваним та простим для розуміння. Замість безпосередньої маніпуляції DOM (Document Object Model), React дозволяє визначати, як інтерфейс повинен виглядати на основі поточного стану компонентів. При зміні стану, React ефективно оновлює та відображає лише змінені компоненти, використовуючи віртуальний DOM, що значно підвищує продуктивність у порівнянні з традиційними підходами.

Наприклад, під час проходження тесту в системі оцінювання, React може ефективно оновлювати лише компоненти, пов'язані з поточним питанням та прогресом тестування, замість перезавантаження всієї сторінки.

React також забезпечує потужну систему управління станом, що полегшує роботу з динамічними даними та взаємодією між компонентами. Бібліотеки, такі як Redux або Context API, дозволяють централізовано керувати станом застосунку та забезпечують передбачувану поведінку під час змін даних.

Для створення привабливого та зручного користувацького інтерфейсу система оцінювання використовує Material-UI [11] – комплексну бібліотеку готових компонентів інтерфейсу, створених відповідно до принципів матеріального дизайну від Google. Material-UI пропонує величезний набір високоякісних та переоптимізованих компонентів, таких як кнопки, меню, діалогові вікна, таблиці, форми введення даних та багато іншого

Перевагою Material-UI є те, що всі компоненти розроблені з урахуванням принципів доступності та оптимізовані для різних пристроїв та платформ, включаючи мобільні та планшети. Це гарантує, що користувацький інтерфейс системи оцінювання буде зручним та зрозумілим для користувачів, незалежно від того, на якому пристрої вони працюють.

Крім того, Material-UI надає гнучкі можливості налаштування, дозволяючи адаптувати стандартні компоненти до унікальних вимог дизайну системи оцінювання. Наприклад, можна легко змінити кольорову схему, типографіку або додати спеціальні ефекти для досягнення бажаного відчуття

користувацького інтерфейсу, зберігаючи при цьому всі переваги готових компонентів.

Завдяки поєднанню потужних можливостей React та багатого функціоналу Material-UI, система оцінювання професійних та комунікативних навичок працівників матиме сучасний, інтерактивний, доступний та високопродуктивний користувацький інтерфейс, який відповідатиме найвищим стандартам якості та зручності використання. Модульність React та готові компоненти Material-UI скоротять час розробки та полегшать підтримку системи в майбутньому, забезпечуючи ефективний процес оцінювання, навчання та розвитку персоналу.

2.3 Вибір середовища програмування для клієнтської частини сайту

Для коректного функціонування та відображення системи оцінювання професійних та комунікативних навичок працівників необхідно використовувати вебпереглядач або браузер. На сьогоднішній день кожен комп'ютер чи інший пристрій з можливістю виходу в інтернет оснащений вебпереглядачем, який є невід'ємним інструментом для перегляду вебсторінок та взаємодії з вебзастосунками.

На ринку доступний широкий вибір серед вебпереглядачів, проте найпопулярнішими є: Google Chrome, Mozilla Firefox, Microsoft Edge, Opera та Safari. Розглянемо їх детальніше:

Google Chrome [12] – безкоштовний вебпереглядач від компанії Google, створений на основі відкритого коду Chromium. Його архітектура була розроблена з чистого аркуша, визнаючи, що більшість сучасних вебсайтів є вебзастосунками. Заявлені переваги Chrome: підвищена стабільність, швидкість, безпека та ефективний інтерфейс користувача.

Mozilla Firefox [13] – вільний безкоштовний вебпереглядач з відкритим кодом на основі ядра Gecko. Після випуску в 2004 році отримав схвальні

відгуки та широке визнання, ставши однією з найпопулярніших програм з відкритим кодом.

Microsoft Edge [14] – вебпереглядач від компанії Microsoft, який прийшов на заміну застарілому Internet Explorer. Початкова версія, випущена в 2015 році, була заснована на власному рушії EdgeHTML, проте у 2020 році Microsoft перейшла на використання рушія Chromium, спільного з Google Chrome. Це забезпечило кращу сумісність з вебстандартами та підвищену продуктивність.

Opera [15] – вебпереглядач від норвезької компанії Opera Software. Спочатку випущений у 1994 році, до 2013 року використовував власний браузерний рушій, а після переходу на відкритий рушій Blink додаткові функції були виокремлені з браузера.

Safari [16] – це вебпереглядач, розроблений компанією Apple для власних операційних систем macOS та iOS. Він використовує рушій WebKit, який також є основою для інших браузерів, таких як Chrome та Opera.

Нижче наведені таблиці з порівнянням популярних вебпереглядачів за різними критеріями (табл. 2.4 – 2.5).

Таблиця 2.4 – Порівняння вебпереглядачів

Критерій	Google Chrome	Mozilla Firefox	Microsoft Edge	Opera	Safari
Тип ліцензії	Вільне та відкрите ПЗ	Вільне та відкрите ПЗ	Проприєтарне ПЗ	Проприєтарне ПЗ	Проприєтарне ПЗ
Рушій	Blink (форк WebKit)	Gecko	Chromium	Blink	WebKit
Підтримка стандартів	Висока	Висока	Висока	Висока	Висока

Продовження таблиці 2.4

Критерій	Google Chrome	Mozilla Firefox	Microsoft Edge	Opera	Safari
Продуктивність	Висока	Висока	Висока	Висока	Висока
Розширюваність	Великий вибір розширень	Великий вибір розширень	Обмежений вибір розширень	Обмежений вибір розширень	Обмежений вибір розширень
Кросплатформеність	Windows, macOS, Linux	Windows, macOS, Linux	Windows, macOS, Linux	Windows, macOS, Linux	macOS

Таблиця 2.5 – Порівняння вебпереглядачів за часткою ринку (дані за квітень 2023 року)

Браузер	Частка ринку
Google Chrome	64,5%
Safari	18,9%
Microsoft Edge	4,3%
Mozilla Firefox	3,7%
Safari	2,4%
Інші	6,2%

Беручи до уваги інформацію з таблиць про підтримку вебтехнологій та протоколів різними браузерами, а також їх поширеність та популярність, рекомендовано використовувати Google Chrome для забезпечення чіткого та безпроблемного відображення системи оцінювання, оскільки він вважається найоптимальнішим серед популярних браузерів. Проте, для забезпечення максимальної сумісності та доступності системи для різних користувачів, слід також враховувати інші поширені браузери, такі як Mozilla Firefox, Microsoft Edge та Safari.

2.4 Бази даних

Для реалізації бази даних було обрано потужну об'єктно-реляційну систему керування базами даних (СКБД) з відкритим кодом – PostgreSQL [17]. База даних є структурованим сховищем інформації, здатним вмістити практично будь-які дані - від простих списків покупок до величезних корпоративних даних та аналітики. Для ефективного зберігання, отримання, обробки та аналізу даних у комп'ютерній базі необхідна спеціалізована система керування, якою в даному випадку виступає PostgreSQL. Завдяки можливості ефективно працювати з великими обсягами інформації, сучасні СКБД, такі як PostgreSQL, відіграють ключову роль в обчислювальних процесах і аналітиці даних.

У своїй основі PostgreSQL є реляційною базою даних, в якій дані організовані у вигляді окремих взаємопов'язаних таблиць, що забезпечує високу швидкість та гнучкість. Проте PostgreSQL також підтримує об'єктно-орієнтовану парадигму, дозволяючи створювати складні об'єктні типи даних та користувацькі функції, процедури і правила. Це робить PostgreSQL високопродуктивною та розширюваною системою. Для маніпулювання даними та визначення схеми бази даних використовується стандартна мова структурованих запитів SQL.

PostgreSQL розповсюджується як вільне програмне забезпечення з повністю відкритим вихідним кодом відповідно до ліцензії MIT подібної до BSD. Це означає, що будь-хто може вільно використовувати, вивчати, модифікувати та поширювати PostgreSQL за умови дотримання положень ліцензії. Проєкт підтримується активною спільнотою розробників та ентузіастів.

Ця система керування базами даних високо цінується за свою швидкість, надійність, та багатий функціонал. Розробники PostgreSQL продовжують активно її вдосконалювати, тісно співпрацюючи з користувацькою спільнотою для впровадження найбільш затребуваних

функцій та можливостей. Спочатку PostgreSQL проєктувалася як потужне рішення для роботи з дуже великими базами даних і високим навантаженням, пропонуючи вищу продуктивність порівняно з аналогами того часу. І вже багато років ця СКБД успішно використовується у високонавантажених виробничих середовищах, довівши свою ефективність.

Хоча розвиток PostgreSQL безперервно продовжується, вона вже зараз пропонує надзвичайно широкий спектр корисних можливостей. PostgreSQL відома вбудованою підтримкою потужних функцій безпеки, таких як ролі користувачів, контроль доступу на рівні рядків, шифрування даних, а також механізми аудиту та відновлення цілісності бази даних після збоїв. Програмісти особливо цінують PostgreSQL за повну відповідність стандартам SQL, підтримку паралельних запитів, тригерів, матеріалізованих уявлень, а також здатність писати масштабовані застосунки, що забезпечують атомарність, узгодженість, ізольованість і довговічність транзакцій (ACID).

Завдяки високій доступності, швидкодії, цілісності та надійним механізмам захисту даних PostgreSQL є фантастичним вибором для вебсервісів, хмарних застосунків, мобільних рішень та інших систем, що потребують доступу до баз даних через інтернет. Забезпечуючи безпечний віддалений доступ, масштабованість та непогану продуктивність навіть на пристроях з обмеженими обчислювальними ресурсами, PostgreSQL дозволяє ефективно вирішувати широке коло завдань з управління даними.

PostgreSQL є класичною клієнт-серверною системою з багатопотоковим SQL-сервером, здатним підтримувати численні бази даних одночасно. Є багато різноманітних клієнтських програм, адміністративних інструментів та інтерфейсів програмування (API) як від основних розробників PostgreSQL, так і від сторонніх команд. Ядро PostgreSQL також можна інтегрувати як вбудовану СКБД у спеціалізовані програми для створення компактних і високопродуктивних рішень.

Структурно PostgreSQL побудована на трирівневій архітектурі: бази даних - таблиці - записи. Бази даних і таблиці фізично представлені окремими

файлами на диску. Логічно кожна таблиця є сукупністю записів, а кожен запис містить набір полів різної структури та типів даних. В межах всієї системи PostgreSQL імена баз даних мають бути унікальними, так само як імена таблиць в межах кожної бази і імена полів в межах таблиці. На одному сервері PostgreSQL можна розміщувати численні бази даних, доступ до яких надається за допомогою облікових записів з логінами та паролями.

Безпека в PostgreSQL забезпечується низкою потужних механізмів контролю доступу на основі гнучкої ієрархії привілеїв користувачів. З'єднання з сервером дозволяється лише після успішної автентифікації користувача за допомогою пароля, який зберігається в захищеній формі із застосуванням одностороннього шифрування. Після підключення система безперервно відстежує дії користувача, жорстко обмежуючи їх відповідно до наданих прав доступу на рівні баз даних, таблиць, рядків або окремих операцій. Підтримується також шифрування з'єднань за допомогою протоколу SSL для гарантування конфіденційності даних при мережевій взаємодії.

Одна з визначних переваг PostgreSQL полягає у вбудованій підтримці широкого спектру типів даних. Крім стандартних числових, текстових, бінарних та типів дат/часу, присутня підтримка спеціалізованих типів, як от масиви, XML, JSON, а також можливість створювати повністю користувацькі типи даних із власними методами та правилами індексації. Це робить PostgreSQL неймовірно гнучкою та розширюваною системою, здатною ефективно працювати з будь-якими видами структурованих даних.

Серед ключових переваг PostgreSQL, які роблять її одним з лідерів у своїй категорії, можна відзначити: величезну потужність і практично необмежену масштабованість для роботи з великими обсягами даних; наявність численних функцій безпеки для забезпечення конфіденційності, цілісності, аудиту та відновлення даних; надзвичайно високу надійність і стійкість до різноманітних збоїв та відмов обладнання; повну відповідність стандартам SQL та підтримку сучасної об'єктно-реляційної моделі даних;

вільне поширення з відкритим кодом за ліцензією MIT та наявність активної спільноти розробників і ентузіастів

Незважаючи на кілька відомих обмежень, PostgreSQL вважається однією з найбільш потужних, масштабованих, надійних і функціональних об'єктно-реляційних систем керування базами даних з відкритим кодом. Її продуктивність, відповідність стандартам і розширюваність роблять PostgreSQL чудовим вибором для реалізації баз даних у корпоративних застосунках, вебсервісах та інших проєктах, де потрібна ефективна і безпечна робота з даними.

2.5 Функціональна схема роботи

Розглянемо функціональну схему роботи системи управління персоналом, яка включає в себе можливості оцінювання за методом 360 градусів, проведення навчальних курсів та тестувань.

На початку процесу HR-спеціаліст має зареєструвати запис з ідентифікаційними даними працівника (наприклад, ІПН) в адміністративній панелі системи. Після цього сам працівник зможе зареєструватися та отримати доступ до сервісу.

Коли кілька працівників будуть зареєстровані, HR-спеціаліст матиме можливість створити команду, в якій вони працюватимуть. Для цієї команди можна буде призначати оцінювання за методом 360 градусів або закриті навчальні курси, розроблені спеціально для них.

Сам HR-спеціаліст або окремий співробітник, відповідальний за навчальний контент, зможе розробляти та записувати в систему опитування за методом 360 градусів, навчальні курси та тестування. Ці матеріали будуть прикріплені до відповідних курсів.

Перед тим, як розпочати проходження тестування, користувач (працівник) має переглянути весь навчальний курс. Після цього він зможе натиснути кнопку "Пройти тестування" і розпочати тестування.

Як тільки користувач завершить тестування, HR-спеціаліст отримає повідомлення в своїй поштовій скриньці з інформацією про проходження тесту та його результатами.

Коли вся команда пройде оцінювання за методом 360 градусів, HR отримає сповіщення про його повне завершення з посиланням на підсумкові результати. Ці результати будуть сформовані та візуалізовані за допомогою інструменту Power BI.

2.6 Висновки за розділом

У цьому розділі був зроблений ретельний вибір технологічного стеку для створення системи оцінювання професійних та комунікативних навичок працівників. Обрані інструменти забезпечать розробку високоякісного і функціонального програмного рішення.

Для клієнтської частини буде використано TypeScript - розширення JavaScript зі статичною типізацією та об'єктно-орієнтованими можливостями. Це забезпечить написання надійного коду з кращою підтримкою інструментів розробки. Бібліотека React дозволить створювати інтерактивні користувацькі інтерфейси, а Material-UI надасть набір готових компонентів з можливістю гнучкого налаштування для отримання сучасного та зручного дизайну.

На серверній стороні також буде застосовано TypeScript у поєднанні з фреймворком NestJS. NestJS забезпечить структурований підхід до розробки застосунків з використанням сучасних шаблонів проєктування.

Для зберігання та обробки даних було обрано об'єктно-реляційну систему керування базами даних PostgreSQL. Вона відома своєю потужністю, високою продуктивністю та відповідністю стандартам, що робить її надійним рішенням для систем, які працюють з великими обсягами інформації.

Для полегшення роботи з базою даних та забезпечення об'єктно-орієнтованого доступу до даних буде використана ORM бібліотека TypeORM.

Для автоматизації процесів розгортання буде застосовано підхід безперервної інтеграції та постійного розгортання (CI/CD) з використанням контейнерів Docker.

Обраний технологічний стек забезпечить створення системи, яка відповідатиме високим вимогам до якості, продуктивності, масштабованості та зручності використання як для користувачів, так і для розробників.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Основні відомості про розробку системи управління навчальним центром

Система управління навчальним центром надає можливість користувачам проходити навчальні курси, виконувати тести та оцінювати свої навички в режимі онлайн, а адміністраторам системи - додавати та редагувати курси та оцінювання.

У ході розробки системи було використано такі мови програмування, як TypeScript для серверної частини та клієнтської частини, а також фреймворки NestJS і React відповідно. Для серверної частини також використовується ORM-бібліотека TypeORM для взаємодії з базою даних PostgreSQL.

Клієнтська частина розроблена з використанням бібліотеки React та React Router DOM для управління маршрутизацією на стороні клієнта. Використання React дозволяє створювати динамічні, інтерактивні сторінки, а React Router DOM робить застосунок односторінковим (SPA), забезпечуючи плавну навігацію без перезавантаження сторінок. Це підвищує швидкість та зручність роботи з системою.

Також, оскільки для будь-якої системи управління необхідна база даних, то використовувалася база даних PostgreSQL, як мова запитів до бази даних використовується TypeORM. Приклади використаних запитів TypeORM наведено в таблиці 3.1.

Таблиця 3.1 - Приклади використаних запитів TypeORM в системі

Текст запиту	Опис
<pre> this.lessonsRepository .createQueryBuilder('lesson') .leftJoin('lesson.teamsLessons', 'teamsLessons') .leftJoinAndSelect('lesson.pages', 'pages') .leftJoinAndSelect('lesson.test', 'test') .where('teamsLessons.id IS NULL'); </pre>	Пошук загальнодоступних лекцій
<pre> this.lessonsRepository .createQueryBuilder('lesson') .leftJoinAndSelect('lesson.teamsLessons', 'teamsLessons') .leftJoinAndSelect('lesson.pages', 'pages') .leftJoinAndSelect('lesson.test', 'test') .where('teamsLessons.teamId = :teamId', { teamId }); </pre>	Пошук лекцій для команди користувача

3.2 Принцип роботи лекційної та тестувальної частини застосунка

Перед тим, як дати користувачам навчальні матеріали адміністратор має створити лекцію, створивши в ній сторінки та додавши відео. Якщо потрібно, адміністратор також додає тестове завдання. Питання в тестах можуть бути як з однією правильною відповіддю, так і з декількома.

Після створення лекцій, користувачі зможуть їх переглядати. Лекції поділяються на два типи: загальнодоступні та приватні. Загальнодоступні лекції доступні для всіх користувачів, тоді як приватні доступні лише тим користувачам, які є членами команди, що отримала доступ до цих лекцій.

Після проходження лекції користувач отримує доступ до тестового завдання якщо його додано. Коли користувач відповідає на всі запитання і підтверджує завершення тесту, адміністратору надсилаються відповіді на його корпоративну пошту. Схему роботи лекційної та тестувальної частини зображено на рисунку 3.1.

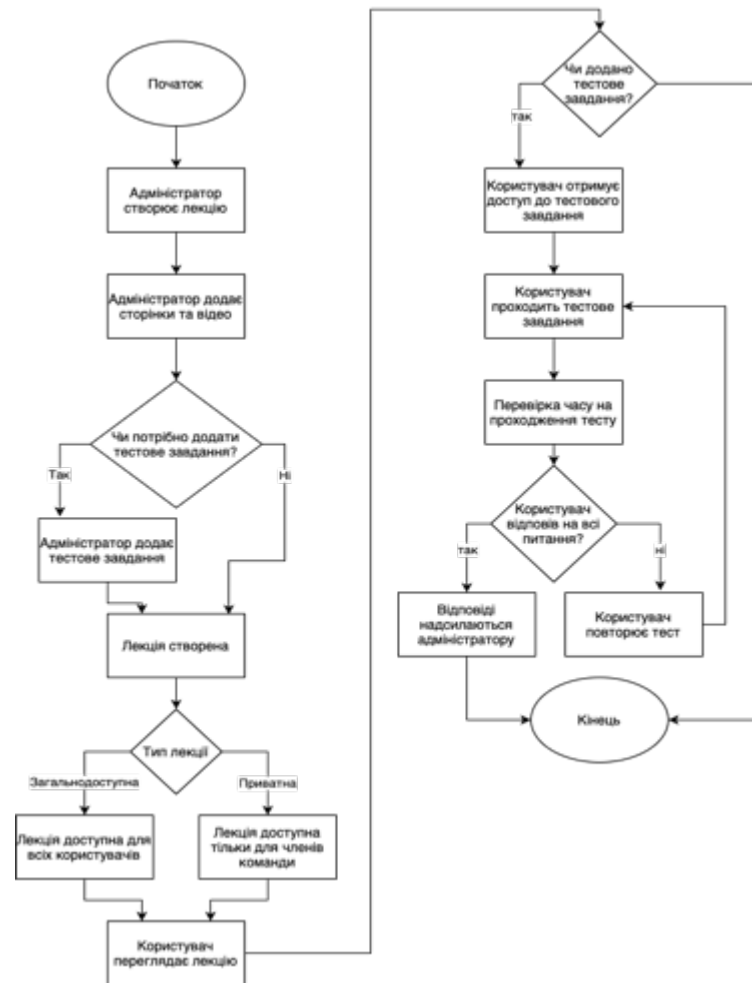


Рисунок 3.1 – Функціонування тестувальної частини

3.3 Принцип роботи частини оцінювання методом «360 градусів»

Для функціонування оцінювання методом «360 градусів» адміністратор має спочатку створити два тести: тест для самооцінки та тест для оцінки інших людей у команді (співробітників, керівників та підлеглих). Питання повинні містити заздалегідь прописані варіанти відповідей: абсолютно не погоджуюсь, не погоджуюсь, нейтрально, погоджуюсь, абсолютно погоджуюсь. Питання поділяються на кілька типів: вирішення проблем, комунікація, лідерство та відповідальність, організаційні навички, професійний розвиток, командна робота.

Після створення тестів адміністратор може перейти до створення «Компанії 360», вказавши її назву та призначивши команди для тестування.

Після створення «Компанії 360» адміністратор може побачити весь перелік оцінювань, відредагувати команди за потреби та натиснути кнопку «Активувати».

Після активації «Компанії 360» користувачі зможуть побачити її на сторінці «Оцінка 360». Після переходу на сторінку компанії користувачі побачать перелік завдань, де буде зазначено, кого вони мають оцінити. При натисканні на завдання користувач потрапить у меню оцінки, де він має відповісти на запитання та підтвердити свій вибір. Схему роботи частини оцінювання методом «360 градусів» зображено на рисунку 3.2.



Рисунок 3.2 – Функціонування частини оцінювання методом «360 градусів»

3.4 Алгоритм функціонування адміністративної частини

У створеному вебзастосунку була розроблена адміністративна панель, яка спрощує процес керування вмістом та стеження за тестуваннями.

Щоб розпочати роботу з адміністративною панеллю, спочатку необхідно увійти в систему. Для цього потрібно ввести облікові дані. Якщо дані правильні і користувач має роль адміністратора, йому відкриється сторінка "Панель адміністратора".

В панелі адміністратора є кілька основних розділів, кожен з яких має свої функціональні можливості. У розділі "Користувачі" можна переглядати користувачів, редагувати їх дані, такі як ПІБ, ППН, номер телефону та пароль, а також видаляти або створювати нових користувачів. Це дозволяє адміністратору ефективно керувати інформацією про всіх користувачів системи.

Розділ "Команди" дозволяє переглядати існуючі команди, редагувати дані про керівника, керуючу команду, підлеглі команди та підлеглих співробітників, а також видаляти або створювати нові команди. Це забезпечує зручне управління структурою команди та її членами.

У розділі "Тести 360" адміністратор може створювати тести для оцінювання методом "360 градусів", редагувати та видаляти їх. Розділ "Компанії 360" дозволяє створювати компанії для оцінювання за методом "360 градусів", редагувати та видаляти їх, якщо компанія не була завершена.

Розділ "Звіт компаній 360" надає можливість переглядати статистику оцінювань методом "360 градусів" за допомогою звітів, створених у PowerBI. Це допомагає отримати детальний аналіз результатів оцінювання.

Розділ "Тестування" дозволяє створювати тести для лекцій, редагувати та видаляти їх. У розділі "Навчання" можна переглядати існуючі лекції, створювати нові, редагувати та видаляти їх.

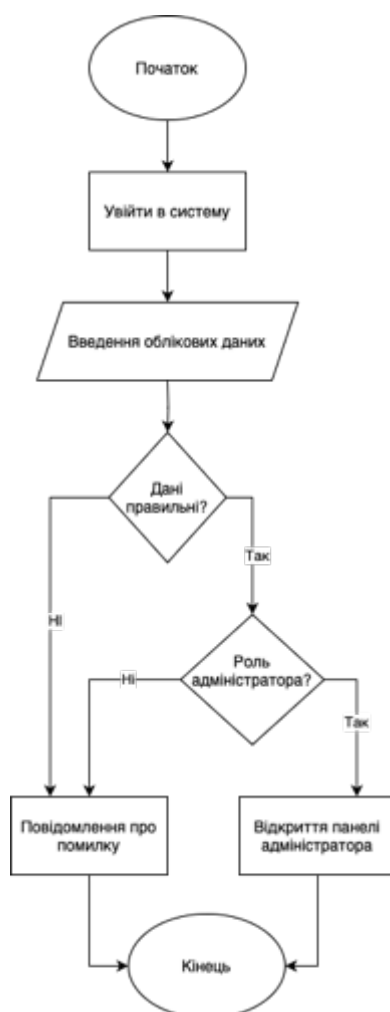


Рисунок 3.3 – Функціонування адміністративної панелі

У розділі "Навчання" можна переглядати існуючі лекції, створювати нові, редагувати та видаляти їх. Це забезпечує ефективне управління навчальним контентом на вебсайті.

3.4 Висновки за розділом

У цьому розділі представлена основна інформація про розробку системи управління навчальним центром. Описано ключові алгоритми, які були використані в процесі роботи над проектом, зокрема алгоритми роботи лекційного та тестувального модулів, а також адміністративної панелі.

Створено детальний опис принципів функціонування адміністративної панелі, яка забезпечує ефективне керування користувачами, командами та навчальним контентом.

Загальна характеристика вебзастосунку включає опис основних функціональних блоків і процедур, які забезпечують роботу системи, таких як додавання та редагування курсів, оцінювання навичок за методом «360 градусів» та управління тестуваннями.

4 Керівництво програміста

4.1 Призначення й умови використання навчального центра

Система управління навчальним центром призначена для автоматизації процесів навчання, тестування та оцінювання навичок користувачів онлайн. Серверна частина системи була розроблена за допомогою мови програмування TypeScript та фреймворка NestJS. Клієнтська частина створена з використанням бібліотеки React та бібліотеки компонентів Material UI. Як база даних використовується СКБД PostgreSQL.

Надійна робота системи управління навчальним центром залежить від наявності стабільного з'єднання з мережею Інтернет та наявності всіх модулів системи та бази даних на вебсервері.

Для роботи системи потрібен комп'ютер з пристроями введення інформації (клавіатура та комп'ютерна миша) або ноутбук, чи інший пристрій з можливістю відображення вебсторінок та доступом до Інтернету. На пристрої повинен бути попередньо встановлений веббраузер: Google Chrome, Opera, Mozilla Firefox, Microsoft Edge або інший.

4.2 Характеристика продукту

Застосунок ділиться на дві частини: клієнтська та серверна.

Клієнтська можна поділити на такі логічні блоки:

- Main.tsx – цей файл відповідає за ініціалізацію react router dom, який відповідає за шляхи окремих сторінок;
- SharedLayout.tsx – файл, який відповідає за загальний вигляд інтерфейсу;
- Auth – блок, що відповідає за реєстрацію, вхід та відновлення пароля користувача;

- Account – сторінка користувача, де він може продивитися свої данні та статистику оцінювань за методом «360 градусів». Також при потребі він може відредагувати свої данні та змінити пароль;
- Lessons – блок, що відповідає за лекції та їх матеріал;
- Tests – блок, що відповідає за тестувальну частину лекцій;
- Tests360 – блок, що відповідає за тестування методом «360 градусів».

Серверну можна поділити на такі логічні блоки:

- main.ts – файл ініціалізації сервера та підключення до бази даних;
- typeorm.ts – конфігураційний файл для налаштування підключення до бази даних PostgreSQL з використанням TypeORM;
- validators – блок, що відповідає за функції, які валідують данні;
- auth – блок, що відповідає за реєстрацію, автентифікацію, авторизацію та відновлення пароля;
- users, tests360, tests, lessons, teams, – блоки, що відповідають за опис сутностей та їх функціоналу.

4.3 Структура бази даних

- База даних складається з декількох таблиць, зв'язаних між собою різними типами зв'язків, включаючи зв'язки «один до багатьох» та «багато до багатьох»:
 - user - містить інформацію про користувачів, включаючи їх ідентифікатори, імена, електронні адреси, паролі, адміністраторські права, контактні дані, дати створення та оновлення та статус реєстрації;
 - teams - містить дані про команди, включаючи імена команд, дати створення та оновлення, ідентифікатори керівників та підлеглих команд;
 - team_members - зберігає дані про членів команд, включаючи ідентифікатори команд та користувачів;

- password_reset_tokens - зберігає дані про токени для скидання паролів, включаючи токени, дати закінчення терміну дії, статус використання та ідентифікатори користувачів;
- test - містить інформацію про тести, включаючи заголовки, реферальний код, бали, час виконання, дати створення та оновлення, ідентифікатори користувачів, які створили тести;
- tests360 - зберігає інформацію про тести, створені для оцінювання за методом «360 градусів», включаючи опис, реферальний код, тип тесту, дати створення та оновлення, ідентифікатори користувачів, які створили тести;
- company360 - містить дані про компанії, які проводять оцінювання за методом «360 градусів», включаючи назву, команди, активний статус, дати створення та оновлення, ідентифікатори тестів для самооцінки та оцінки інших членів команди;
- completed_test - зберігає дані про завершені тести, включаючи статус завершення, бали, дати створення та оновлення, ідентифікатори користувачів та тестів;
- lesson - містить дані про навчальні лекції, включаючи заголовки, дати створення та оновлення, реферальний код, ідентифікатори тестів;
- lesson_page - містить дані про сторінки лекцій, включаючи заголовки, номери, URL-адреси відео, дати створення та оновлення, ідентифікатори лекцій;
- test_question - зберігає дані про питання для тестів, включаючи заголовки, номери питань, бажані бали, дати створення та оновлення, ідентифікатори тестів;
- question_answer - зберігає дані про відповіді на запитання, включаючи текст, правильність, дати створення та оновлення, ідентифікатори питань;
- user_answer - зберігає дані про відповіді користувачів на запитання, включаючи дати створення та оновлення, ідентифікатори запитань, відповідей та користувачів;

- `users_test` -: зберігає дані про спроби проходження тестів користувачами, включаючи статус завершення, кількість спроб, дати останньої та передостанньої спроби, дати створення та оновлення, ідентифікатори користувачів та тестів;
- `Test360QuestionAnswers` - зберігає дані про відповіді на питання тестів за методом «360 градусів», включаючи текст, бали, дати створення та оновлення;
- `Test360Questions` - містить дані про питання тестів за методом «360 градусів», включаючи заголовки, номери питань, тип питань, дати створення та оновлення, ідентифікатори тестів;
- `Company360Task` - містить дані про завдання в рамках компаній «360 градусів», включаючи статус завершення, дати створення та оновлення, ідентифікатори оцінювачів, оцінюваних користувачів, компаній та тестів;
- `Test360UserAnswer` - зберігає дані про відповіді користувачів на питання тестів за методом «360 градусів», включаючи дати створення та оновлення, ідентифікатори завдань, користувачів, питань та відповідей;
- `teams_lessons` - зберігає дані про доступ команд до лекцій, включаючи дати створення та оновлення, ідентифікатори команд та лекцій.

Схема структури бази даних представлена на рис. 4.1.

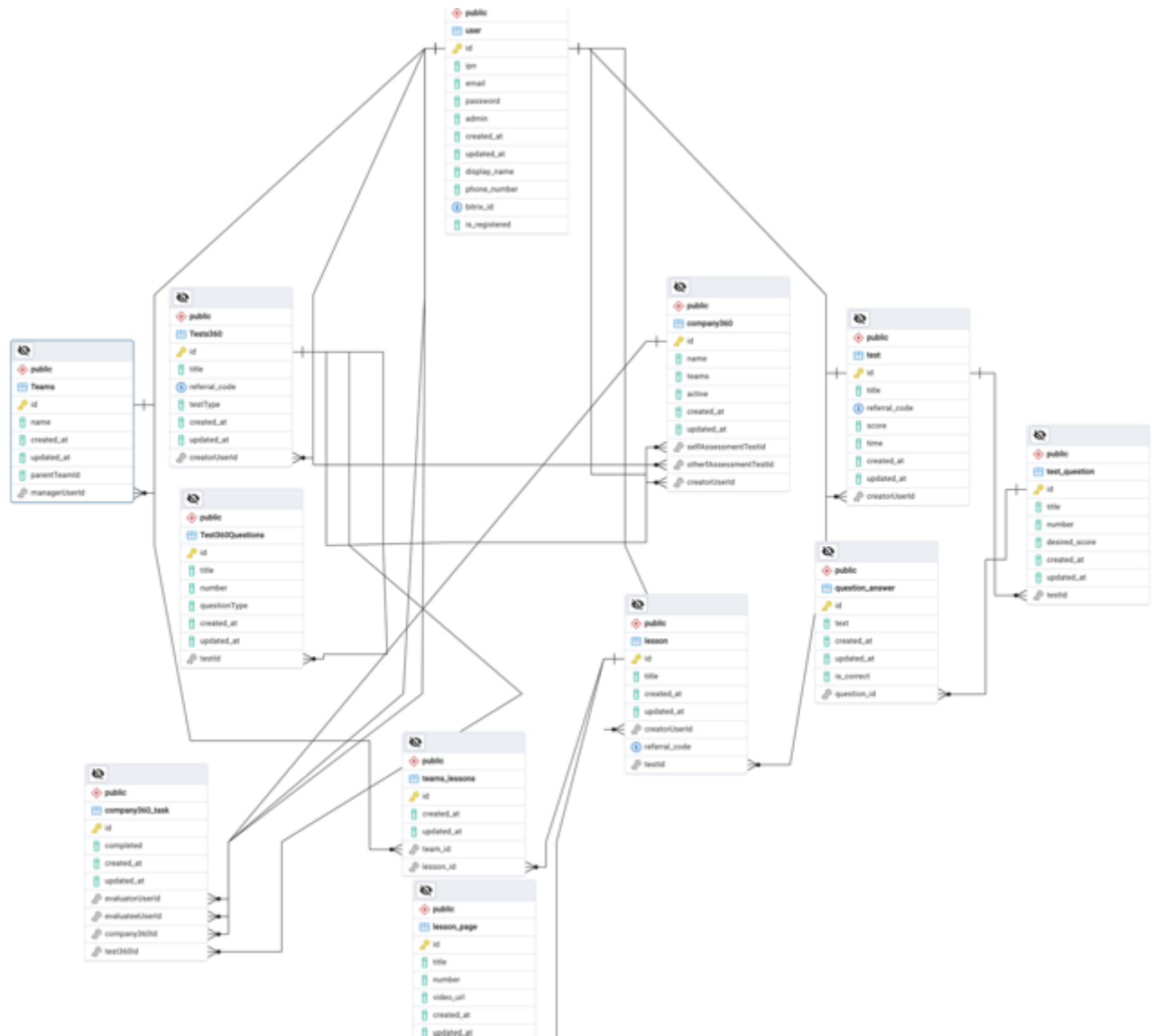


Рисунок 4.1 – Схема структури бази даних

4.4 Структура застосунку

Застосунок можна умовно поділити на дві основні частини: клієнтську та адміністративну. Клієнтська частина забезпечує взаємодію з користувачами, надаючи їм доступ до лекцій, тестів і можливості оцінювання за методом "360 градусів". Адміністративна частина призначена для керування базою даних і доступна лише адміністраторам. У цій частині можна додавати, редагувати, видаляти та фільтрувати дані у всіх таблицях. Схему застосунку представлено на рисунку 4.2

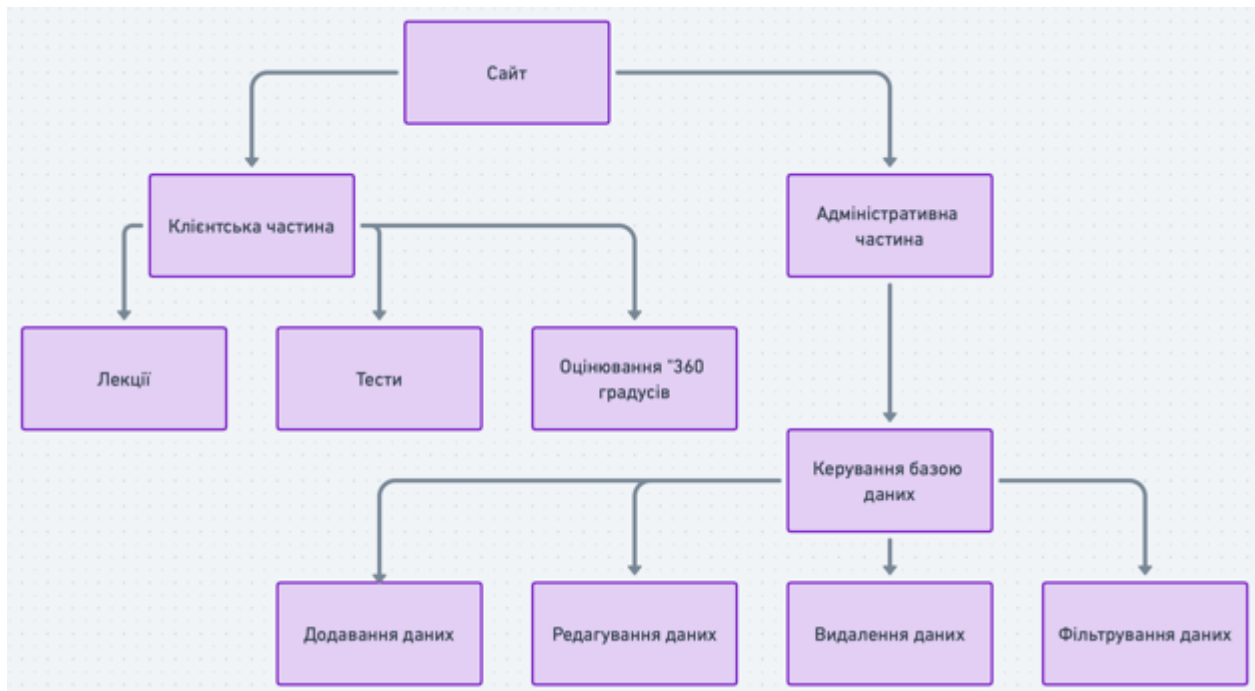


Рисунок 4.2 – Схема структури застосунку

4.5 Висновки за розділом

У цьому розділі розглянуті основні логічні блоки, створені під час розробки навчального центру. Детально описана структура бази даних, включаючи первинні ключі та основні зв'язки між таблицями. Також представлена структурна схема компонентів навчального центру, яка ілюструє логічну взаємодію між розробленими компонентами. Описано структуру бази даних навчального центру та схему розробленої системи.

5 КЕРІВНИЦТВО СИСТЕМНОГО АДМІНІСТРАТОРА

5.1 Розгортання навчального центра

Навчальний центр знаходиться на локальних репозиторіях з назвами «echr-frontend» та «echr-backend».

Щоб розмістити застосунок на вебсервері потрібно:

- завантажити проєкт з репозиторіїв;
- встановити Docker та розширення Docker compose;
- налаштувати вебсервер;
- створити базу даних сайту;
- налаштувати .env файли;
- запустити контейнери frontend та backend частин;

5.2 Розгортання навчального центра

Щоб створити вебсервер за допомогою Nginx, необхідно створити конфігураційний файл у папці «/etc/nginx/sites-available». Після цього потрібно створити символічне посилання на цей файл у директорії «sites-enabled» та перезапустити сервіс Nginx.

5.3 Розгортання бази даних

Для розгортання бази даних необхідно перемістити файл db.yml до зручної папки та перейменувати його на docker-compose.yml. Після цього потрібно відредагувати цей файл, змінивши значення в рядках «POSTGRES_DB», «POSTGRES_USER» та «POSTGRES_PASSWORD» на свої власні дані. Тут POSTGRES_DB - це назва бази даних, POSTGRES_USER - ім'я користувача, а POSTGRES_PASSWORD - пароль від бази даних. Після внесення змін, запустіть контейнер командою: `docker-compose up -d`.

5.4 Розгортання серверної частини

Перед запуском серверної частини необхідно створити файл з назвою «.env» та вказати в ньому значення для наступних параметрів: POSTGRES_PASSWORD, POSTGRES_DATABASE, POSTGRES_USER, DATABASE_PORT, CORS_ORIGINS, MAIL_HOST, MAIL_PORT, MAIL_USER, MAIL_PASS. POSTGRES_PASSWORD, POSTGRES_DATABASE, POSTGRES_USER, POSTGRES_USER, DATABASE_PORT, CORS_ORIGINS, MAIL_HOST, MAIL_PORT, MAIL_USER, MAIL_PASS, де:

- POSTGRES_PASSWORD – пароль від бази даних,
- POSTGRES_DATABASE – назва бази даних,
- POSTGRES_USER – ім'я користувача,
- DATABASE_PORT – порт бази даних,
- CORS_ORIGINS - список дозволених джерел для CORS (Cross-Origin Resource Sharing),
- MAIL_HOST – хост сервера електронної пошти,
- MAIL_PORT – порт сервера електронної пошти,
- MAIL_USER – електронна пошта,
- MAIL_PASS – пароль від електронної пошти.

Після налаштування змінних середовища, потрібно запустити Docker контейнер за допомогою команди: `docker-compose up --build -d`. Після розгортання сервера автоматично відбудеться міграція бази даних. Міграція – це процес внесення змін до структури бази даних, що включає створення, зміну або видалення таблиць, стовпців та інших об'єктів.

5.5 Розгортання клієнтської частини

Після успішного розгортання бази даних та backend частини залишається записати таку змінну в середовищі, як

REACT_APP_BACKEND_URL, яка посилається на адресу backend частини та запустити за допомогою команди `docker-compose up --build -d`.

5.6 Адміністративна панель навчального центра

В навчальному центрі є адміністративна панель, доступна лише для адміністраторів вебсайту, яка забезпечує автоматизацію та спрощення роботи з необхідним контентом. Щоб потрапити в панель адміністратора потрібно зайти на сайт під своїм логіном та паролем та натиснути на кнопку «Панель адміністратора» в боковому навігаційному меню. Потрапивши до адміністративної панелі можна побачити наступні пункти: Користувачі, Команди, Тести 360, Компанії 360, Звіт компаній 360, Тестування та Навчання.

На сторінці користувачів адміністратор може створити, видалити або відредагувати вже створеного користувача, як показано на рисунку 5.1.

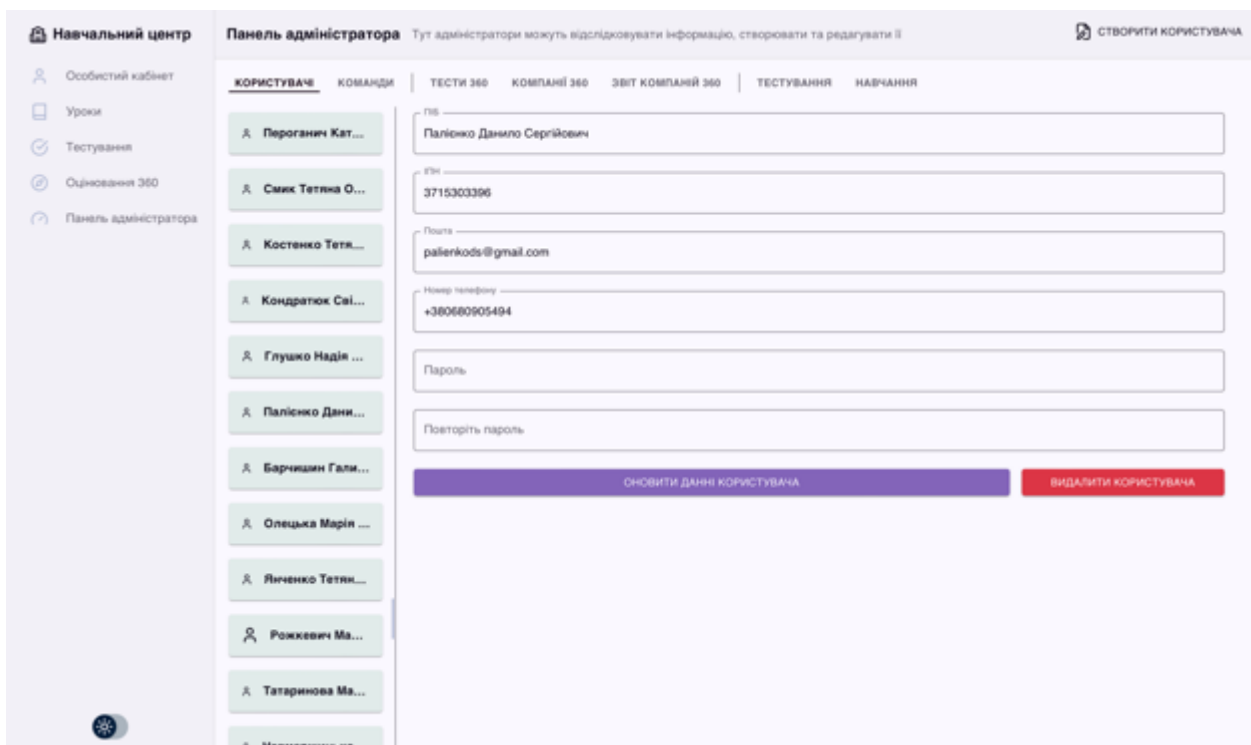


Рисунок 5.1 – Сторінка редагування користувача

На рисунку 5.2 відображено сторінку команд, де можна відредагувати їх назву, керівника, керуючу команду, підлеглі команди та підлеглих працівників.

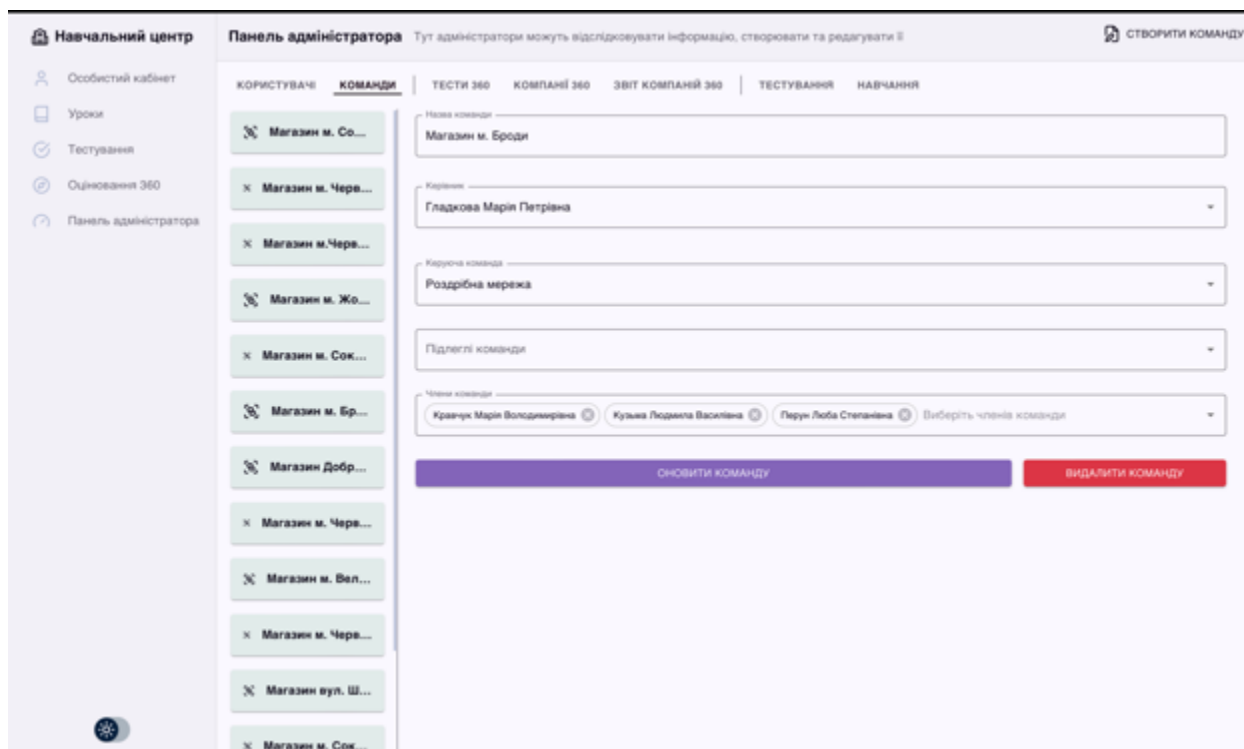


Рисунок 5.2 – Сторінка редагування команди

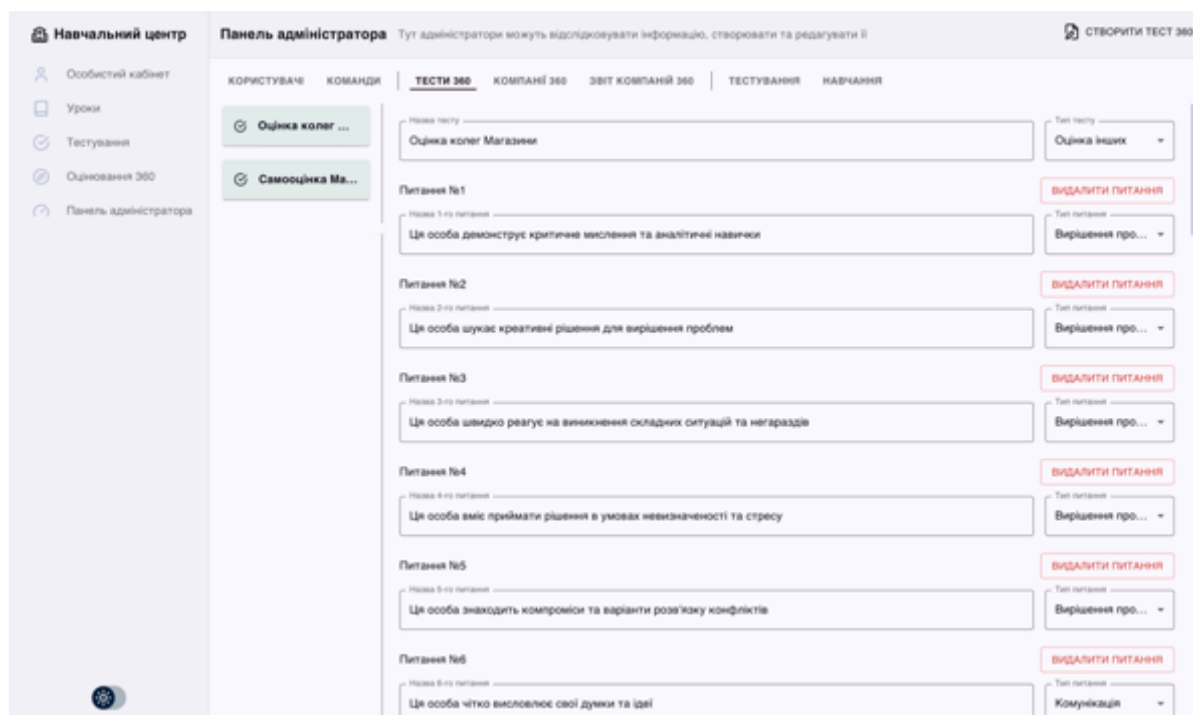


Рисунок 5.3 – Сторінка редагування тестів для оцінювання команд методом «360 градусів»

На рисунку 5.4 відображено сторінку «Компаній 360», де можна відредагувати назву, тести на самооцінку та оцінку інших працівників, команди та побачити наявність виконання завдань.

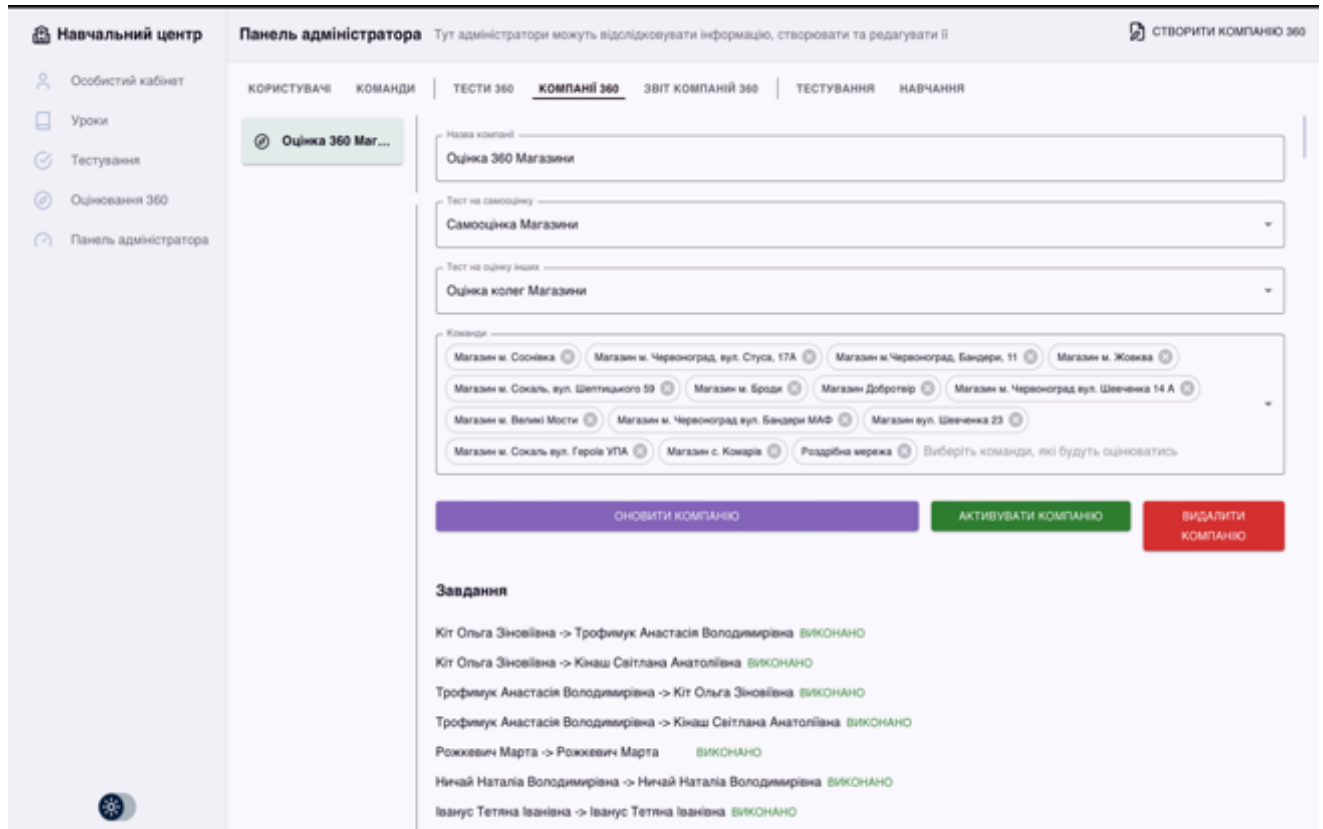


Рисунок 5.4 – Сторінка редагування «Компаній 360»

На сторінці «Звіти компаній 360» адміністратор може побачити звіти оцінюванням колег. Тут можна відсортувати звіти по «Компаніям 360», командам, оцінювачах та оцінюваних і побачити діаграму, яка показує результати оцінювання. Вікно звіту відображено на рисунку 5.5.

На сторінці «Тестування» адміністратор може редагувати тести для лекцій, як зображено на рисунку 5.6.

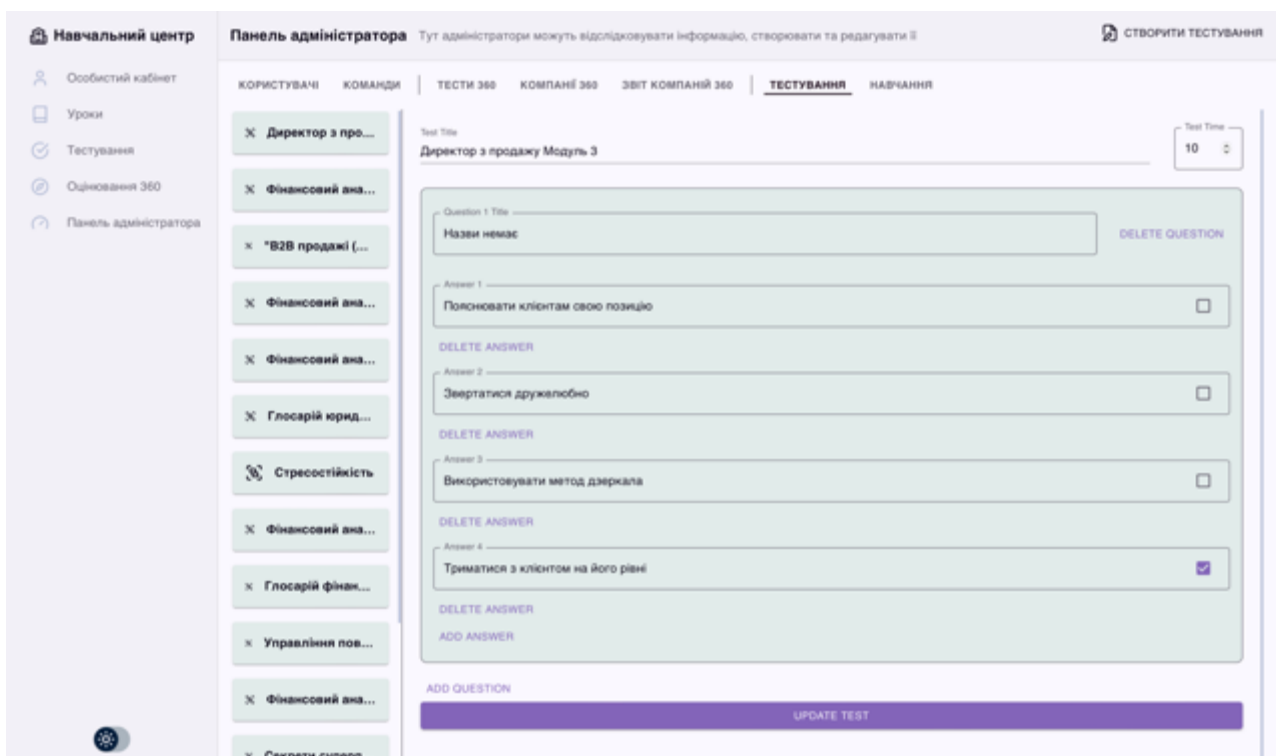


Рисунок 5.6 – Сторінка редагування тестів для лекцій

На останній сторінці адміністративної панелі «Навчання» адміністратор може створювати та редагувати лекції, завдяки яким користувачі зможуть навчатися та покращувати свої професійні навички. Сторінку редагування лекцій можна побачити на рисунку 5.7.

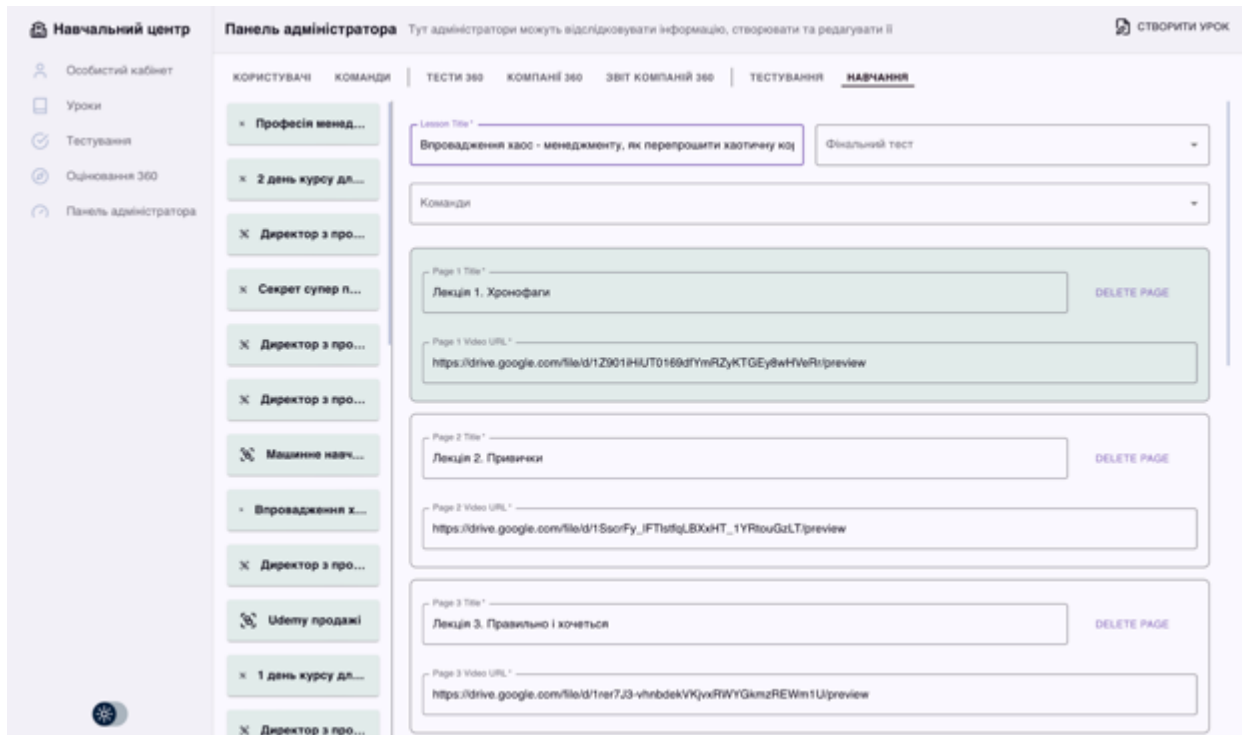


Рисунок 5.7 – Сторінка редагування лекції

5.7 Висновки за розділом

У цьому розділі надано опис призначення та умов використання вебзастосунку. Деталізовано процес розгортання frontend і backend частин разом із базою даних та створення вебсервера. Також подано загальний огляд адміністративної панелі та редагування даних.

6 КЕРІВНИЦТВО КОРИСТУВАЧА

6.1 Призначення вебзастосунку

Вебзастосунок призначений для співробітників в середині компаній.

Для надійної експлуатації вебсайту користувачам необхідно забезпечити стабільне з'єднання з інтернетом та наявність усіх необхідних модулів і бази даних на вебсервері.

Для роботи з вебзастосунком необхідний комп'ютер із клавіатурою та мишею, ноутбук або інший пристрій, здатний вводити інформацію, відображати вебсторінки та підключатися до інтернету. На пристрої повинен бути встановлений один із веббраузерів, таких як Google Chrome, Opera, Mozilla Firefox, Internet Explorer або будь-який інший аналогічний браузер.

6.2 Авторизація, процес навчання та проведення тестувань

Щоб відкрити цей вебзастосунок, потрібно ввести його вебадресу в адресний рядок браузера та зареєструвати свій профіль або увійти до нього, якщо він вже створений. Приклад входу до вебзастосунку зображено на рисунку 6.1.

Рисунок 6.1 – Відкриття вебзастосунку

Після авторизації користувач потрапляє на його персональну сторінку, де він може побачити його персональні данні та відредагувати їх за потреби. Також на сторінці користувача він може побачити статистику, яка показує як його оцінюють його колеги, підлеглі та керівник. Зліва можна побачити бічну навігаційну панель, за допомогою якої користувач може потрапити на потрібні йому сторінки. Загальний вигляд застосунку та персональної сторінки користувача зображений на рисунку 6.2.

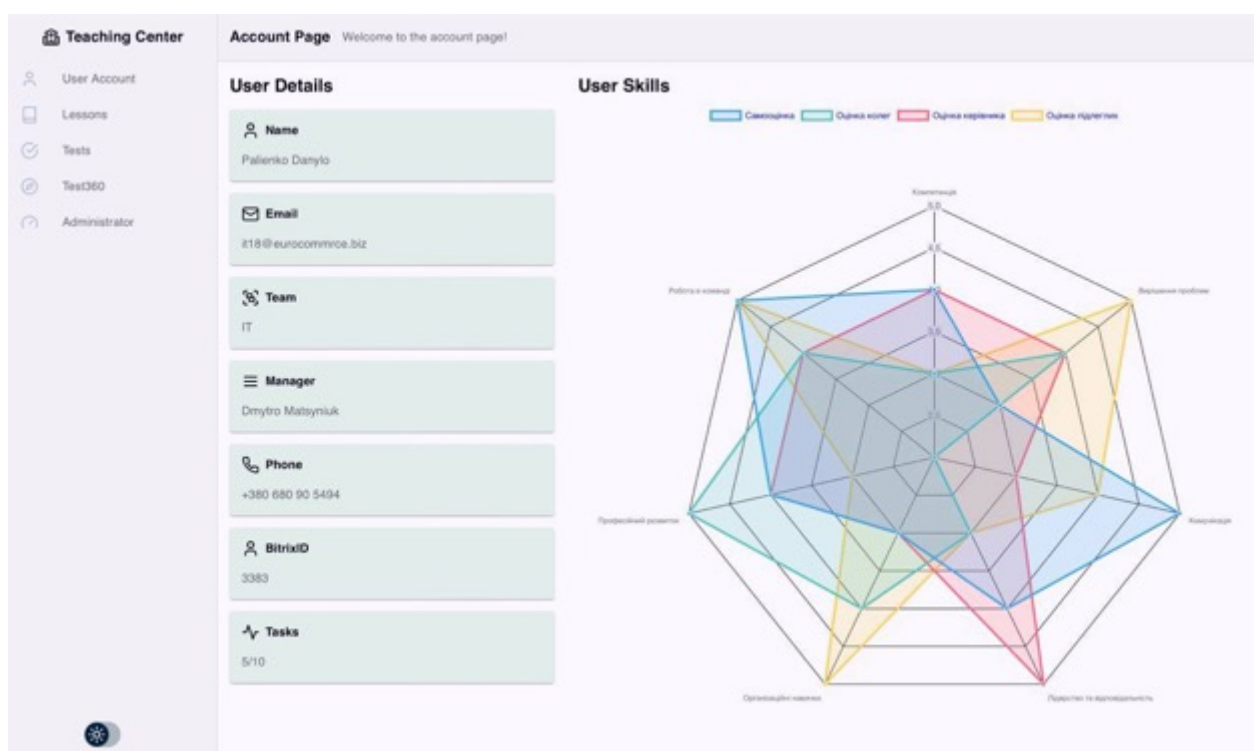


Рисунок 6.2 – Загальний вигляд застосунку та персональної сторінки користувача

Головною частиною сайту являється частина навчання та тестування. Щоб потрапити на сторінку з лекціями користувачу потрібно натиснути на кнопку «Уроки» в боковому меню. На цій сторінці користувач може побачити строку пошуку, розділ з лекціями, які призначені його команді та розділ із загальнодоступними лекціями, як зображено на рисунку 6.3.

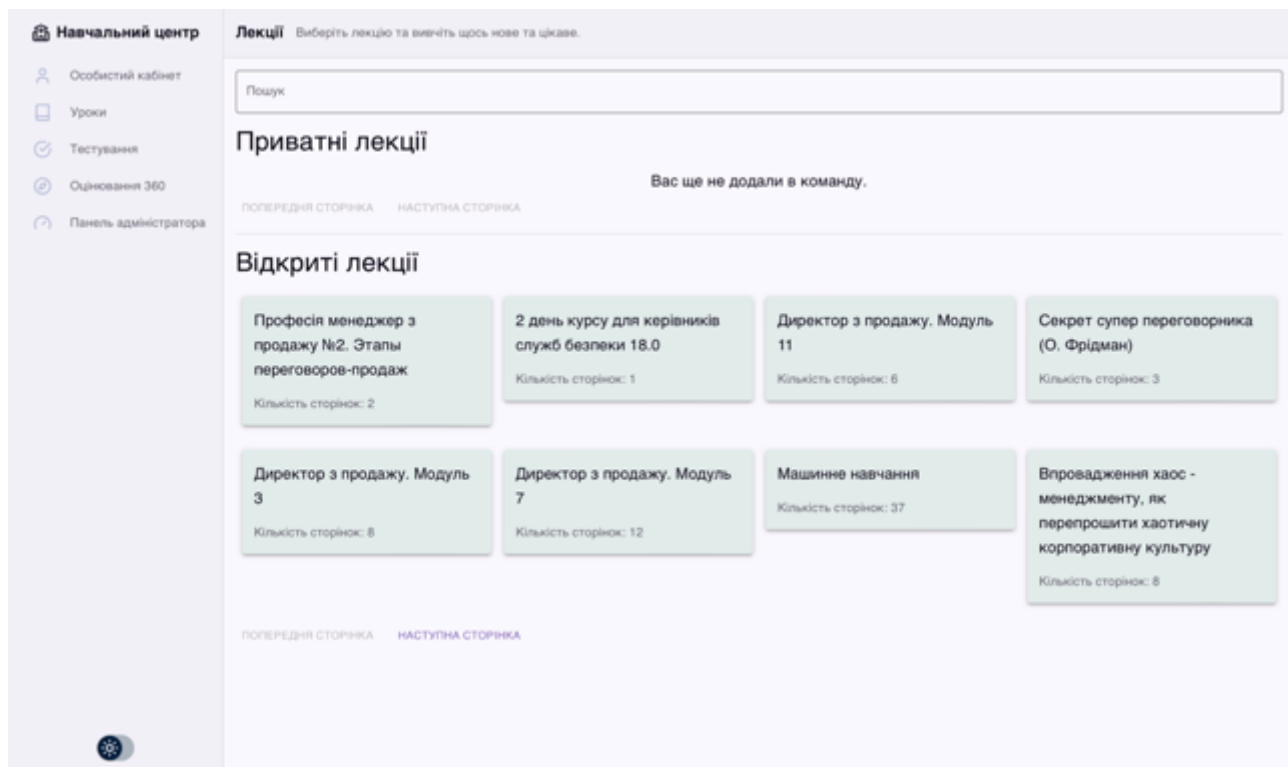


Рисунок 6.3 – Сторінка лекцій

Сам процес вивчення матеріалу складається з двох частин: вивчення матеріалу послідовно продивляючись сторінки лекцій та проходження тестового завдання. Приклади сторінки та тесту зображені на рисунках 6.4, 6.5, 6.6, відповідно.

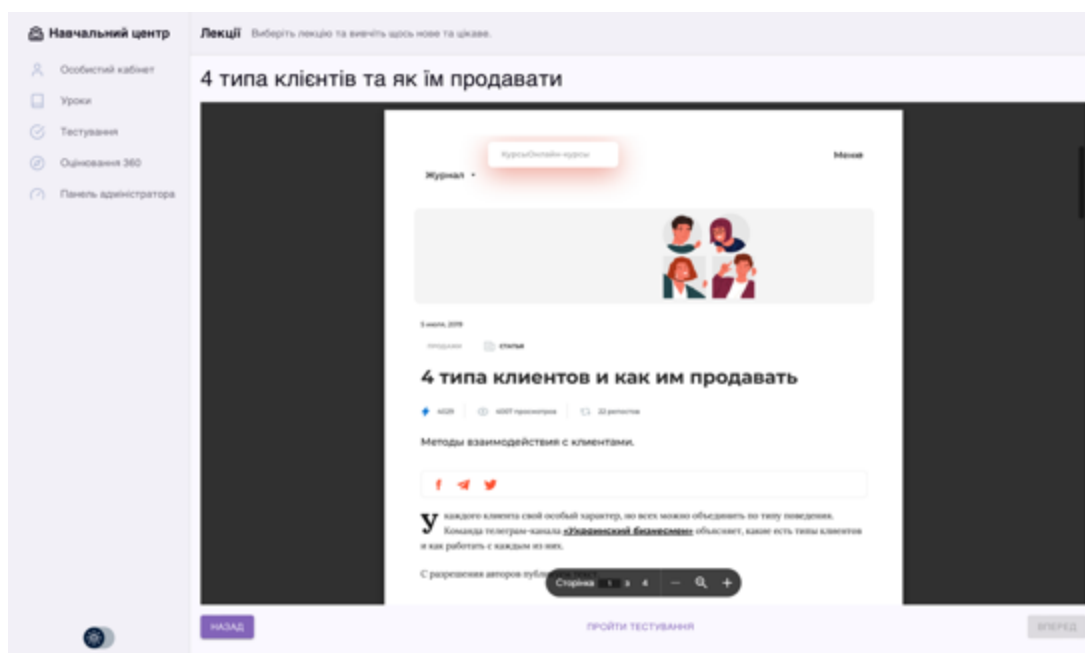


Рисунок 6.4 – Сторінка лекції

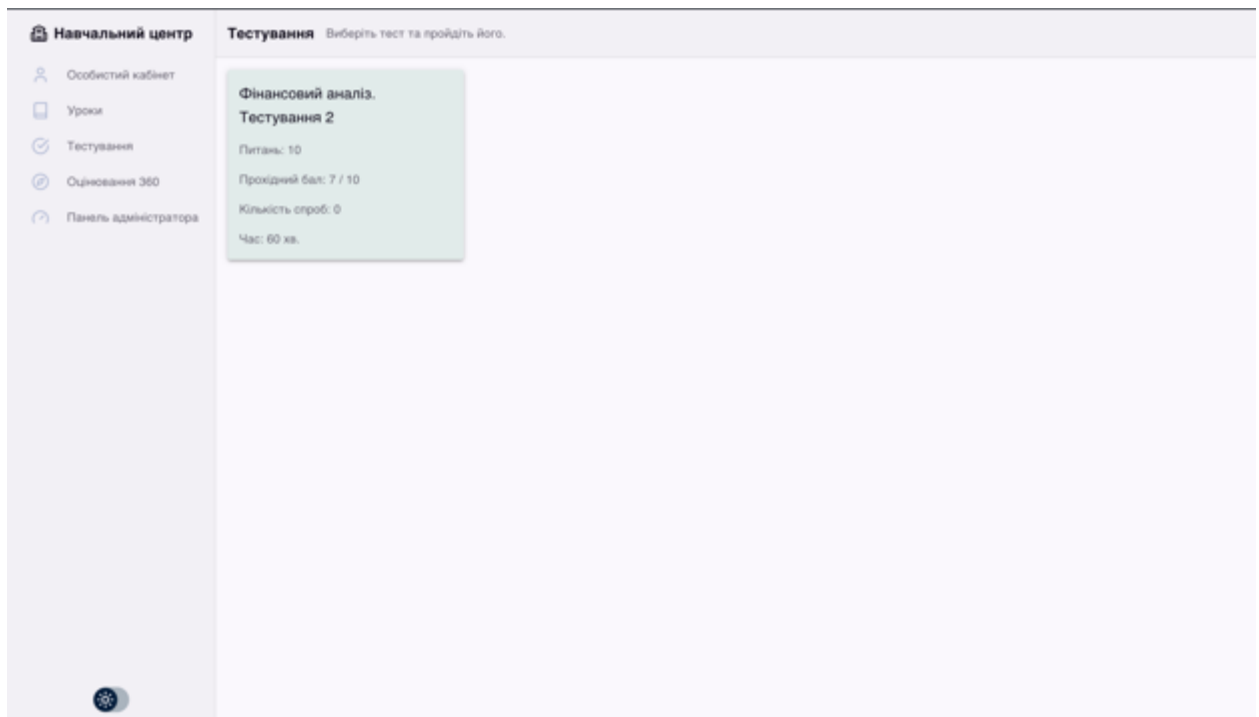


Рисунок 6.5 – Сторінка тестів для проходження

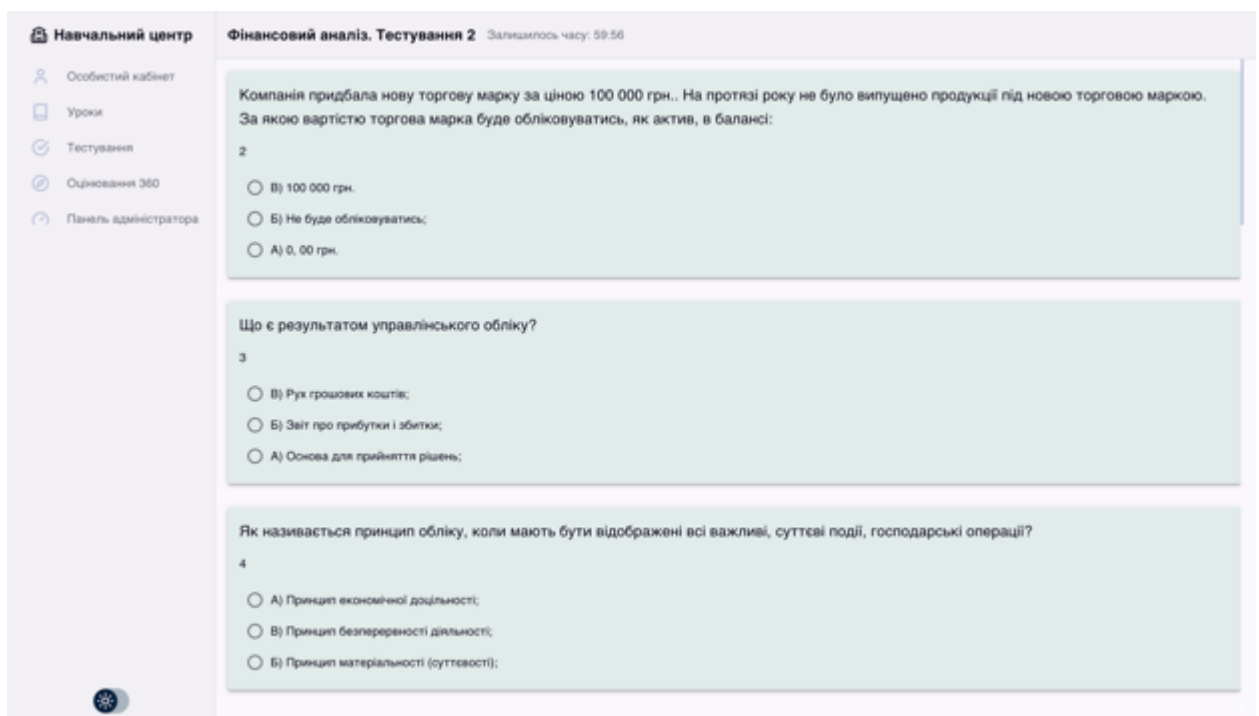


Рисунок 6.6 - Сторінка проходження тесту

Останньою частиною навчального центру є тестування методом «360 градусів». Після створення адміністратором тестів та «Компанії 360» користувач, зайшовши на сторінку «Оцінювання 360» зможе побачити компанії, які йому потрібно пройти. Зайшовши на сторінку «Компанії 360»

користувач побачить список завдань, де його потрібно оцінити себе та інших співробітників, як показано на наступних рисунках 6.7, 6.8 та рисунку 6.9.

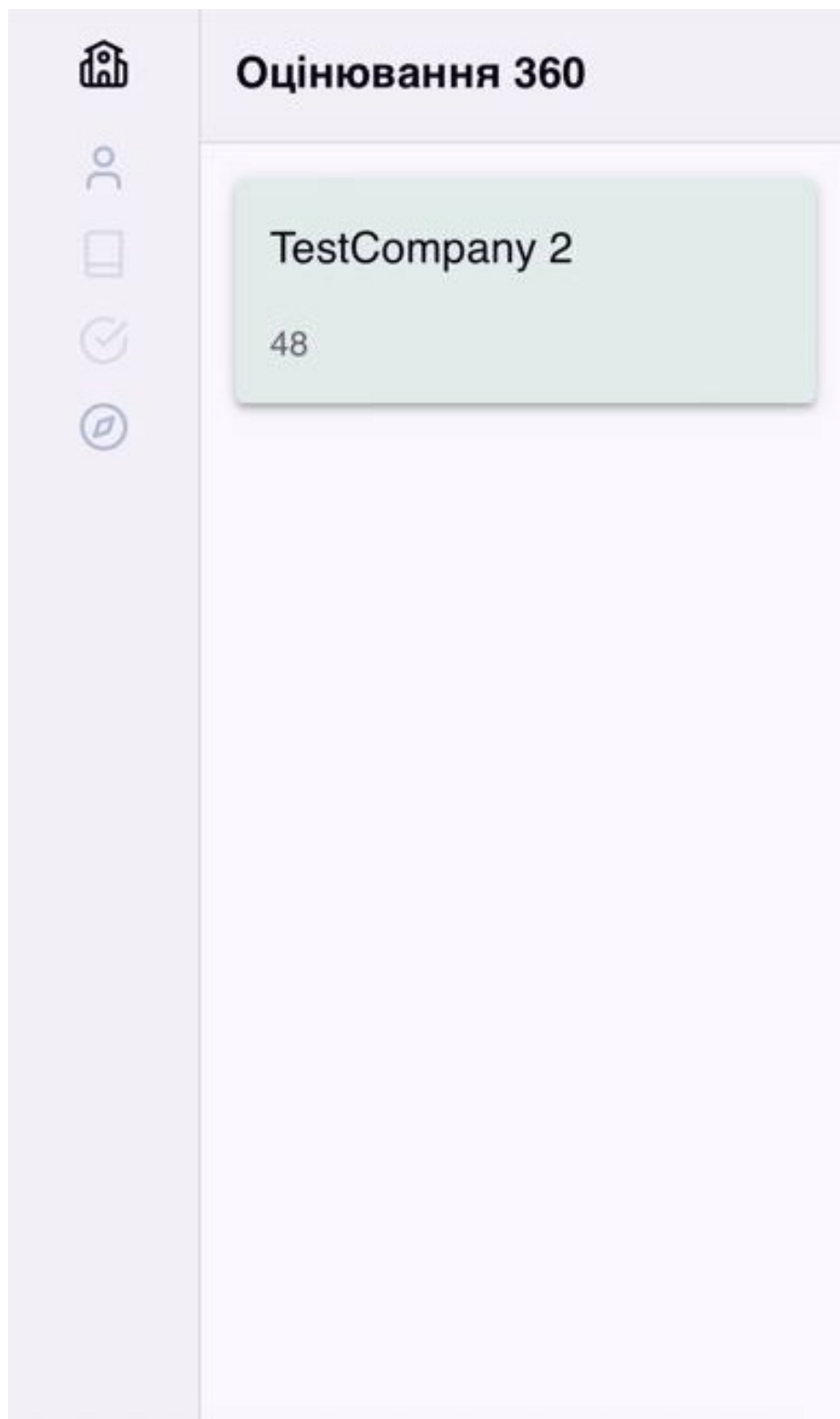
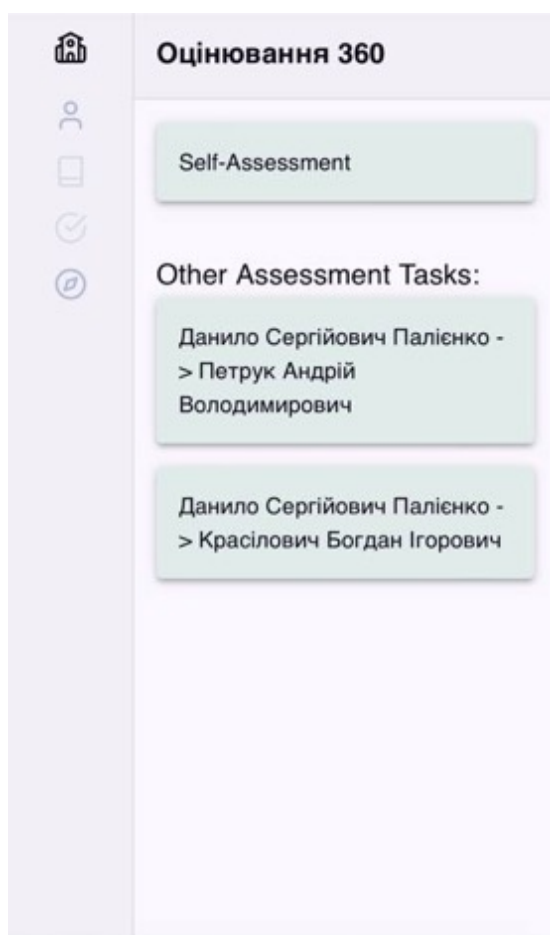


Рисунок 6.7 – Сторінка «Компаній 360»



Оцінювання 360

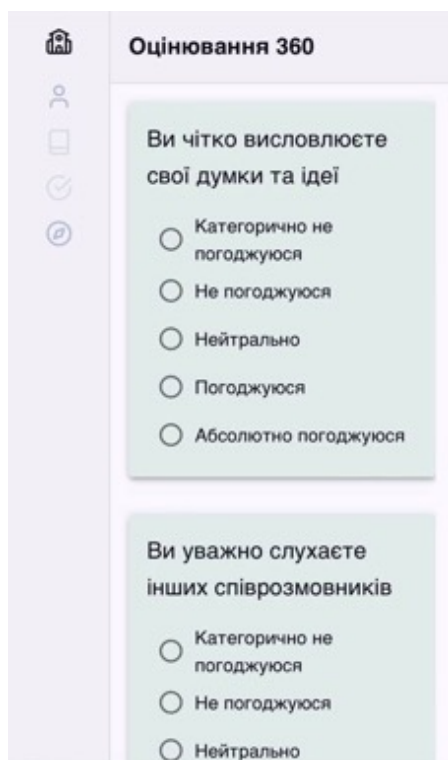
Self-Assessment

Other Assessment Tasks:

Данило Сергійович Палієнко -
> Петрук Андрій
Володимирович

Данило Сергійович Палієнко -
> Красілович Богдан Ігорович

Рисунок 6.8 – Сторінка завдань в «Компанії 360»



Оцінювання 360

Ви чітко висловлюєте свої думки та ідеї

☐ Категорично не погоджуюся

☐ Не погоджуюся

☐ Нейтрально

☐ Погоджуюся

☐ Абсолютно погоджуюся

Ви уважно слухаєте інших співрозмовників

☐ Категорично не погоджуюся

☐ Не погоджуюся

☐ Нейтрально

Рисунок 6.9 – Сторінка оцінки співробітників методом «360 градусів»

6.3 Висновки за розділом

У розділі було описано, як користувач взаємодіє з навчальним центром, включаючи способи авторизації, пошук інформації та проходження тестів.

Інтерфейс розробленого вебзастосунку створений таким чином, щоб бути максимально зручним та зрозумілим для будь-якого користувача. Завдяки цьому, він не повинен викликати жодних труднощів при виконанні основних функцій та використанні за призначенням.

ВИСНОВКИ

Під час виконання дипломної кваліфікаційної роботи було створено вебзастосунок «Навчальний центр» для оцінки та покращення професійних навичок.

Після завершення тестування та налагодження роботи навчального центру можна зробити наступні висновки:

- функціонал навчального центру повністю відповідає вимогам технічного завдання;
- вебзастосунок працює стабільно та автономно;
- інтерфейс вебзастосунку зручний та підтримує діалог із користувачем;
- вебзастосунок забезпечує валідацію даних при введенні та виведенні;
- навчальний центр коректно відображається на всіх сучасних браузерах та платформах.

Для розробки проєкту було обрано мову програмування TypeScript з використанням фреймворка NestJS для серверної частини та React для клієнтської частини.

Для бази даних було обрано PostgreSQL. Розроблено структуру бази даних для навчального центру та описано блок-схему створеної системи.

Також було проаналізовано середовище розробки, яке відповідає критеріям, зазначеним у постановці задачі, та визначено необхідний набір інструментів для вирішення проєктної задачі.

Обґрунтовано вибір мови програмування та середовища функціонування системи, що забезпечує оптимальну продуктивність і відповідність технічним вимогам проєкту.

В результаті випробувань розроблений навчальний центр демонструє здатність швидко навчати персонал, перевіряти знання та проводити оцінювання за методом "360 градусів".

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Оцінка персоналу за методом 360 градусів [Електрон. ресурс].
– Режим доступу:
https://uk.wikipedia.org/wiki/Оцінка_персоналу_за_методом_360_градусів
2. Moodle [Електрон. ресурс]. – Режим доступу: <https://moodle.org>
3. TalentLMS [Електрон. ресурс]. – Режим доступу:
<https://www.talentlms.com>
4. Qualtrics [Електрон. ресурс]. – Режим доступу:
<https://www.qualtrics.com>
5. Методи оцінювання для досягнення цілей [Електрон. ресурс] –
Режим доступу: <https://www.questionpro.com/blog/performance-evaluation-methods/>
6. Документація TypeScript [Електрон. ресурс]. – Режим доступу:
<https://www.typescriptlang.org>
7. Документація NestJS [Електрон. ресурс]. – Режим доступу:
<https://docs.nestjs.com>
8. Документація TypeORM [Електрон. ресурс] – Режим доступу:
<https://typeorm.io>
9. Документація Python [Електрон. ресурс]. – Режим доступу:
<https://www.python.org>
10. Документація Java [Електрон. ресурс]. – Режим доступу:
<https://www.java.com/>
11. Документація Material UI [Електрон. Ресурс] – Режим доступу:
<https://mui.com/material-ui/getting-started/>
12. Google Chrome [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Google_Chrome
13. Mozilla Firefox [Електрон. ресурс]. – Режим доступу:
https://uk.wikipedia.org/wiki/Mozilla_Firefox

14. Microsoft Edge [Електрон. ресурс] – Режим доступу:
https://en.wikipedia.org/wiki/Microsoft_Edge

15. Opera [Електрон. ресурс]. – Режим доступу:
<https://uk.wikipedia.org/wiki/Opera>

16. Safari [Електрон. ресурс]. – Режим
доступу: <https://uk.wikipedia.org/wiki/Safari>

17. Документація PostgreSQL [Електрон. ресурс] – Режим доступу:
<https://www.postgresql.org>



ДОДАТОК А
Текст програми



tests360.service.ts

```

import { Injectable, NotFoundException } from '@nestjs/common';
import { UpdateTest360Dto } from '../dto/update-test360.dto';
import { InjectRepository } from '@nestjs/typeorm';
import { Test360, TestType } from '../entities/tests360.entity';
import { Repository } from 'typeorm';
import { Test360Question } from '../test360-questions/entities/test360-question.entity';
import { User } from 'src/users/entities/user.entity';

@Injectable()
export class Tests360Service {
  constructor(
    @InjectRepository(Test360)
    private test360Repository: Repository<Test360>,
    @InjectRepository(Test360Question)
    private test360QuestionRepository: Repository<Test360Question>,
    @InjectRepository(User)
    private userRepository: Repository<User>
  ) {}

  async create(createTest360Dto: any): Promise<Test360> {
    const { title, creatorUserId, referralCode, testType, questions } =
      createTest360Dto;
    const creatorUser = await this.userRepository.findOneOrFail({
      where: { id: creatorUserId },
    });
    const generatedReferralCode = referralCode || this.generateReferralCode();
    const test360 = this.test360Repository.create({
      title,
      creatorUserId: creatorUser,
      referralCode: generatedReferralCode,
      testType,
    });

    const savedTest360 = await this.test360Repository.save(test360);
    if (questions && questions.length > 0) {
      for (const questionData of questions) {
        const question = this.test360QuestionRepository.create({
          ...questionData,
          testId: test360.id,
        });
        await this.test360QuestionRepository.save(question);
      }
    }

    return savedTest360;
  }

  async findAll(): Promise<Test360[]> {
    return await this.test360Repository.find({ relations: ['questions'] });
  }

  async findAllSelfAssessment(): Promise<Test360[]> {

```

```

return await this.test360Repository.find({
  where: { testType: TestType.SelfAssessment },
  relations: ['questions'],
});
}

async findAllColleaguetoColleague(): Promise<Test360[]> {
  return await this.test360Repository.find({
    where: { testType: TestType.ColleaguetoColleague },
    relations: ['questions'],
  });
}

async findOne(id: number): Promise<Test360> {
  const test = await this.test360Repository.findOne({
    where: { id },
    relations: ['questions'],
  });

  if (!test) {
    throw new NotFoundException(`Test360 with ID "${id}" not found`);
  }

  return test;
}

async update(
  id: number,
  updateTest360Dto: UpdateTest360Dto
): Promise<Test360> {
  const test360 = await this.test360Repository.findOne({ where: { id } });
  if (!test360) {
    throw new NotFoundException(`Test360 with ID "${id}" not found`);
  }

  test360.title = updateTest360Dto.title ?? test360.title;
  test360.testType = updateTest360Dto.testType ?? test360.testType;

  if (updateTest360Dto.creatorUserId) {
    const creatorUser = await this.userRepository.findOneOrFail({
      where: { id: updateTest360Dto.creatorUserId },
    });
    test360.creatorUserId = creatorUser;
  }

  if (
    updateTest360Dto.questions &&
    Array.isArray(updateTest360Dto.questions)
  ) {
    await this.test360QuestionRepository.delete({
      testId: test360,
    });
  }
}

```

```

    for (const questionData of updateTest360Dto.questions) {
      const question = this.test360QuestionRepository.create({
        ...questionData,
        testId: test360,
      });
      await this.test360QuestionRepository.save(question);
    }
  }

  return await this.test360Repository.save(test360);
}

async remove(id: number): Promise<void> {
  const test360 = await this.test360Repository.findOne({ where: { id } });

  if (!test360) {
    throw new NotFoundException(`Test360 with ID "${id}" not found`);
  }

  await this.test360QuestionRepository.delete({
    testId: test360,
  });

  await this.test360Repository.delete(id);
}

private generateReferralCode(): number {
  return Math.floor(100000 + Math.random() * 900000);
}
}

```

tests.service.ts

```

import { Injectable, NotFoundException } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { Test } from '../entities/test.entity';
import { CreateTestDto } from '../dto/create-test.dto';
import { TestQuestion } from '../test-questions/entities/test-question.entity';
import { User } from '../users/entities/user.entity';
import { QuestionAnswer } from '../question-answers/entities/question-answer.entity';
import { UpdateTestDto } from '../dto/update-test.dto';

@Injectable()
export class TestsService {
  constructor(
    @InjectRepository(Test)
    private testRepository: Repository<Test>,
    @InjectRepository(TestQuestion)
    private testQuestionRepository: Repository<TestQuestion>,
    @InjectRepository(QuestionAnswer)

```

```

private questionAnswerRepository: Repository<QuestionAnswer>,
@InjectRepository(User)
private userRepository: Repository<User>
) {}

async create(createTestDto: CreateTestDto): Promise<Test> {
  const { title, creatorUserId, time, questions } = createTestDto;

  const creatorUser = await this.userRepository.findOneOrFail({
    where: { id: creatorUserId },
  });

  const referralCode =
    createTestDto.referralCode || this.generateUniqueReferralCode();

  const test = this.testRepository.create({
    title,
    creatorUserId: creatorUser,
    referralCode,
    time,
    score: 0,
  });

  const savedTest = await this.testRepository.save(test);

  let totalTestScore = 0;

  const questionIds = [];
  for (const questionDto of questions) {
    questionDto.desiredScore = 0;
    const question = this.testQuestionRepository.create({
      ...questionDto,
      test: savedTest,
    });
    const savedQuestion = await this.testQuestionRepository.save(question);
    questionIds.push(savedQuestion.id);
  }

  for (let i = 0; i < questions.length; i++) {
    const questionId = questionIds[i];
    const answers = questions[i].answers;
    for (const answerDto of answers) {
      const answer = this.questionAnswerRepository.create({
        ...answerDto,
        questionId: questionId,
      });
      if (answer.isCorrect) {
        questions[i].desiredScore++;
        await this.testQuestionRepository.update(
          { id: questionId },
          { desiredScore: questions[i].desiredScore }
        );
      }
    }
  }
}

```

```

        await this.questionAnswerRepository.save(answer);
    }
}

questions.forEach((question) => {
    totalTestScore += question.desiredScore;
});
savedTest.score = totalTestScore;
await this.testRepository.save(savedTest);

return savedTest;
}

async findAll(): Promise<Test[]> {
    return await this.testRepository.find({
        relations: ['questions', 'questions.answers'],
    });
}

async findOne(id: number): Promise<Test> {
    const test = await this.testRepository.findOne({
        where: { id },
        relations: ['questions', 'questions.answers'],
    });
    if (!test) {
        throw new NotFoundException(`Test with ID ${id} not found`);
    }
    return test;
}

async findOneAnswer(id: number): Promise<QuestionAnswer> {
    const test = await this.questionAnswerRepository.findOne({
        where: { id },
        relations: ['question'],
    });
    if (!test) {
        throw new NotFoundException(`Test with ID ${id} not found`);
    }
    return test;
}

async findQuestionByNumber(
    testId: number,
    questionNumber: number
): Promise<TestQuestion | undefined> {
    const test = await this.testRepository.findOne({
        where: { id: testId },
        relations: ['questions', 'questions.answers'],
    });
    if (!test) {
        throw new NotFoundException(`Test with ID ${testId} not found`);
    }
}

```

```

return test.questions.find(
  (question) => question.number === questionNumber
);
}

async update(id: number, updateTestDto: UpdateTestDto): Promise<Test> {
  const test = await this.testRepository.findOne({ where: { id } });
  if (!test) {
    throw new NotFoundException(`Test with ID "${id}" not found`);
  }

  test.title = updateTestDto.title ?? test.title;
  test.score = updateTestDto.score ?? test.score;
  test.time = updateTestDto.time ?? test.time;
  test.referralCode = updateTestDto.referralCode ?? test.referralCode;

  if (updateTestDto.questions) {
    const existingQuestions = await this.testQuestionRepository.find({
      where: { test: { id: test.id } },
    });

    for (const question of existingQuestions) {
      await this.questionAnswerRepository.delete({
        questionId: question.id,
      });
    }

    await this.testQuestionRepository.delete({ test: { id: test.id } });

    let totalTestScore = 0;

    const questionIds = [];
    for (const questionDto of updateTestDto.questions) {
      questionDto.desiredScore = 0;
      const question = this.testQuestionRepository.create({
        ...questionDto,
        test: test,
      });
      const savedQuestion = await this.testQuestionRepository.save(question);
      questionIds.push(savedQuestion.id);
    }

    for (let i = 0; i < updateTestDto.questions.length; i++) {
      const questionId = questionIds[i];
      const answers = updateTestDto.questions[i].answers;
      for (const answerDto of answers) {
        const answer = this.questionAnswerRepository.create({
          ...answerDto,
          questionId: questionId,
        });
        if (answer.isCorrect) {
          updateTestDto.questions[i].desiredScore++;
        }
      }
    }
  }
}

```

```

        await this.testQuestionRepository.update(
            { id: questionId },
            { desiredScore: updateTestDto.questions[i].desiredScore }
        );
    }
    await this.questionAnswerRepository.save(answer);
}
}

updateTestDto.questions.forEach((question) => {
    totalTestScore += question.desiredScore;
});
test.score = totalTestScore;
await this.testRepository.update({ id: test.id }, test);

return test;
}

return await this.testRepository.save(test);
}

async remove(id: number): Promise<void> {
    const result = await this.testRepository.delete(id);
    if (result.affected === 0) {
        throw new NotFoundException(`Test with ID ${id} not found`);
    }
}

private generateUniqueReferralCode(): number {
    return Math.floor(Math.random() * 1000000);
}
}

```

test360UserAnswers.service.ts

```

import { Injectable, NotFoundException } from '@nestjs/common';
import { CreateTest360UserAnswerDto } from './dto/create-test360-user-answer.dto';
import { UpdateTest360UserAnswerDto } from './dto/update-test360-user-answer.dto';
import { InjectRepository } from '@nestjs/typeorm';
import { Test360UserAnswer } from './entities/test360-user-answer.entity';
import { Repository } from 'typeorm';

@Injectable()
export class Test360UserAnswersService {
    constructor(
        @InjectRepository(Test360UserAnswer)
        private readonly test360UserAnswerRepo: Repository<Test360UserAnswer>
    ) {}

    async create(
        createDto: CreateTest360UserAnswerDto
    ): Promise<Test360UserAnswer> {

```

```

const newAnswerData = {
  ...createDto,
  task: { id: createDto.taskId },
  user: { id: createDto.userId },
  questionId: { id: createDto.questionId },
  answerId: { id: createDto.answerId },
};

const newAnswer = this.test360UserAnswerRepo.create(newAnswerData);
return this.test360UserAnswerRepo.save(newAnswer);
}

async findAll(): Promise<Test360UserAnswer[]> {
  return this.test360UserAnswerRepo.find();
}

async findOne(id: number): Promise<Test360UserAnswer> {
  const answer = await this.test360UserAnswerRepo.findOne({
    where: { id },
  });
  if (!answer) {
    throw new NotFoundException(`Test360UserAnswer with ID ${id} not found`);
  }
  return answer;
}

async update(
  id: number,
  updateDto: UpdateTest360UserAnswerDto
): Promise<Test360UserAnswer> {
  const answer = await this.findOne(id);
  Object.assign(answer, updateDto);
  return this.test360UserAnswerRepo.save(answer);
}

async remove(id: number): Promise<void> {
  const answer = await this.findOne(id);
  await this.test360UserAnswerRepo.remove(answer);
}
}

```

lessons.service.ts

```

import { Injectable, NotFoundException } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { Lesson } from '../entities/lesson.entity';
import { User } from '../users/entities/user.entity';
import { LessonPage } from '../lesson-pages/entities/lesson-page.entity';
import { CreateLessonsDto } from '../dto/create-lesson.dto';
import { UpdateLessonsDto } from '../dto/update-lesson.dto';
import { Test } from '../tests/entities/test.entity';

```

```

import { TeamsLessons } from 'src/teamsLessons/entities/teamsLessons.entity';
import { Team } from 'src/teams/entities/team.entity';

@Injectable()
export class LessonsService {
  constructor(
    @InjectRepository(Lesson)
    private lessonsRepository: Repository<Lesson>,
    @InjectRepository(LessonPage)
    private lessonPagesRepository: Repository<LessonPage>,
    @InjectRepository(User)
    private userRepository: Repository<User>,
    @InjectRepository(Test)
    private testRepository: Repository<Test>,
    @InjectRepository(TeamsLessons)
    private teamsLessonsRepository: Repository<TeamsLessons>,
    @InjectRepository(Team)
    private teamsRepository: Repository<Team>
  ) {}

  async create(createLessonsDto: CreateLessonsDto): Promise<Lesson> {
    const { teamsLessons, pages, ...lessonData } = createLessonsDto;

    const creatorUser = await this.userRepository.findOne({
      where: { id: lessonData.creatorUserId },
    });

    if (!creatorUser) {
      throw new NotFoundException(
        `Creator user with ID ${lessonData.creatorUserId} not found`
      );
    }

    let test = null;
    if (lessonData.testId) {
      test = await this.testRepository.findOne({
        where: { id: lessonData.testId },
      });

      if (!test) {
        throw new NotFoundException(
          `Test with ID ${lessonData.testId} not found`
        );
      }
    }

    teamsLessons.forEach((team) => {
      this.teamsRepository.findOneByOrFail({
        id: team,
      });
    });

    console.log(test);
  }
}

```

```

const referralCode =
  createLessonsDto.referralCode ||
  (await this.generateUniqueReferralCode());

const lesson = this.lessonsRepository.create({
  ...lessonData,
  referralCode,
  ...(test && { test }),
  creatorUserId: creatorUser,
});

await this.lessonsRepository.save(lesson);

for (const pageDto of pages) {
  const lessonPage = this.lessonPagesRepository.create({
    ...pageDto,
    lessonId: lesson,
  });

  await this.lessonPagesRepository.save(lessonPage);
}

for (const teamDto of teamsLessons) {
  const teamsLessons = this.teamsLessonsRepository.create({
    teamId: { id: teamDto },
    lessonId: { id: lesson.id },
  });

  await this.teamsLessonsRepository.save(teamsLessons);
}

return lesson;
}

async findAll(): Promise<Lesson[]> {
  return await this.lessonsRepository.find({
    relations: ['pages', 'test'],
  });
}

async findAllByTeamId(
  teamId: number,
  searchQuery?: string,
  skip?: number,
  take?: number
): Promise<[Lesson[], number]> {
  if (isNaN(teamId)) {
    console.error('teamId is NaN');
    throw new Error('Invalid teamId');
  }

  this.teamsRepository.findOneByOrFail({ id: teamId });

```

```

let query = this.lessonsRepository
  .createQueryBuilder('lesson')
  .leftJoinAndSelect('lesson.teamsLessons', 'teamsLessons')
  .leftJoinAndSelect('lesson.pages', 'pages')
  .leftJoinAndSelect('lesson.test', 'test')
  .where('teamsLessons.teamId = :teamId', { teamId });

if (searchQuery) {
  query = query.andWhere('lesson.title ILIKE :searchQuery', {
    searchQuery: `%${searchQuery}%`,
  });
}

if (skip !== undefined) {
  query = query.skip(skip);
}

if (take !== undefined) {
  query = query.take(take);
}

return await query.getManyAndCount();
}

async findAllWithoutTeams(
  searchQuery?: string,
  skip?: number,
  take?: number
): Promise<[Lesson[], number]> {
  let query = this.lessonsRepository
    .createQueryBuilder('lesson')
    .leftJoin('lesson.teamsLessons', 'teamsLessons')
    .leftJoinAndSelect('lesson.pages', 'pages')
    .leftJoinAndSelect('lesson.test', 'test')
    .where('teamsLessons.id IS NULL');

  if (searchQuery) {
    query = query.andWhere('lesson.title ILIKE :searchQuery', {
      searchQuery: `%${searchQuery}%`,
    });
  }

  if (skip !== undefined) {
    query = query.skip(skip);
  }

  if (take !== undefined) {
    query = query.take(take);
  }

  return await query.getManyAndCount();
}

```

```

async findOne(id: number): Promise<Lesson> {
  const lesson = await this.lessonsRepository.findOne({
    where: { id },
    relations: ['pages', 'test', 'teamsLessons', 'teamsLessons.teamId'],
  });

  if (!lesson) {
    throw new NotFoundException(`Lesson with ID ${id} not found`);
  }

  return lesson;
}

async findLessonPageByNumber(
  lessonId: number,
  pageNumber: number
): Promise<LessonPage | undefined> {
  const lesson = await this.lessonsRepository.findOne({
    where: { id: lessonId },
    relations: ['pages', 'teamsLessons'],
  });

  if (!lesson) {
    throw new NotFoundException(`Lesson with ID ${lessonId} not found`);
  }

  const lessonPage = lesson.pages.find((page) => page.number === pageNumber);

  if (!lessonPage) {
    throw new NotFoundException(
      `LessonPage with number ${pageNumber} not found in lesson ID ${lessonId}`
    );
  }

  return lessonPage;
}

async update(
  id: number,
  updateLessonsDto: UpdateLessonsDto
): Promise<Lesson> {
  const lesson = await this.lessonsRepository.findOne({
    where: { id },
    relations: ['pages', 'teamsLessons'],
  });

  if (!lesson) {
    throw new NotFoundException(`Lesson with ID ${id} not found`);
  }

  const test = await this.testRepository.findOne({
    where: { id: updateLessonsDto.testId },
  });

```

```

    });

    console.log(test);

    if (!test) {
        throw new NotFoundException(
            `Test with ID ${updateLessonsDto.testId} not found`
        );
    }

    lesson.test = test;

    await this.lessonPagesRepository.delete({ lessonId: lesson });
    await this.teamsLessonsRepository.delete({ lessonId: lesson });

    Object.assign(lesson, updateLessonsDto);
    await this.lessonsRepository.save(lesson);

    for (const pageDto of updateLessonsDto.pages) {
        const lessonPage = this.lessonPagesRepository.create({
            ...pageDto,
            lessonId: { id: lesson.id },
        });

        await this.lessonPagesRepository.save(lessonPage);
    }

    for (const teamDto of updateLessonsDto.teamsLessons) {
        const teamsLessons = this.teamsLessonsRepository.create({
            teamId: { id: teamDto },
            lessonId: { id: lesson.id },
        });

        await this.teamsLessonsRepository.save(teamsLessons);
    }

    return lesson;
}

async remove(id: number): Promise<void> {
    const result = await this.lessonsRepository.delete(id);

    if (result.affected === 0) {
        throw new NotFoundException(`Lesson with ID ${id} not found`);
    }
}

private async generateUniqueReferralCode(): Promise<number> {
    const existingCodes = new Set();
    const lessons = await this.lessonsRepository.find({
        select: ['referralCode'],
    });
    lessons.forEach((lesson) => existingCodes.add(lesson.referralCode));
}

```

```

let potentialCode: number;
do {
  potentialCode = Math.floor(Math.random() * 1000000);
} while (existingCodes.has(potentialCode));

return potentialCode;
}
}

```

company360.service.ts

```

import { Injectable, NotFoundException } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { In, Repository } from 'typeorm';
import { Company360 } from '../entities/company360.entity';
import {
  CreateCompany360Dto,
  CreateCompany360TaskDto,
} from '../dto/create-company360.dto';
import {
  UpdateCompany360Dto,
  UpdateCompany360TaskDto,
} from '../dto/update-company360.dto';
import { Company360Task } from '../company360-tasks/entities/company360-task.entity';
import { User } from '../users/entities/user.entity';
import { Test360 } from '../tests360/entities/tests360.entity';
import { TeamMember } from '../team-members/entities/team-member.entity';
import { Team } from '../teams/entities/team.entity';
import { Test360UserAnswer } from 'src/test360-user-answers/entities/test360-user-answer.entity';

@Injectable()
export class Company360Service {
  constructor(
    @InjectRepository(Company360)
    private company360Repository: Repository<Company360>,
    @InjectRepository(Company360Task)
    private company360TaskRepository: Repository<Company360Task>,
    @InjectRepository(User)
    private userRepository: Repository<User>,
    @InjectRepository(Test360)
    private test360Repository: Repository<Test360>,
    @InjectRepository(TeamMember)
    private teamMemberRepository: Repository<TeamMember>,
    @InjectRepository(Team)
    private teamRepository: Repository<Team>,
    @InjectRepository(Test360UserAnswer)
    private test360UserAnswerRepository: Repository<Test360UserAnswer>
  ) {}

  async create(createCompany360Dto: CreateCompany360Dto): Promise<Company360> {
    const creatorUser = await this.userRepository.findOne({

```

```

    where: { id: createCompany360Dto.creatorUserId },
  });
  if (!creatorUser) {
    throw new NotFoundException(
      `Creator user with ID ${createCompany360Dto.creatorUserId} not found`
    );
  }

  const selfAssessmentTest = await this.test360Repository.findOne({
    where: { id: createCompany360Dto.selfAssessmentTestId },
  });
  const otherAssessmentTest = await this.test360Repository.findOne({
    where: { id: createCompany360Dto.otherAssessmentTestId },
  });

  const company360 = this.company360Repository.create({
    name: createCompany360Dto.name,
    selfAssessmentTest,
    otherAssessmentTest,
    creatorUser,
    teams: createCompany360Dto.teams,
  });

  await this.company360Repository.save(company360);

  await this.createSelfAssessmentTasks(company360, createCompany360Dto.teams);
  await this.createColleagueAssessmentTasks(
    company360,
    createCompany360Dto.teams
  );
  await this.createSubordinateManagerTasks(
    company360,
    createCompany360Dto.teams
  );
  await this.createManagerSubordinateTasks(
    company360,
    createCompany360Dto.teams
  );

  return company360;
}

async createTask(
  createTaskDto: CreateCompany360TaskDto,
  company360: Company360
): Promise<Company360Task> {
  const evaluatorUser = await this.userRepository.findOne({
    where: { id: createTaskDto.evaluatorUserId },
  });
  const evaluateeUser = await this.userRepository.findOne({
    where: { id: createTaskDto.evaluateeUserId },
  });
  const test360 = await this.test360Repository.findOne({

```

```

    where: { id: createTaskDto.test360Id },
  });

  if (!evaluatorUser || !evaluateeUser || !test360) {
    throw new NotFoundException('One or more entities not found');
  }

  const task = this.company360TaskRepository.create({
    evaluatorUser,
    evaluateeUser,
    test360,
    company360,
    completed: false,
  });

  return this.company360TaskRepository.save(task);
}

async findAll(): Promise<Company360[]> {
  return await this.company360Repository.find({
    relations: ['tasks'],
  });
}

async findOne(id: number): Promise<Company360> {
  const company = await this.company360Repository.findOne({
    where: { id },
    relations: ['tasks', 'selfAssessmentTest', 'otherAssessmentTest'],
  });

  if (!company) {
    throw new NotFoundException(`Company360 with ID ${id} not found`);
  }

  return company;
}

async findAllTasksByCompanyId(companyId: number): Promise<Company360Task[]> {
  const company = await this.company360Repository.findOne({
    where: { id: companyId },
  });

  if (!company) {
    throw new NotFoundException(`Company360 with ID ${companyId} not found`);
  }

  return await this.company360TaskRepository.find({
    where: { company360: { id: companyId } },
    relations: ['evaluatorUser', 'evaluateeUser', 'test360', 'company360'],
  });
}

async findTasksByUserId(userId: number): Promise<Company360Task[]> {
  return this.company360TaskRepository.find({

```

```

    where: { evaluatorUser: { id: userId }, completed: false },
    relations: ['company360'],
  });
}

async findTaskById(id: number): Promise<Company360Task> {
  const task = await this.company360TaskRepository.findOne({
    where: { id },
    relations: ['evaluatorUser', 'evaluateeUser', 'test360', 'company360'],
  });

  if (!task) {
    throw new NotFoundException(`Task with ID ${id} not found`);
  }

  return task;
}

async getUserAssessmentAverages(userId: number): Promise<any> {
  const tasks = await this.company360TaskRepository.find({
    where: [
      { evaluatorUser: { id: userId } },
      { evaluateeUser: { id: userId } },
    ],
    relations: ['test360', 'evaluatorUser', 'evaluateeUser', 'company360'],
  });

  let selfAssessmentScores = {};
  let colleagueAssessmentScores = {};
  let subordinateAssessmentScores = {};
  let managerAssessmentScores = {};

  for (const task of tasks) {
    const detailedTeams = await this.teamRepository.find({
      where: { id: In(task.company360.teams) },
      relations: ['manager', 'teamMembers', 'childTeams', 'parentTeam'],
    });

    const evaluator = await this.userRepository.findOne({
      where: { id: task.evaluatorUser.id },
    });
    const evaluatee = await this.userRepository.findOne({
      where: { id: task.evaluateeUser.id },
    });

    if (!evaluator || !evaluatee) {
      return;
    }

    const answers = await this.test360UserAnswerRepository.find({
      where: { task: { id: task.id } },
      relations: ['questionId', 'answerId'],
    });
  }
}

```

```

for (const answer of answers) {
  if (
    !answer.answerId ||
    !answer.questionId ||
    !answer.questionId.questionType
  ) {
    continue;
  }

  const questionType = answer.questionId.questionType;
  const score = answer.answerId.score;

  if (task.evaluateeUser.id === userId) {
    if (task.evaluatorUser.id === userId) {
      selfAssessmentScores = this.updateScores(
        selfAssessmentScores,
        questionType,
        score
      );
    } else {
      const relationship = this.determineRelationship(
        evaluator,
        evaluatee,
        detailedTeams
      );

      switch (relationship) {
        case 'manager':
          managerAssessmentScores = this.updateScores(
            managerAssessmentScores,
            questionType,
            score
          );

          break;
        case 'subordinate':
          subordinateAssessmentScores = this.updateScores(
            subordinateAssessmentScores,
            questionType,
            score
          );

          break;
        case 'colleague':
          colleagueAssessmentScores = this.updateScores(
            colleagueAssessmentScores,
            questionType,
            score
          );

          break;
      }
    }
  }
}

```

```

    }
  }
}

return {
  selfAssessment: this.calculateAverages(selfAssessmentScores),
  colleagueAssessment: this.calculateAverages(colleagueAssessmentScores),
  subordinateAssessment: this.calculateAverages(
    subordinateAssessmentScores
  ),
  managerAssessment: this.calculateAverages(managerAssessmentScores),
};
}

private determineRelationship(
  evaluator: User,
  evaluatee: User,
  teams: Team[]
): string {
  if (evaluator.id === evaluatee.id) {
    return 'self';
  }

  for (const team of teams) {
    const isEvaluatorManager = team.manager?.id === evaluator.id;
    const isEvaluateeManager = team.manager?.id === evaluatee.id;
    const isEvaluatorTeamMember = true;
    const isEvaluateeTeamMember = true;

    if (isEvaluatorManager && isEvaluateeTeamMember) {
      return 'manager';
    } else if (isEvaluateeManager && isEvaluatorTeamMember) {
      return 'subordinate';
    } else if (isEvaluatorTeamMember && isEvaluateeTeamMember) {
      return 'colleague';
    } else if (team.parentTeam) {
      const isEvaluatorParentManager =
        team.parentTeam.manager?.id === evaluator.id;
      const isEvaluateeChildManager = team.childTeams?.some(
        (childTeam) => childTeam.manager?.id === evaluatee.id
      );

      if (isEvaluatorParentManager && isEvaluateeChildManager) {
        return 'manager';
      } else if (
        isEvaluateeManager &&
        team.parentTeam.manager?.id === evaluator.id
      ) {
        return 'subordinate';
      }
    }
  }
}

```

```

    return 'unknown';
  }

  private updateScores(scores: any, questionType: string, score: number): any {
    if (!scores[questionType]) {
      scores[questionType] = { totalScore: 0, count: 0 };
    }
    scores[questionType].totalScore += score;
    scores[questionType].count++;
    return scores;
  }

  private calculateAverages(scores: any): any {
    const averages = {};
    Object.keys(scores).forEach((questionType) => {
      averages[questionType] =
        scores[questionType].totalScore / scores[questionType].count;
    });
    return averages;
  }

  async update(
    id: number,
    updateCompany360Dto: UpdateCompany360Dto
  ): Promise<Company360> {
    const company360 = await this.company360Repository.findOne({
      where: { id },
      relations: ['tasks'],
    });

    if (!company360) {
      throw new NotFoundException(`Company360 with ID ${id} not found`);
    }

    company360.name = updateCompany360Dto.name || company360.name;

    if (updateCompany360Dto.selfAssessmentTestId) {
      const selfAssessmentTest = await this.test360Repository.findOne({
        where: { id: updateCompany360Dto.selfAssessmentTestId },
      });
      if (selfAssessmentTest) {
        company360.selfAssessmentTest = selfAssessmentTest;
      }
    }

    if (updateCompany360Dto.otherAssessmentTestId) {
      const otherAssessmentTest = await this.test360Repository.findOne({
        where: { id: updateCompany360Dto.otherAssessmentTestId },
      });
      if (otherAssessmentTest) {
        company360.otherAssessmentTest = otherAssessmentTest;
      }
    }
  }

```

```

if (updateCompany360Dto.teams) {
  const updatedTeams = await this.teamRepository.findByIds(
    updateCompany360Dto.teams
  );

  const updatedTeamIds = updatedTeams.map((team) => team.id);

  company360.teams = updatedTeamIds;
}

await this.company360Repository.save(company360);

await this.company360TaskRepository.delete({
  company360: { id: company360.id },
});

await this.createSelfAssessmentTasks(company360, updateCompany360Dto.teams);
await this.createColleagueAssessmentTasks(
  company360,
  updateCompany360Dto.teams
);
await this.createSubordinateManagerTasks(
  company360,
  updateCompany360Dto.teams
);
await this.createManagerSubordinateTasks(
  company360,
  updateCompany360Dto.teams
);

company360.teams = Array.from(new Set(updateCompany360Dto.teams));

return company360;
}

async updateTask(
  id: number,
  updateTaskDto: UpdateCompany360TaskDto
): Promise<Company360Task> {
  const task = await this.company360TaskRepository.findOne({ where: { id } });

  if (!task) {
    throw new NotFoundException(`Task with ID ${id} not found`);
  }

  if (updateTaskDto.evaluatorUserId)
    task.evaluatorUser = await this.userRepository.findOne({
      where: { id: updateTaskDto.evaluatorUserId },
    });
  if (updateTaskDto.evaluateeUserId)
    task.evaluateeUser = await this.userRepository.findOne({
      where: { id: updateTaskDto.evaluateeUserId },
    });
}

```

```

    });
    if (updateTaskDto.test360Id)
        task.test360 = await this.test360Repository.findOne({
            where: { id: updateTaskDto.test360Id },
        });
    if (updateTaskDto.completed !== undefined) {
        task.completed = updateTaskDto.completed;
    }

    return this.company360TaskRepository.save(task);
}

async delete(id: number): Promise<void> {
    const company360 = await this.company360Repository.findOne({
        where: { id },
        relations: ['tasks'],
    });

    if (!company360) {
        throw new NotFoundException(`Company360 with ID ${id} not found`);
    }

    for (const task of company360.tasks) {
        const answersCount = await this.test360UserAnswerRepository.count({
            where: { task: { id: task.id } },
        });

        if (answersCount > 0) {
            throw new Error(
                `Cannot delete Company360 with ID ${id} as there are existing answers for its tasks`
            );
        }
    }

    for (const task of company360.tasks) {
        await this.company360TaskRepository.delete({ id: task.id });
    }

    await this.company360Repository.delete({ id: company360.id });
}

async deleteTask(id: number): Promise<void> {
    const result = await this.company360TaskRepository.delete({ id });

    if (result.affected === 0) {
        throw new NotFoundException(`Task with ID ${id} not found`);
    }
}

async activate(id: number): Promise<Company360> {
    const company = await this.company360Repository.findOne({ where: { id } });
    if (!company) {
        throw new NotFoundException(`Company360 with ID ${id} not found`);
    }
}

```

```

    }
    company.active = true;
    return this.company360Repository.save(company);
  }

  async deactivate(id: number): Promise<Company360> {
    const company = await this.company360Repository.findOne({ where: { id } });
    if (!company) {
      throw new NotFoundException(`Company360 with ID ${id} not found`);
    }
    company.active = false;
    return this.company360Repository.save(company);
  }

  async completeTask(taskId: number): Promise<Company360Task> {
    const task = await this.company360TaskRepository.findOne({
      where: { id: taskId },
      relations: ['company360'],
    });

    if (!task) {
      throw new NotFoundException(`Task with ID ${taskId} not found`);
    }

    task.completed = true;
    const updatedTask = await this.company360TaskRepository.save(task);

    await this.checkAndDeactivateCompanyIfTasksCompleted(task.company360.id);

    return updatedTask;
  }

  async checkAndDeactivateCompanyIfTasksCompleted(
    companyId: number
  ): Promise<void> {
    const company = await this.company360Repository.findOne({
      where: { id: companyId },
      relations: ['tasks'],
    });

    if (!company) {
      throw new NotFoundException(`Company360 with ID ${companyId} not found`);
    }

    const allTasksCompleted = company.tasks.every((task) => task.completed);

    if (allTasksCompleted) {
      company.active = false;
      await this.company360Repository.save(company);
    }
  }

  private async createSelfAssessmentTasks(

```

```

company360: Company360,
teamIds: number[]
): Promise<void> {
  for (const teamId of teamIds) {
    const team = await this.teamRepository.findOne({
      where: { id: teamId },
      relations: ['manager', 'teamMembers', 'teamMembers.user'],
    });

    if (!team) {
      continue;
    }

    if (team.manager) {
      const existingManagerTask = await this.company360TaskRepository.findOne(
        {
          where: {
            evaluatorUser: { id: team.manager.id },
            evaluateeUser: { id: team.manager.id },
            company360: { id: company360.id },
          },
        }
      );

      if (!existingManagerTask) {
        const managerTask = this.company360TaskRepository.create({
          evaluatorUser: { id: team.manager.id },
          evaluateeUser: { id: team.manager.id },
          test360: company360.selfAssessmentTest,
          company360,
        });

        await this.company360TaskRepository.save(managerTask);
      }
    }

    for (const member of team.teamMembers) {
      if (team.manager && member.user.id === team.manager.id) {
        continue;
      }

      const existingTask = await this.company360TaskRepository.findOne({
        where: {
          evaluatorUser: member.user,
          evaluateeUser: member.user,
          company360: { id: company360.id },
        },
      });

      if (!existingTask) {
        const task = this.company360TaskRepository.create({
          evaluatorUser: member.user,
          evaluateeUser: member.user,

```

```

        test360: company360.selfAssessmentTest,
        company360,
    });

    await this.company360TaskRepository.save(task);
  }
}
}
}

private async createColleagueAssessmentTasks(
  company360: Company360,
  teamIds: number[]
): Promise<void> {
  const teamsWithManagers = await Promise.all(
    teamIds.map(async (teamId) => {
      const team = await this.teamRepository.findOne({
        where: { id: teamId },
        relations: ['manager', 'parentTeam'],
      });
      return team;
    })
  );

  for (let i = 0; i < teamsWithManagers.length; i++) {
    for (let j = 0; j < teamsWithManagers.length; j++) {
      if (
        i !== j &&
        teamsWithManagers[i]?.parentTeam?.id ===
          teamsWithManagers[j]?.parentTeam?.id &&
        teamsWithManagers[i]?.manager &&
        teamsWithManagers[j]?.manager
      ) {
        const taskForManagerI = this.company360TaskRepository.create({
          evaluatorUser: teamsWithManagers[i].manager,
          evaluateeUser: teamsWithManagers[j].manager,
          test360: company360.otherAssessmentTest,
          company360,
        });

        await this.company360TaskRepository.save(taskForManagerI);
      }
    }
  }

  const teamMembers = await this.teamMemberRepository.find({
    where: { team: { id: teamsWithManagers[i].id } },
    relations: ['user'],
  });

  for (let k = 0; k < teamMembers.length; k++) {
    for (let l = 0; l < teamMembers.length; l++) {
      if (k !== l) {
        const task = this.company360TaskRepository.create({

```

```

        evaluatorUser: teamMembers[k].user,
        evaluateeUser: teamMembers[l].user,
        test360: company360.otherAssessmentTest,
        company360,
    });

    await this.company360TaskRepository.save(task);
  }
}
}
}
}

private async createSubordinateManagerTasks(
  company360: Company360,
  teamIds: number[]
): Promise<void> {
  for (const teamId of teamIds) {
    const team = await this.teamRepository.findOne({
      where: { id: teamId },
      relations: [
        'manager',
        'teamMembers',
        'parentTeam',
        'parentTeam.manager',
      ],
    });

    if (team) {
      if (team.manager) {
        const teamMembers = await this.teamMemberRepository.find({
          where: { team: { id: teamId } },
          relations: ['user'],
        });

        for (const member of teamMembers) {
          const task = this.company360TaskRepository.create({
            evaluatorUser: member.user,
            evaluateeUser: team.manager,
            test360: company360.otherAssessmentTest,
            company360,
          });

          await this.company360TaskRepository.save(task);
        }
      }
    }

    if (team.parentTeam && team.parentTeam.manager && team.manager) {
      const task = this.company360TaskRepository.create({
        evaluatorUser: team.manager,
        evaluateeUser: team.parentTeam.manager,
        test360: company360.otherAssessmentTest,
        company360,
      });
    }
  }
}

```

```

    });

    const task2 = this.company360TaskRepository.create({
      evaluatorUser: team.parentTeam.manager,
      evaluateeUser: team.manager,
      test360: company360.otherAssessmentTest,
      company360,
    });

    await this.company360TaskRepository.save(task);
    await this.company360TaskRepository.save(task2);
  }
}
}
}

private async createManagerSubordinateTasks(
  company360: Company360,
  teamIds: number[]
): Promise<void> {
  for (const teamId of teamIds) {
    const team = await this.teamRepository.findOne({
      where: { id: teamId },
      relations: ['manager', 'teamMembers', 'childTeams'],
    });

    if (team && team.manager) {
      const teamMembers = await this.teamMemberRepository.find({
        where: { team: { id: teamId } },
        relations: ['user'],
      });

      for (const member of teamMembers) {
        const task = this.company360TaskRepository.create({
          evaluatorUser: team.manager,
          evaluateeUser: member.user,
          test360: company360.otherAssessmentTest,
          company360,
        });

        await this.company360TaskRepository.save(task);
      }

      for (const childTeam of team.childTeams) {
        if (childTeam.manager) {
          const task = this.company360TaskRepository.create({
            evaluatorUser: team.manager,
            evaluateeUser: childTeam.manager,
            test360: company360.otherAssessmentTest,
            company360,
          });

          await this.company360TaskRepository.save(task);
        }
      }
    }
  }
}

```

```

    }
  }
}
}
}
}
}

```

lessonPage.tsx

```

import React, { useEffect, useState } from 'react';
import { Typography, Button } from '@mui/material';
import axios, { isAxiosError } from 'axios';
import { useNavigate, useParams } from 'react-router-dom';
import { jwtDecode } from 'jwt-decode';
import { QueryClient, useMutation } from 'react-query';
import { PageProps } from '../../interfaces/page';
import usePageMetadataStore from '../../stores/usePageMetadataStore';
import backendUrl from '../../config';
import useLessonStore from '../../stores/lessonStore';

const Page: React.FC<PageProps> = ({ name, description }) => {
  const queryClient = new QueryClient();
  const navigate = useNavigate();
  const token = localStorage.getItem('token');

  const { lessonId, pageId } = useParams<{ lessonId: string; pageId: string }>();
  const [pageIndex, setPageIndex] = useState<number>(
    parseInt(pageId || '0', 10) - 1
  );

  const { lesson, setLesson } = useLessonStore();
  const userId = token ? jwtDecode<{ id: number }>(token).id : null;

  const setName = usePageMetadataStore((state) => state.setName);
  const setDescription = usePageMetadataStore((state) => state.setDescription);
  const setButton = usePageMetadataStore((state) => state.setButton);

  useEffect(() => {
    setName(name);
    setDescription(description);
    setButton(null);
  }, [setName, setDescription, setButton, name, description]);

  useEffect(() => {
    const fetchLesson = async () => {
      if (!lesson || !lesson.pages || Number.isNaN(pageIndex)) {
        try {
          const response = await axios.get(`${backendUrl}/lessons/${lessonId}`);
          setLesson(response.data);
        } catch (error) {
          console.error('Error fetching lesson:', error);
        }
      }
    };
  });

```

```

    }
  };

  fetchLesson();
}, [lesson, pageIndex, lessonId]);

const userTestsMutation = useMutation(
  (newUserTest: { userId: number; testId: number }) => {
    return axios.post(`${backendUrl}/users-tests`, newUserTest);
  },
  {
    onSuccess: () => {
      queryClient.invalidateQueries('uncompletedTests');
      navigate('/tests');
    },
    onError: (error: unknown) => {
      let message = 'An error occurred while creating the test.';
      if (isAxiosError(error)) {
        message = error.response?.data?.message || message;
        console.log(message);
      }
    },
  }
);

console.log(pageIndex);

if (!lesson || !lesson.pages || Number.isNaN(pageIndex)) {
  return (
    <Typography style={{ textAlign: 'center' }} variant="h6">
      Сторінка не знайдена або помилка завантаження лекції
    </Typography>
  );
}

return (
  <div style={{ display: 'flex', flexDirection: 'column' }}>
    <Typography variant="h4" style={{ marginBottom: '16px' }}>
      {lesson.pages[pageIndex].title}
    </Typography>
    <div style={{ flexGrow: 1, overflow: 'hidden', width: '100%' }}>
      <iframe
        src={lesson.pages[pageIndex].videoUrl}
        title="Unique title here"
        width="100%"
        height="100%"
        allow="autoplay"
        frameBorder="none"
      />
    </div>
  </div>
  <div
    style={{
      marginBottom: '1rem',

```

```

    marginTop: '1rem',
    display: 'flex',
    justifyContent: 'space-between',
  }}
>
<Button
  variant="contained"
  onClick={() => {
    const newIndex = pageIndex - 1;
    setPageIndex(newIndex);
    navigate(`/lesson/${lesson.id}/${newIndex + 1}`);
  }}
  disabled={pageIndex === 0}
>
  Назад
</Button>
{pageIndex === lesson.pages.length - 1 && lesson.test ? (
  <Button
    onClick={() => {
      if (userId && lesson.test?.id) {
        userTestsMutation.mutate({
          userId,
          testId: lesson.test.id,
        });
      }
    }}
  >
    Пройти Тестування
  </Button>
) : null}
<Button
  variant="contained"
  onClick={() => {
    const newIndex = pageIndex + 1;
    setPageIndex(newIndex);
    navigate(`/lesson/${lesson.id}/${newIndex + 1}`);
  }}
  disabled={pageIndex === lesson.pages.length - 1}
>
  Вперед
</Button>
</div>
</div>
);
};

export default Page;

```

testTasks.tsx

```

import React, { useEffect, useState } from 'react';
import {

```

```

Grid,
Paper,
Typography,
CircularProgress,
Tooltip,
} from '@mui/material';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import { jwtDecode } from 'jwt-decode';
import { PageProps } from '../interfaces/page';
import usePageMetadataStore from '../stores/usePageMetadataStore';
import backendUrl from '../config';
import { IUsersTest } from '../interfaces/responses';
import useUserTestStore from '../stores/userTestStore';

const TestsTasks: React.FC<PageProps> = ({ name, description }) => {
  const token = localStorage.getItem('token');
  const navigate = useNavigate();

  const { setUserTest } = useUserTestStore();
  const [fetchedUncompletedTests, setFetchedUncompletedTests] = useState<IUsersTest[]>([]);
  const [isLoading, setIsLoading] = useState<boolean>(true);
  const [error, setError] = useState<string | null>(null);

  const setName = usePageMetadataStore((state) => state.setName);
  const setDescription = usePageMetadataStore((state) => state.setDescription);
  const setButton = usePageMetadataStore((state) => state.setButton);

  useEffect(() => {
    setName(name);
    setDescription(description);
    setButton(null);
  }, [setName, setDescription, setButton, name, description]);

  const userId = token ? jwtDecode<{ id: number }>(token).id : null;
  console.log('User ID:', userId); // Додано логування

  const fetchUncompletedTests = async () => {
    try {
      const { data: allTests } = await axios.get(`${backendUrl}/users-tests`);
      console.log('API Response:', allTests); // Додано логування
      const uncompletedTests = allTests.filter((test: IUsersTest) => !test.isCompleted);
      console.log('Uncompleted tests:', uncompletedTests); // Додано логування
      setFetchedUncompletedTests(uncompletedTests);
    } catch (error) {
      console.error('Error fetching tests:', error);
      setError('Відбулася помилка під час завантаження завдань.');
```

```

if (userId) {
  fetchUncompletedTests();
} else {
  setIsLoading(false);
  setError('User ID is null');
}
}, [userId]);

useEffect(() => {
  console.log('Fetched uncompleted tests:', fetchedUncompletedTests); // Додано логування
}, [fetchedUncompletedTests]);

if (isLoading) {
  return (
    <div>
      <CircularProgress />
    </div>
  );
}

if (error) {
  return (
    <Typography
      style={{
        height: '100%',
        width: '100%',
        textAlign: 'center',
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center',
      }}
      variant="h6"
    >
      {error}
    </Typography>
  );
}

const navigateToTest = (url: string, userTest: IUsersTest) => {
  setUserTest(userTest);
  navigate(url);
};

return (
  <Grid container spacing={4} height="fit-content">
    {fetchedUncompletedTests.length > 0 ? (
      fetchedUncompletedTests.map((userTest: IUsersTest) => {
        const lastAttemptDate = userTest.lastAttemptDate
          ? new Date(userTest.lastAttemptDate)
          : null;
        const penultimateAttemptDate = userTest.penultimateAttemptDate
          ? new Date(userTest.penultimateAttemptDate)
          : null;

```

```

const now = new Date();

const hoursSinceLastAttempt = lastAttemptDate
  ? Math.abs(now.getTime() - lastAttemptDate.getTime()) / (1000 * 60 * 60)
  : null;

const penultimateAttemptAttempt = penultimateAttemptDate
  ? Math.abs(now.getTime() - penultimateAttemptDate.getTime()) /
    (1000 * 60 * 60)
  : null;

const canAttemptTest =
  (hoursSinceLastAttempt ?? 25) >= 24 ||
  (penultimateAttemptAttempt ?? 25) >= 24;

const handleClick = (uTest: IUsersTest) => {
  if (canAttemptTest) {
    setUserTest(uTest);
    navigateToTest(`/tests/${uTest.testId?.id}`, uTest);
  } else {
    console.log("Test cannot be attempted yet.");
  }
};

return (
  <Grid item key={userTest.id} xs={12} sm={6} md={4} lg={3}>
    <Tooltip
      title={
        !canAttemptTest
          ? 'Тест не можна пройти. Потрібно почекати 24 години з моменту останньої спроби.'
          : ''
        }
      followCursor
    >
    <Paper
      elevation={3}
      onClick={() => handleClick(userTest)}
      style={{ textDecoration: 'none' }}
      sx={{ ':hover': { cursor: 'pointer' } }}
    >
      <Typography variant="h6">{userTest.testId?.title}</Typography>
      <Typography color="textSecondary">
        Питань: {userTest?.testId?.questions?.length}
      </Typography>
      <Typography color="textSecondary">
        Прогідний бал: (((userTest?.testId?.score ?? 0) / 1.5).toFixed(0)) / '{ ' }
        {userTest?.testId?.score ?? 'N/A'}
      </Typography>
      <Typography color="textSecondary">
        Кількість спроб: {userTest.attempts}
      </Typography>
      <Typography color="textSecondary">
        Час: {userTest.testId?.time} хв.

```

```

    </Typography>
    {userTest.lastAttemptDate && (
      <Typography color="textSecondary">
        Остання спроба: {lastAttemptDate?.toLocaleDateString()}
      </Typography>
    )}
  </Paper>
</Tooltip>
</Grid>
);
})
): (
  <Typography variant="h6">Немає тестів для відображення.</Typography>
)
</Grid>
);
};

export default TestsTasks;

```

test360Companies.tsx

```

import React, { useEffect } from 'react';
import { Grid, Paper, Typography, CircularProgress } from '@mui/material';
import { useQuery } from 'react-query';
import axios from 'axios';
import { Link } from 'react-router-dom';
import { jwtDecode } from 'jwt-decode';
import { TCompany360, TCompany360Task } from '../interfaces/types';
import { PageProps } from '../interfaces/page';
import usePageMetadataStore from '../stores/usePageMetadataStore';
import backendUrl from '../config';

const ActiveCompanies360: React.FC<PageProps> = ({ name, description }) => {
  const token = localStorage.getItem('token');

  const setName = usePageMetadataStore((state) => state.setName);
  const setDescription = usePageMetadataStore((state) => state.setDescription);
  const setButton = usePageMetadataStore((state) => state.setButton);

  useEffect(() => {
    setName(name);
    setDescription(description);
    setButton(null);
  }, [setName, setDescription, setButton, name, description]);

  const userId = token ? jwtDecode<{ id: number }>(token).id : null;

  const {
    data: teamId,
    isLoading: isTeamIdLoading,
    error: teamIdError,
  }

```

```

} = useQuery(
  ['teamId', userId],
  () =>
    axios
      .get(`${backendUrl}/teams/team-and-manager/${userId}`)
      .then((res) => res.data.team.id),
  { enabled: !!userId }
);

const { data: userTasks, isLoading: isUserTasksLoading } = useQuery(
  ['userTasks', userId],
  () =>
    axios
      .get(`${backendUrl}/companies360/user-tasks/${userId}`)
      .then((res) => res.data),
  { enabled: !!userId }
);

const fetchActiveCompanies = async (tId: number) => {
  const { data: allCompanies } = await axios.get(`${backendUrl}/companies360`);
  return allCompanies.filter(
    (company: TCompany360) => company.active && company.teams.includes(tId)
  );
};

const {
  data: companies,
  isLoading: isCompaniesLoading,
  error: companiesError,
} = useQuery(
  ['activeCompanies360', teamId],
  () => fetchActiveCompanies(teamId),
  { enabled: !!teamId }
);

// Combine team companies and task-related companies
const combinedCompanies = React.useMemo(() => {
  if (!companies || !userTasks) return [];

  // Filter out tasks that are not completed
  const incompleteTasks = userTasks.filter(
    (task: TCompany360Task) => !task.completed
  );

  // Get unique company IDs from incomplete tasks
  const taskCompanyIds = incompleteTasks.map(
    (task: TCompany360Task) => task.company360.id
  );

  // Combine and filter companies
  const activeCompanies = companies.filter(
    (company: TCompany360) =>
      company.active &&

```

```

    (company.teams.includes(teamId) || taskCompanyIds.includes(company.id))
  );

  // Filter out companies where all tasks are completed
  return activeCompanies.filter((company: TCompany360) =>
    userTasks.some(
      (task: TCompany360Task) =>
        task.company360.id === company.id && !task.completed
    )
  );
}, [companies, userTasks, teamId]);

if (isTeamIdLoading || isCompaniesLoading || isUserTasksLoading)
  return (
    <div>
      <CircularProgress />
    </div>
  );
if (teamIdError)
  return (
    <Typography
      style={{
        height: '100%',
        width: '100%',
        textAlign: 'center',
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center',
      }}
      variant="h6"
    >
      Вас ще не додали в команду. Зачекайте та почніть оцінювати.
    </Typography>
  );
if (companiesError)
  return (
    <Typography
      style={{
        height: '100%',
        width: '100%',
        textAlign: 'center',
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center',
      }}
      variant="h6"
    >
      Відбулася помилка під час завантаження завдань.
    </Typography>
  );

if (combinedCompanies.length === 0)
  return (

```

```

<Typography
  style={{
    height: '100%',
    width: '100%',
    textAlign: 'center',
    display: 'flex',
    justifyContent: 'center',
    alignItems: 'center',
  }}
  variant="h6"
>
  У вас ще немає завдань на оцінювання.
</Typography>
);

return (
  <Grid container spacing={4} height="fit-content">
    {combinedCompanies.map((company: TCompany360) => (
      <Grid item key={company.id} xs={12} sm={6} md={4} lg={3}>
        <Paper
          elevation={3}
          component={Link}
          to={` /tests360/$ {company.id} `}
          style={{ textDecoration: 'none' }}
          sx={{ ':hover': { cursor: 'pointer' } }}
        >
          <Typography variant="h6">{company.name}</Typography>
          <Typography color="textSecondary">{company.teams.join(', ')}</Typography>
        </Paper>
      </Grid>
    ))}
  </Grid>
);
};

export default ActiveCompanies360;

```

ДОДАТОК Б
Слайди презентації

Національний університет «Запорізька Політехніка»
Кафедра програмних засобів

Розробка вебзастосунку для оцінки професійних та комунікативних навичок працівників

Виконав:
ст. гр. КНТ-121 сп
Палієнко Д.С.

Керівник:
к.н.т., доцент
Каплієнко Т.І.

Рисунок Б.1 – Слайд №1

Інформація про роботу

- Мета роботи – створення вебзастосунку для автоматизації та прискорення навчання співробітників та оцінки їх комунікативних та професійних навичок.
- Об'єкт дослідження – програмні засоби для оцінки співробітників.

Завдання на дипломну роботу:

- Аналіз аналогів;
- Вибір програмного забезпечення та технологій для розробки;
- Проектування архітектури застосунку;
- Створення вебзастосунку;
- Тестування продукту;

2

Рисунок Б.2 – Слайд №2

Порівняння переваг аналогів

	<u>Moodle</u>	<u>TalentLMS</u>	<u>Qualtrics 360</u>
Можливості кастомізації	Широкі можливості для кастомізації та розширення функціоналу за допомогою розширень	інтуїтивно зрозумілий інтерфейс для адміністраторів та користувачів	гнучкі можливості для створення та налаштування опитувань і звітів
Підтримувані типи контенту	підтримує різні типи контенту та <u>активностей</u> , як-от тести, завдання та форуми»	вбудовані інструменти для <u>гейміфікації</u> навчального процесу	інтегрується з популярними системами управління персоналом (HRIS)
Вимоги до технічних знань	вимагає технічних знань для налаштування та адміністрування системи	інтуїтивно зрозумілий інтерфейс для адміністраторів та користувачів	для налаштування й аналізу результатів оцінювання потрібні кваліфіковані фахівці
Обмеження вбудованих можливостей	обмежені вбудовані можливості для оцінювання та зворотного зв'язку	не має вбудованих інструментів для оцінювання за методом 360 градусів	не має вбудованих інструментів для навчання та розвитку персоналу
Проблеми масштабованості	при значному збільшенні кількості користувачів та курсів можуть виникати проблеми з масштабованістю	менші можливості для <u>кастомізації</u> порівняно з Moodle	для налаштування й аналізу результатів оцінювання потрібні кваліфіковані фахівці

3

Рисунок Б.3 – Слайд №3

Порівняння технологій для серверної частини сайту

	<u>NestJS</u>	<u>Express</u>	<u>Koa</u>	<u>Hapi</u>
Архітектурний підхід	<u>Модульний</u>	<u>Мінімалістичний</u>	<u>Мінімалістичний</u>	<u>Конфігураційний</u>
Типова система	TypeScript	JavaScript	JavaScript	TypeScript, JavaScript
Шаблони проектування	Dependency Injection, модулі, декоратори	<u>Відсутні</u>	<u>Відсутні</u>	<u>Плагіни, хуки</u>
Вбудована підтримка API	<u>Повна</u>	<u>Мінімальна</u>	<u>Мінімальна</u>	<u>Середня</u>
Документація	<u>Розгорнута</u>	<u>Середня</u>	<u>Середня</u>	<u>Розгорнута</u>
Підтримка тестування	<u>Висока</u>	<u>Мінімальна</u>	<u>Мінімальна</u>	<u>Висока</u>
Продуктивність	<u>Висока</u>			
Складність навчання	<u>Середня/Висока</u>	<u>Низька</u>	<u>Середня</u>	<u>Середня</u>
Підтримка TypeScript	<u>Вбудована</u>	<u>Може бути налаштована</u>	<u>Може бути налаштована</u>	<u>Вбудована</u>

4

Рисунок Б.4 – Слайд №4

Порівняння технологій для клієнтської частини сайту

	React	Angular	Vue.js	Svelte
Архітектурний підхід	Компонентний	MVC	Компонентний	Компонентний
Типова система	JavaScript/TypeScript	TypeScript	JavaScript/TypeScript	JavaScript
Документація	Висока, велика спільнота	Висока, офіційні ресурси та спільнота	Висока, доступна та добре структурована	Висока, зростаюча спільнота
Продуктивність	Висока	Висока	Висока	Дуже висока
Підтримка мобільних пристроїв	React Native	Ionic	Weex	Svelte Native
Легкість навчання	Середня	Висока складність	Легка	Легка
Популярність	Дуже висока	Висока	Висока	Зростаюча
Вбудовані інструменти	Обмежені, але багато сторонніх бібліотек	Багато вбудованих інструмент	Обмежені, але багато сторонніх бібліотек	Багато вбудованих інструмент

5

Рисунок Б.5 – Слайд №5

Порівняння систем керування базами даних

	PostgreSQL	MySQL	SQLite	MongoDB
Тип	Реляційна	Реляційна	Реляційна	Документоорієнтована
Система управління транзакціями	ACID	ACID	ACID	BASE
Масштабованість	Висока	Висока	Обмежена	Висока
Підтримка складних запитів	Висока	Висока	Середня	Низька
Підтримка JSON	Вбудована підтримка JSON та JSONB	Вбудована підтримка JSON	Обмежена	Нативна підтримка JSON
Підтримка реплікації	Асинхронна та синхронна реплікація	Асинхронна реплікація	Відсутня	Асинхронна реплікація
Розширюваність	Висока	Середня	Обмежена	Висока
Продуктивність	Висока	Висока	Висока для малих обсягів даних	Висока для великих обсягів даних
Легкість налаштування	Середня, вимагає технічних знань	Легка	Дуже легка	Середня

6

Рисунок Б.6 – Слайд №6



Рисунок Б.7 – Слайд №7

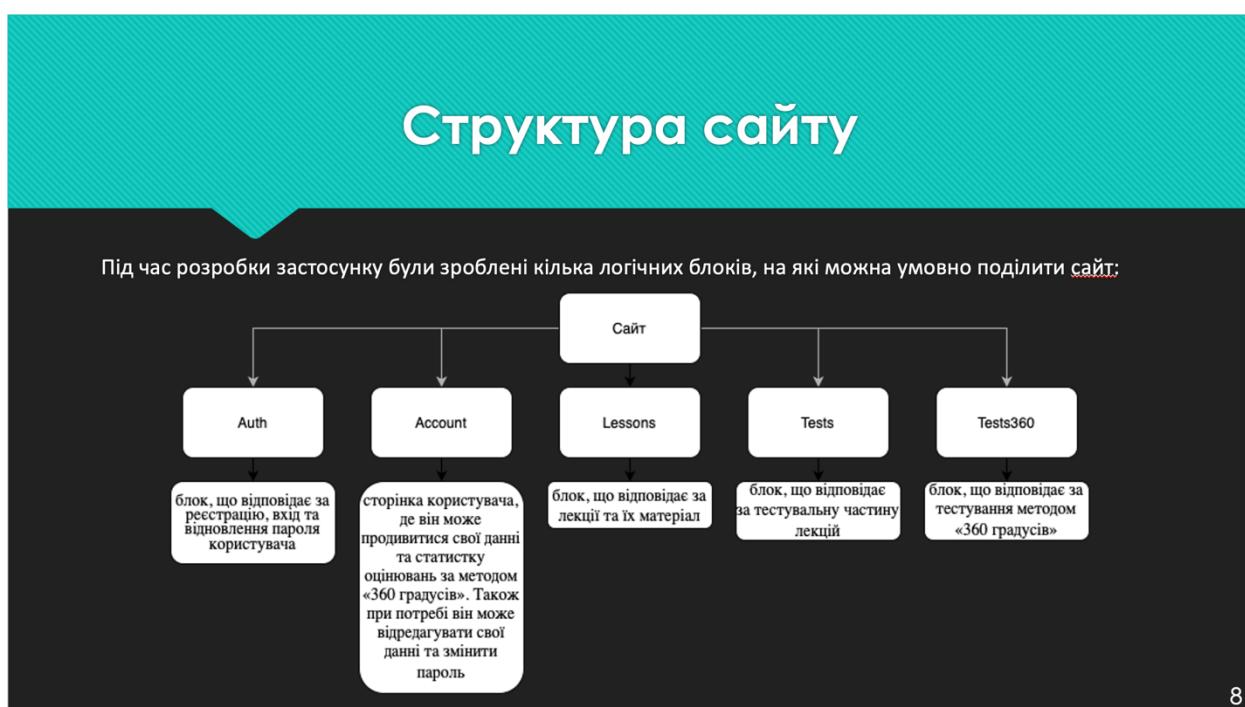


Рисунок Б.8 – Слайд №8

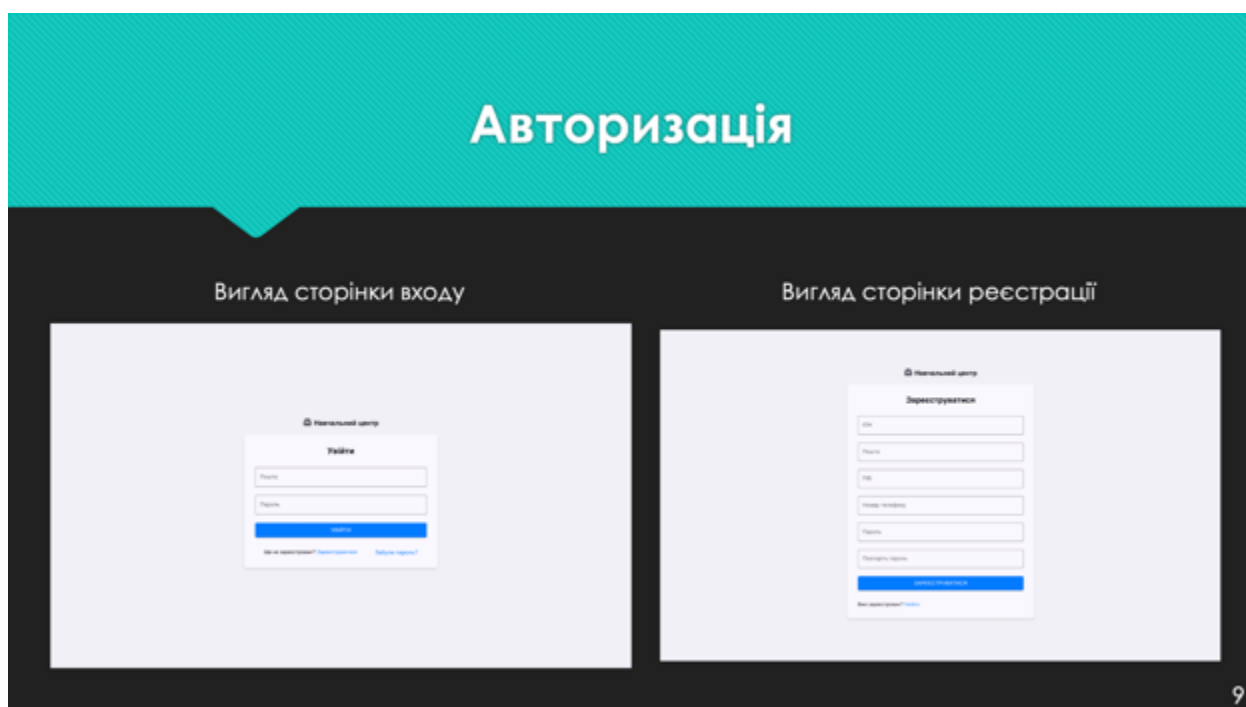


Рисунок Б.9 – Слайд №9

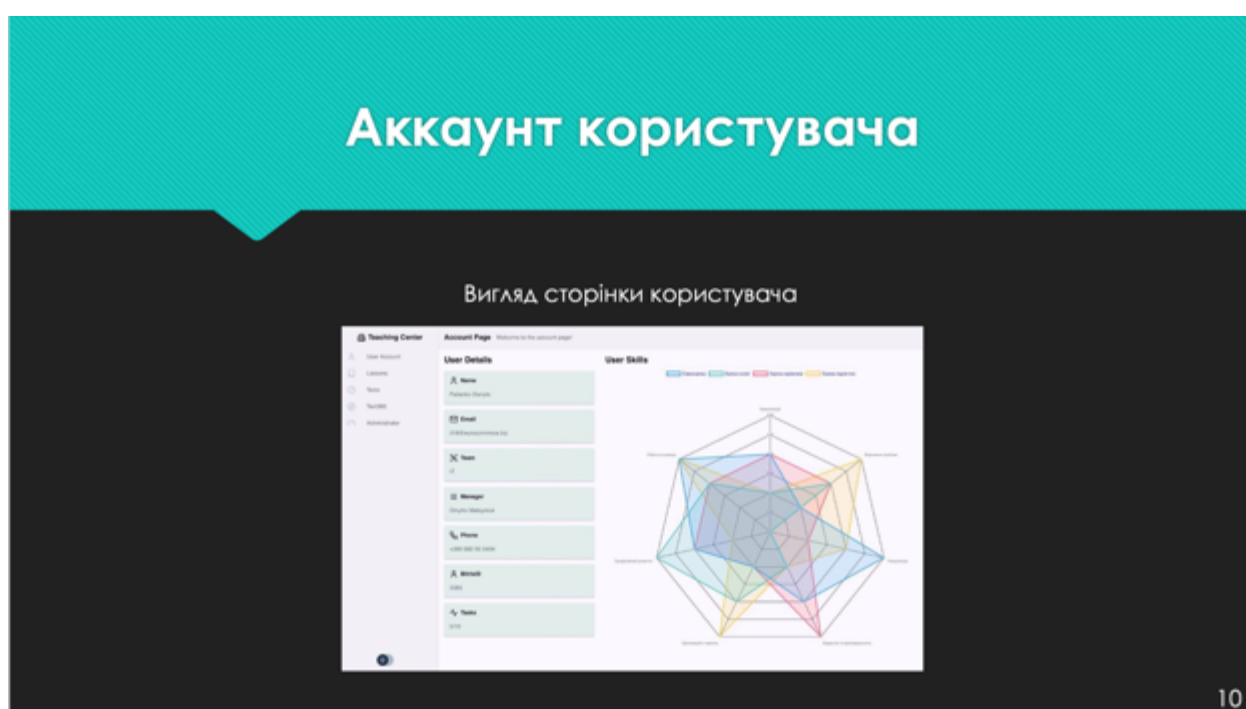


Рисунок Б.10 – Слайд №10

Лекційна частина сайту

Вигляд сторінки з лекціями

Вигляд сторінки лекції

12

Рисунок В.12 – Слайд №12

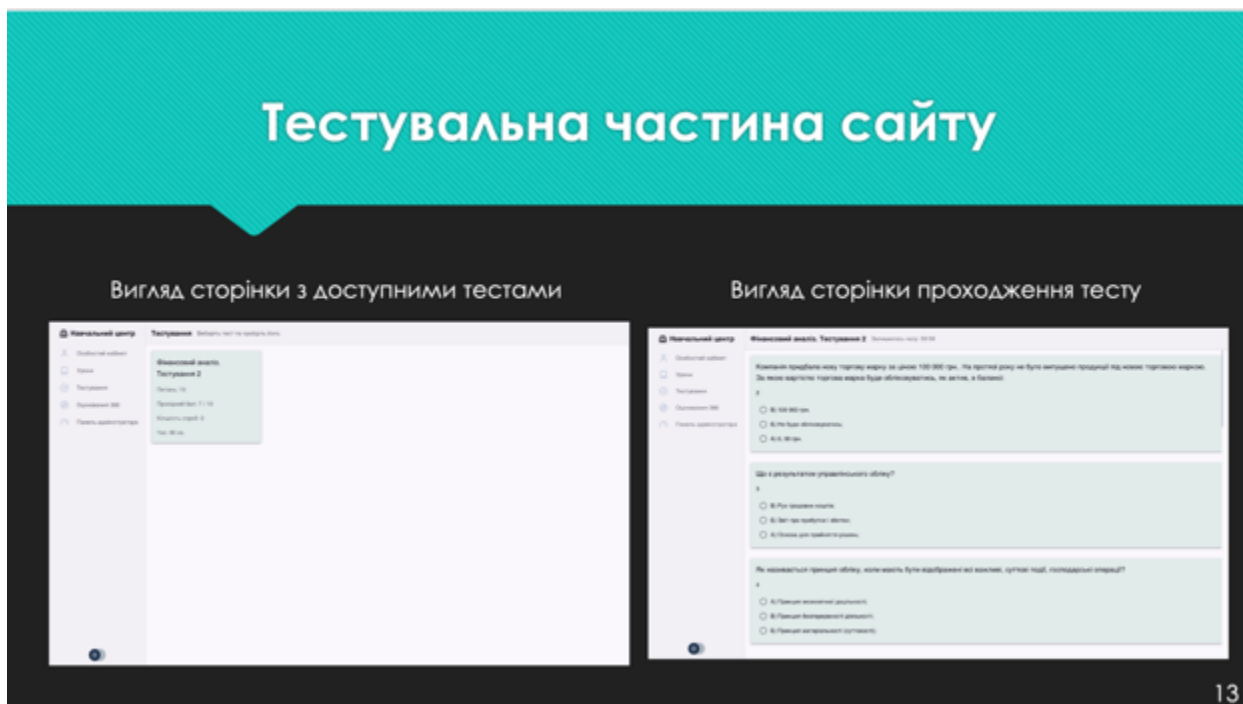


Рисунок Б.13 – Слайд №13

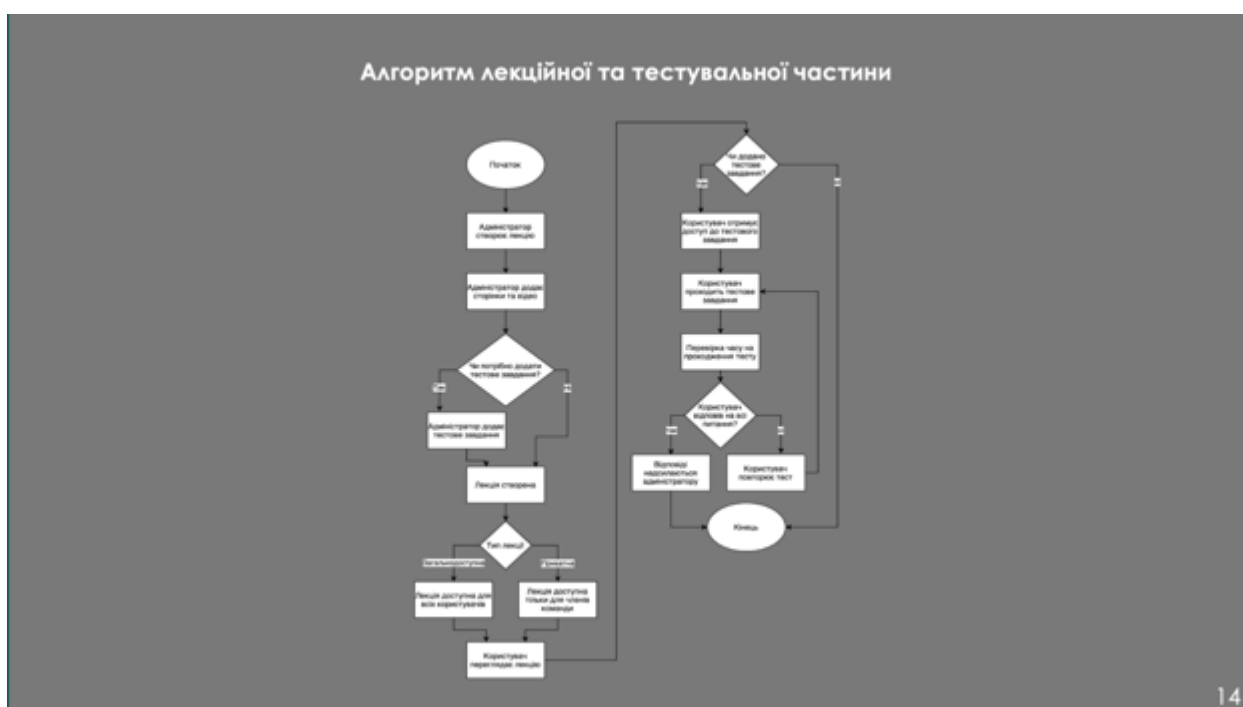


Рисунок Б.14 – Слайд №14

Частина оцінювання методом «360 градусів»

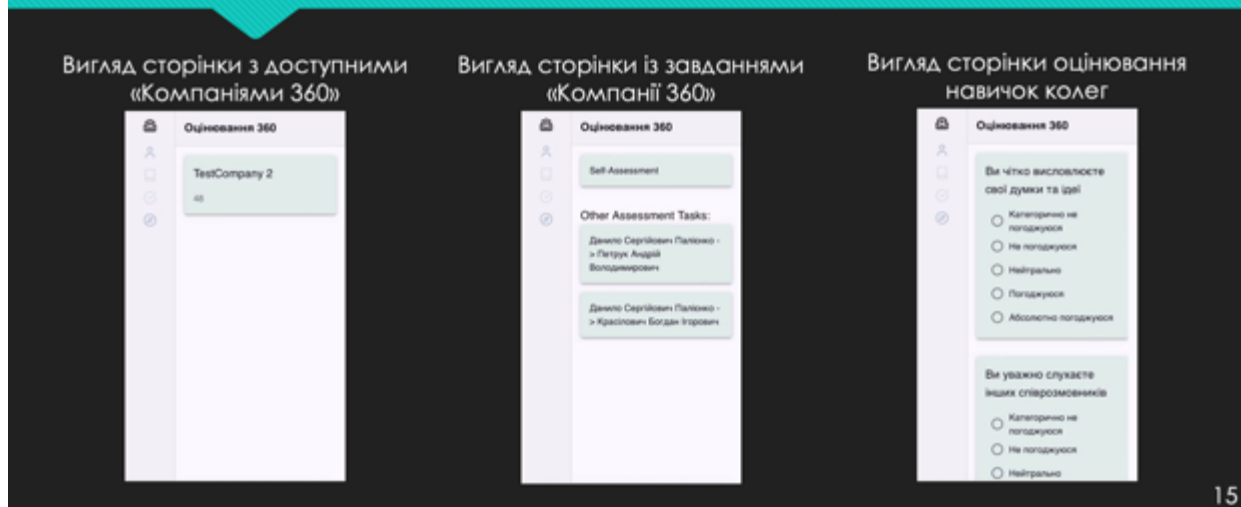


Рисунок Б.15 – Слайд №15



Рисунок Б.16 – Слайд №16

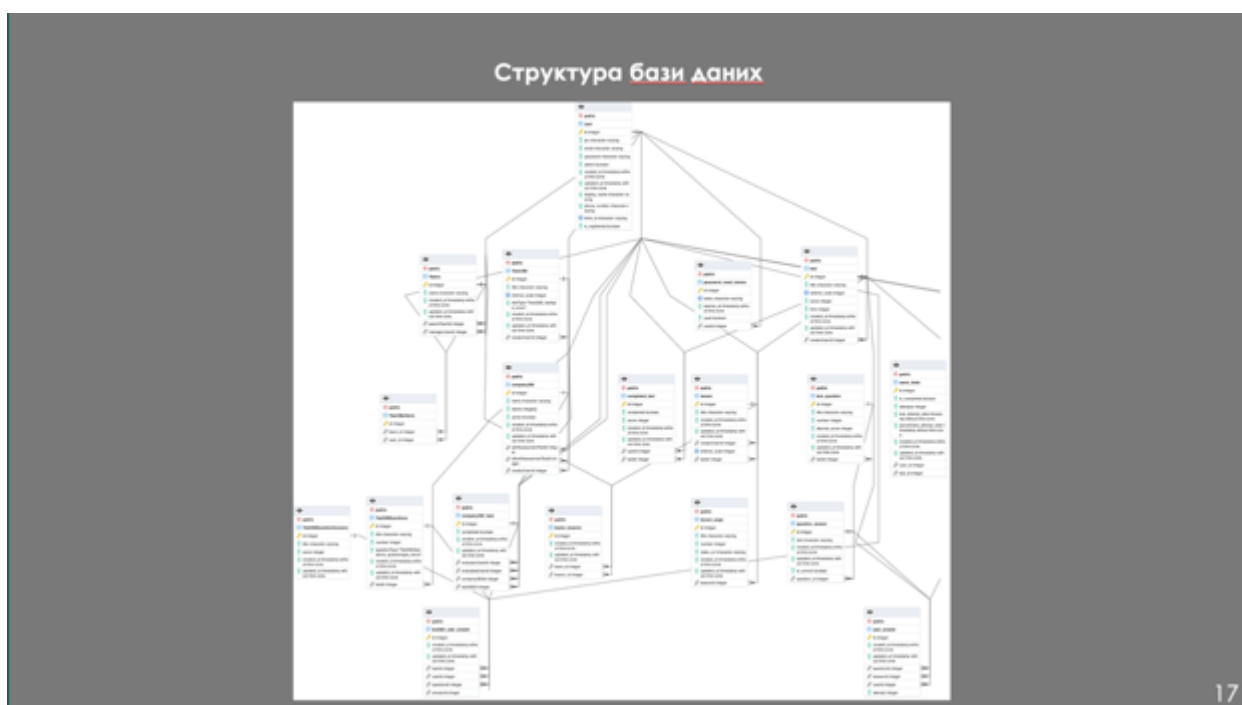


Рисунок Б.17 – Слайд №17

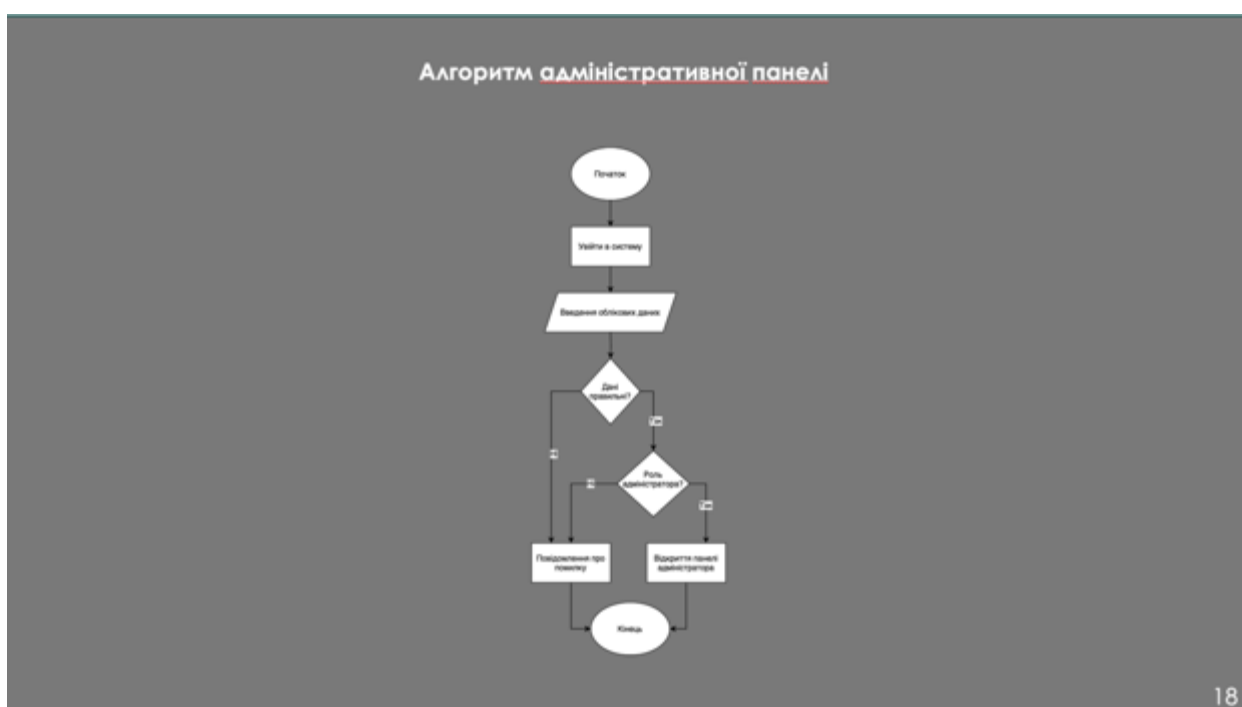


Рисунок Б.18 – Слайд №18

Приклади використаних запитів TypeORM в системі

Текст запиту	Опис
<code>this.lessonsRepository.createQueryBuilder('lesson').leftJoin('lesson.lessons', 'teamsLessons').leftJoinAndSelect('lesson.pages', 'pages').leftJoinAndSelect('lesson.test', 'test').where('teamsLessons.id IS NULL');</code>	Пошук загальнодоступних лекцій
<code>this.lessonsRepository.createQueryBuilder('lesson').leftJoinAndSelect('lesson.lessons', 'teamsLessons').leftJoinAndSelect('lesson.pages', 'pages').leftJoinAndSelect('lesson.test', 'test').where('teamsLessons.teamId = :teamId', { teamId });</code>	Пошук лекцій для команди користувача

19

Рисунок Б.19 – Слайд №19

Висновки

Під час виконання дипломної роботи були виконані наступні завдання:

- Створено вебзастосунок «Навчальний центр»;
- Обрано технології, які найкраще підходять для виконання поставленої задачі;
- Вебзастосунок протестовано та налагоджено;

Розроблений застосунок надає користувачам можливість навчатися, проходити тестування та оцінювати професійні та комунікативні навички співробітників. Застосунок забезпечує необхідний функціонал для реєстрації користувачів, вибору лекцій, проходження тестів після лекцій та оцінки співробітників за методом «360 градусів».

20

Рисунок Б.20 – Слайд №20