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

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 126 «Інформаційні системи та технології»
на тему: «Інформаційна система оцінювання якості роботи співробітників»

Виконав:
студент IV курсу, групи ІС-11
Бондарчук Олександр Андрійович _____

Керівник:
Доцент кафедри ІСТ ФІОТ, к.т.н., доцент,
Жураковська Оксана Сергіївна _____

Рецензент:
Доцент кафедри ОТ, к.т.н., доцент,
Марковський Олександр Петрович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____2025 р.

ЗАВДАННЯ
на дипломний проєкт студенту
Бондарчуку Олександру Андрійовичу

1. Тема проєкту «Інформаційна система оцінювання якості роботи співробітників», керівник проєкту Жураковська Оксана Сергіївна, к.т.н., доцент, затверджені наказом по університету від «*23*» травня 2025 р.
№ 1705-с.

2. Термін подання студентом проєкту: “*9*” червня 2025 р.

3. Вихідні дані до проєкту: мова програмування C#, платформа .NET, платформа ASP.NET core, база даних MS SQL Server, бібліотека DinkToPdf, мова програмування TypeScript, фреймворк Angular, бібліотека PrimeNG.

4. Зміст пояснювальної записки:

1. Опис предметної області: основні визначення та терміни, опис предметного середовища, постановка задачі;
2. Огляд наявних аналогів: порівняльна характеристика існуючих аналогів із розробленою інформаційною системою;
3. Формування вимог до системи: вимоги до структури, функціональних та нефункціональних характеристик програмного забезпечення;
4. Вибір технологій розробки: обґрунтування вибору мов програмування, фреймворків, СУБД, інструментів розробки;

5. Розроблення інформаційної системи: архітектура програмного забезпечення, структура інтерфейсу, вхідні та вихідні дані;
6. Математичне забезпечення: змістовна та математична постановка задачі, обґрунтування та опис методу розв’язання;
7. Тестування системи: перевірка відповідності програмного забезпечення системи функціональним та нефункціональним вимогам.

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо): діаграма використання, діаграма послідовності, схема бази даних, діаграма розгортання

6. Дата видачі завдання «*4*» березня 2025 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	14.04.2025	
2	Аналіз предметної області	20.04.2025	
3	Постановка та формалізація задачі	21.04.2025	
4	Огляд наявних аналогів	23.04.2025	
5	Аналіз вимог до програмного забезпечення	24.04.2025	
6	Обґрунтування використовуваних технічних засобів	30.04.2025	
7	Розробка архітектури програмного забезпечення	08.05.2025	
8	Розробка програмного забезпечення	11.05.2025	
9	Тестування та налагодження програми	25.05.2025	
10	Виконання графічних документів	30.05.2025	
11	Оформлення пояснювальної записки	03.06.2025	

Студент Олександр БОНДАРЧУК

Керівник Оксана ЖУРАКОВСЬКА

АНОТАЦІЯ

Інформаційна система оцінювання якості роботи співробітників.

Проєкт містить 73 с. тексту, 33 рисунки, 12 таблиць, посилання на 15 літературних джерел, додатки та 4 конструкторських документи.

ВЕБЗАСТОСУНОК, ОЦІНЮВАННЯ СПІВРОБІТНИКІВ, KPI, РОЛІ, КОМАНДА, ЗВІТ.

Об’єктом розробки є процес оцінювання якості роботи співробітників у командно-орієнтованих компаніях.

Мета розробки – пришвидшення та покращення ефективності процесів оцінювання якості роботи співробітників у командах на основі ключових показників ефективності (KPI).

У першому розділі проаналізовано особливості процесу оцінювання співробітників в сучасних компаніях, визначено основні проблеми та поставлено задачі, які вирішує розроблена система. У другому розділі здійснено огляд існуючих HRM-систем, їхніх переваг і недоліків. У третьому розділі сформовано функціональні та нефункціональні вимоги, зокрема багаторольову модель доступу, підтримку KPI, генерацію звітів та прогнозів. У четвертому розділі обґрунтовано вибір технологій розробки, додаткові бібліотеки. У п’ятому розділі реалізовано архітектуру системи, описано модель бази даних, взаємодію компонентів, варіанти використання та інтерфейси. У шостому розділі представлено математичне забезпечення: методи прогнозування результативності з прикладами обчислення тренду розвитку. У сьомому розділі проведено тестування системи: перевірено відповідність функціональним та нефункціональним вимогам.

Результати розробки можуть бути використані HR-відділами компаній для об’єктивного оцінювання персоналу, формування звітності, а також для планування розвитку працівників.

SUMMARY

Information system for evaluating employee performance.

The project includes 73 pages of text, 33 figures, 12 tables, references to 15 literary sources, appendices, and 4 design documents.

WEB APPLICATION, EMPLOYEE EVALUATION, KPI, ROLES, TEAM, REPORT.

Object of the development is the process of evaluating employee performance in team-oriented companies.

Purpose of the development is to accelerate and improve the efficiency of employee performance evaluation processes within teams based on key performance indicators (KPI).

The first section analyzes the specifics of employee evaluation processes in modern companies, identifies key issues, and defines the tasks that the developed system aims to solve. The second section provides an overview of existing HRM systems, highlighting their strengths and weaknesses. The third section formulates functional and non-functional requirements, including a multi-role access model, KPI support, and the generation of reports and forecasts. The fourth section justifies the choice of development technologies and additional libraries. The fifth section implements the system architecture, describes the database model, component interaction, usage scenarios, and user interfaces. The sixth section presents the mathematical support: methods for forecasting employee performance, including examples of trend calculation. The seventh section covers system testing: verifying compliance with both functional and non-functional requirements.

The results of the development can be used by HR departments for objective employee evaluation, report generation, and planning of employee development.

Номер рядка	Формат	Позначення	Найменування	Кільк. аркушів	Номер екзем.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІС11.040БАК.005 ПЗ	Пояснювальна записка	73		
6	А3	ІС11.040БАК.005 Д1	Інформаційна система	1		
7			оцінювання якості роботи			
8			співробітників. Діаграма			
9			використання			
10	А3	ІС11.040БАК.005 Д2	Інформаційна система	1		
11			оцінювання якості роботи			
12			співробітників. Діаграма			
13			послідовності			
14	А3	ІС11.040БАК.005 Д3	Інформаційна система	1		
15			оцінювання якості роботи			
16			співробітників. Схема			
17			бази даних			
18	А3	ІС11.040БАК.005 Д4	Інформаційна система	1		
19			оцінювання якості роботи			
20			співробітників. Діаграма			
21			розгортання			
22						
23						
24						
25						
26						
27						
28						

					ІС-11.040БАК.005 ТП
Зм.	Аркуш	№ докум.	Підпис		
Розробив		Бондарчук О.А.			Інформаційна система оцінювання якості роботи співробітників Відомість дипломного проєкту
Перевірив		Жураковська О.С.			
Затв.					

Літ.	Аркуш	Аркушів
	1	1

КПІ ім. Ігоря Сікорського
Група ІС-11

Пояснювальна записка
до дипломного проєкту
на тему: «Інформаційна система оцінювання якості роботи співробітників»

Київ – 2025 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API (Application Programming Interface) – прикладний програмний інтерфейс

Backend – серверна частина додатку

CSS (Cascading Style Sheets) – каскадні таблиці стилів

Frontend – інтерфейс для взаємодії між користувачем та сервером

HR (Human Resources) – людські ресурси

IT – інформаційні технології

JWT (JSON Web Token) – ключ аутентифікації користувача

KPI (Key Performance Indicators) – ключові показники ефективності

OKR (Objectives and Key Results) – цілі і ключові результати

ORM (Object-relational mapping) – технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних»

SCSS (Syntactically Awesome Style Sheet) – синтаксично чудова таблиця стилів

SQL (Structured Query Language) – мова структурованих запитів

UI (User Interface) – інтерфейс користувача

ВСТУП

У сучасних реаліях ефективне управління людськими ресурсами є ключовим чинником успіху командно-орієнтованих компаній, зокрема в галузі інформаційних технологій. Зростання складності проєктів, швидка зміна ринкових вимог і технологій, а також високий рівень конкуренції створюють потребу в систематичному підході до оцінювання якості роботи працівників. Саме тому автоматизація процесу оцінки співробітників на основі ключових показників ефективності (KPI), а також прогнозування їхнього професійного розвитку є актуальним напрямом у сфері розробки корпоративного програмного забезпечення.

Незважаючи на наявність окремих HR-платформ, більшість компаній стикається з проблемою відсутності гнучких та адаптованих до внутрішніх процесів рішень для об’єктивного аналізу динаміки змін у якості роботи співробітників. Особливої уваги потребує врахування не лише поточних оцінок, але й історичних даних з урахуванням вагових коефіцієнтів, часової давності та трендів розвитку. Відсутність таких інструментів ускладнює прийняття управлінських рішень, знижує прозорість оцінювання та стримує кар’єрне зростання працівників.

Об’єктом дослідження є процес оцінювання якості роботи співробітників у командно-орієнтованих компаніях.

Призначенням розробки є підтримка процесів оцінювання якості роботи співробітників на основі ключових показників ефективності та історичних даних.

Практичне значення одержаних результатів полягає в можливості застосування розробленого вебзастосунку в HR-відділах IT-компаній та інших організацій для підвищення якості управління персоналом, спрощення та вдосконалення процесів оцінювання та сприяння прийняттю стратегічних рішень щодо розвитку працівників.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис процесу діяльності

У більшості IT-компаній або інших командно-орієнтованих організацій оцінювання ефективності роботи співробітників здійснюється вручну або за допомогою базових офісних інструментів, таких як таблиці Excel, опитувальники Google Forms, або навіть усні обговорення на командних зустрічах. Як правило, оцінювання проводиться періодично – один або два рази на рік – і ініціюється менеджерами команд або HR-відділом.

Процес включає в себе збір оцінок за певними критеріями: технічні навички, якість виконаних завдань, командна взаємодія, дотримання термінів тощо. Ці критерії можуть бути прописані у внутрішніх регламентах компанії, але часто вони застосовуються неформально або без єдиної шкали. Менеджери зазвичай оцінюють працівників інтуїтивно або на основі власного суб’єктивного досвіду взаємодії з підлеглими, що знижує об'єктивність та ускладнює порівняння результатів між різними командами. Також відсутність єдиної бази зберігання оцінок і механізму автоматизованого аналізу історичних даних ускладнює відстеження динаміки змін професійної результативності співробітника.

Ускладнення виникають і на етапі узагальнення результатів: необхідно вручну порівнювати оцінки, визначати середні значення, робити висновки щодо потенційного розвитку працівника. Немає можливості швидко згенерувати звіт або побачити позитивні та негативні зміни, особливо якщо мова йде про десятки співробітників. Крім того, керівникам бракує інструментів для формування рекомендацій на основі оцінювання – наприклад, щодо навчання, кар’єрного зростання або зміни ролі в команді.

Загалом, наявна система оцінювання в компаніях є трудомісткою, недостатньо стандартизованою, вразливою до людського фактору і позбавленою аналітичних механізмів. Це створює потребу у впровадженні автоматизованого рішення, яке стандартизуватиме процес, забезпечить об’єктивність та зручність у прийнятті кадрових рішень.

1.2 Постановка задачі

1.2.1 Призначення розробки

Призначенням розробки є підтримка процесів оцінювання якості роботи співробітників на основі ключових показників ефективності та історичних даних у командно-орієнтованих компаніях. Система повинна забезпечити зручне управління критеріями оцінювання, збереження історії результатів, візуалізацію динаміки розвитку працівників, а також підтримку прийняття рішень щодо їхнього подальшого розвитку.

1.2.2 Цілі та задачі розробки

Метою розробки є спрощення та підвищення ефективності процесів оцінювання якості роботи співробітників на основі ключових показників ефективності, що дозволить не лише оцінювати працівників у конкретний момент часу, а й виявляти тенденції їх професійного росту чи спаду. Це є надзвичайно важливим для прийняття обґрунтованих управлінських рішень у межах командо-орієнтованих компаній. Для досягнення цієї мети необхідно виконати такі задачі:

– сформувати набір релевантних метрик для оцінки працівників, які можуть виступати як KPI. До них можуть входити технічні навички, дотримання дедлайнів, участь у командній роботі, ініціативність, тощо. Важливо також визначити їх вагомість у загальній оцінці;

– розробити структуровану та гнучку модель бази даних для зберігання результатів оцінювання, даних про співробітників, ролі, команди та сесії оцінювання;

– розробити алгоритм обчислення оцінки якості роботи працівника та алгоритм прогнозування результативності роботи працівника, який дозволить аналізувати зміну якості роботи працівника з урахуванням часу, вагових коефіцієнтів метрик, а також останніх оцінок, що мають більшу актуальність. Цей алгоритм має формувати висновки про тренд розвитку (позитивний, негативний);

– реалізувати зручний та інтуїтивно зрозумілий інтерфейс користувача, адаптований під потреби менеджерів, HR-фахівців та інших співробітників. Застосунок має підтримувати багаторольову модель з відповідним доступом до функцій;

– забезпечити можливість генерації звітів та індивідуальних рекомендацій щодо розвитку працівника. Це сприятиме кращому кадровому плануванню та підвищенню ефективності управління персоналом.

Висновок до розділу 1

У даному розділі було здійснено аналіз предметної області, пов’язаної з процесом оцінювання якості роботи співробітників у сучасних організаціях. Детально описано поточний стан процесу оцінювання в компаніях без використання спеціалізованих інформаційних систем, що дозволило виявити ключові проблеми, серед яких – суб’єктивність оцінювання, складність аналітики, відсутність автоматизації та обмеженість у прогнозуванні розвитку персоналу.

У підсумку було чітко визначено призначення системи, її цілі та задачі, що визначають напрям подальшої розробки та вказують на практичну потребу створення інноваційного інструменту для HR-аналітики та прийняття управлінських рішень.

2. ОГЛЯД НАЯВНИХ АНАЛОГІВ

Важливою частиною процесу розробки будь-якої інформаційної системи є дослідження існуючих рішень або аналогів. Це дозволяє не лише оцінити поточний стан ринку, а й виявити сильні та слабкі сторони наявних продуктів, що, у свою чергу, сприяє формуванню унікальних переваг власної розробки.

У ході дослідження було знайдено кілька платформ, які частково або повністю реалізують функціонал оцінювання персоналу в командо-орієнтованих організаціях. Серед них особливу увагу привернули такі рішення: PeopleForce, Hurma та Smart IT HR. Нижче наведено короткий опис кожної з них.

2.1 Платформа PeopleForce

PeopleForce – це хмарна HR-платформа, орієнтована на автоматизацію процесів управління персоналом у малих та середніх компаніях. Система пропонує модулі для рекрутингу, управління відпустками, оцінювання ефективності, аналітики та самообслуговування співробітників. Особливістю PeopleForce є гнучке налаштування OKR, проведення опитувань для вимірювання залученості персоналу та зручна система звітності. Проте, хоча платформа надає інструменти для оцінювання, вона не акцентує увагу на довгостроковому аналізі динаміки розвитку співробітників або прогнозуванні їхньої професійної результативності на основі історичних даних.

2.2 Система Hurma

Hurma – українська HRM-система, яка об'єднує функції управління персоналом, рекрутингу та OKR в єдиній платформі. Вона забезпечує автоматизацію процесів адаптації нових працівників, облік відпусток, управління цілями та зворотним зв'язком. Hurma також пропонує аналітичні інструменти для оцінки ефективності HR-процесів. Незважаючи на широкий функціонал, система

більше зосереджена на поточному стані справ, ніж на прогнозуванні майбутньої професійної результативності співробітників на основі оцінок.

2.3 Система Smart IT HR

Smart IT HR – рішення, яке охоплює базові HR-функції, включаючи управління персоналом, облік робочого часу, нарахування заробітної плати та управління проектами. Платформа також пропонує портал самообслуговування для співробітників, де вони можуть переглядати свої дані, подавати заявки на відпустки та ознайомлюватися з розрахунковими листами. Хоча Smart IT HR забезпечує основні потреби HR-відділів, вона не має розширених можливостей для глибокого аналізу ефективності працівників або прогнозування їхнього розвитку.

Систематизуємо зібрану інформацію відносно наших потреб у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика існуючих аналогів.

Рішення	Переваги	Недоліки
PeopleForce	Гнучка система OKR, зручна візуалізація звітності, можливість проведення опитувань та збору зворотного зв’язку	Відсутність аналізу історичних даних, недостатня увага до прогнозування розвитку персоналу.
Hurma	Широкий спектр функціоналу: рекрутинг, OKR, облік часу, адаптація. Зручний інтерфейс.	Орієнтація на поточну оцінку, відсутність аналітики на основі попередніх результатів.
Smart IT HR	Підтримка базових HR-процесів, зручність для малого бізнесу, портал самообслуговування.	Відсутність функціоналу для оцінювання ефективності та аналітики професійної результативності.

Продовження таблиці 1.1

Розроблене рішення	Орієнтація на прогнозування професійної результативності співробітників, використання KPI, збереження історії оцінювання, аналітика з урахуванням динаміки та вагових коефіцієнтів.	Не така широка функціональність у сфері рекрутингу та бухгалтерського обліку, як у деяких аналогів.

Висновок до розділу 2

Проведений аналіз наявних аналогів показав, що сучасні HRM-системи, такі як PeopleForce, Hurma та Smart IT HR, пропонують широкий набір інструментів для управління персоналом, однак мають певні обмеження в контексті аналізу та прогнозування професійної результативності співробітників. PeopleForce вирізняється гнучкою системою OKR і зручною звітністю, але не акцентує увагу на довгостроковій аналітиці. Hurma забезпечує комплексний функціонал для HR-процесів, проте орієнтована переважно на поточну оцінку, а не на прогнозування. Smart IT HR підходить для базових потреб, але має нестачу розширених можливостей для оцінки ефективності.

Розроблене рішення, на відміну від аналогів, фокусується на прогнозуванні результативності з використанням KPI, збереженні історії оцінювання та аналітиці з урахуванням динаміки й вагових коефіцієнтів, хоча поступається конкурентам у широті функціоналу для рекрутингу та бухгалтерського обліку. Таким чином, створена система займає унікальну нішу, пропонуючи інструменти для стратегічного управління персоналом із акцентом на довгостроковий розвиток.

3 ФОРМУВАННЯ ВИМОГ ДО СИСТЕМИ

3.1 Функціональні вимоги

Інформаційна система оцінювання якості роботи співробітників повинна забезпечувати виконання ряду функцій, що охоплюють усі етапи процесу оцінювання та адаптовані до ролей користувачів: менеджера, лідера команди, оцінювача та співробітника.

Система повинна надавати можливість:

– створення, редагування та видалення користувачів, команд і ролей у межах організаційної структури;

– визначення переліку метрик для оцінювання персоналу відповідно до ролі, з можливістю призначення вагових коефіцієнтів для кожної метрики;

– формування сесій оцінювання, що мають часові обмеження;

– оцінювання співробітників з обмеженням доступу на основі ієрархії ролей і приналежності до команди;

– додавання коментарів до оцінок для уточнення причин і контексту;

– автоматичного розрахунку середньозваженої оцінки на основі призначених коефіцієнтів;

– класифікації співробітників за рівнем ефективності на основі підсумкової оцінки (наприклад: «початковий», «стабільний», «лідер»);

– генерування рекомендацій щодо розвитку працівника на основі виявлених сильних і слабких сторін;

– формування PDF-звітів з результатами оцінювання, прогнозом ефективності, коментарями та рекомендаціями, збереження таких звітів у базі даних;

– перегляду співробітниками власних рекомендацій після оцінювання;

– забезпечення багаторівневої системи авторизації та автентифікації з можливістю гнучкого керування правами доступу до функціональних розділів;

– підтримки користувацького інтерфейсу, доступного через сучасні браузери, з фокусом на інтуїтивність і простоту взаємодії.

Система також повинна мінімізувати вплив людського фактору в процесі оцінювання, забезпечити захист даних і дотримання принципу об'єктивності при виставленні оцінок завдяки обмеженню доступу до оцінювання лише в межах команди. Така модель взаємодії передбачає посилення довіри до системи, сприяє формуванню культури зворотного зв’язку та підтримує розвиток командної ефективності.

3.2 Нефункціональні вимоги

Нефункціональні вимоги визначають загальні характеристики, яким має відповідати система для забезпечення якості, надійності та зручності використання.

По-перше, важливою вимогою є стабільність і надійність функціонування. Система має бути готовою до безперервної роботи впродовж тривалого часу без збоїв або втрати даних. Усі дані про оцінювання, профілі співробітників, метрики та звіти повинні зберігатися у реляційній базі даних, що гарантує цілісність та узгодженість даних.

З точки зору продуктивності, система повинна забезпечувати швидке завантаження сторінок, миттєвий відгук на дії користувача та підтримку одночасної роботи щонайменше кількох десятків активних користувачів. Критично важливими є оптимізовані запити до бази даних та асинхронне оброблення запитів, зокрема при формуванні звітів.

У плані безпеки система повинна реалізовувати захист від типових загроз, зокрема SQL-ін’єкцій і підміни сесій. Необхідна підтримка захищеного протоколу передачі гіпертексту (HTTPS), зберігання паролів у хешованому вигляді та захист адміністративного доступу.

Інтерфейс користувача має бути доступним, з чіткою візуалізацією даних і адаптивною верстkою для коректного відображення на різних пристроях.

У перспективі, система повинна бути розширюваною: підтримувати додавання нових типів метрик, інтеграцію з зовнішніми HR-системами, можливість вивантаження даних у зовнішні формати (Excel, CSV) та підтримку REST API для

взаємодії з іншими програмами. Загалом, система має бути надійним, безпечним і зручним інструментом для забезпечення об’єктивного оцінювання персоналу, який легко масштабувати та адаптувати до потреб конкретної організації.

Висновок до розділу 3

У результаті формування вимог до системи було визначено повний перелік функціональних і нефункціональних характеристик, які стануть основою подальшої розробки вебзастосунку для оцінювання якості роботи співробітників на основі KPI. Сформульовані функціональні вимоги охоплюють всі основні сценарії взаємодії користувачів із системою – від налаштування структури оцінювання менеджерами до проведення оцінювання і генерації звітів. Особливу увагу приділено ролям користувачів, обмеженням доступу до функцій оцінювання та автоматизації процесів формування рекомендацій.

Нефункціональні вимоги забезпечують надійність, безпеку та масштабованість системи, що є важливими для її ефективної роботи в умовах реального бізнес-середовища. Запропонована структура вимог створює основу для подальшого проєктування архітектури системи, розробки інтерфейсів і реалізації алгоритмів оцінювання. Такий підхід дозволяє створити гнучкий інструмент, що сприятиме підвищенню професійної результативності команд та індивідуального професійного розвитку співробітників.

4 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ

У цьому розділі обґрунтовано вибір інструментів і технологій, які були використані для розробки вебзастосунку оцінювання співробітників на основі KPI. Вибір здійснювався з урахуванням функціональних потреб системи, вимог до продуктивності, безпеки, масштабованості, а також досвіду командної розробки з відповідним набором технологій. Кожна технологія була ретельно проаналізована, щоб забезпечити ефективну реалізацію як клієнтської, так і серверної частин застосунку.

4.1 Клієнтська частина

Клієнтська частина системи реалізована за допомогою сучасного фреймворку Angular [1], оскільки він є одним із провідних інструментів для створення односторінкових застосунків (Single Page Applications, SPA), що ідеально підходить для динамічного інтерфейсу оцінювання співробітників. Його компонентно-орієнтована архітектура дозволяє розбивати інтерфейс на незалежні, повторно використовувані компоненти, такі як таблиці оцінок, форми введення KPI чи діалогові вікна. Також Angular інтегрований з бібліотекою RxJS, яка забезпечує реактивне програмування за допомогою потоків подій, що є корисним для обробки асинхронних запитів, оновлення даних у реальному часі та управління станом компонентів [2].

Angular підтримує двосторонній зв’язок даних (two-way data binding), що автоматично синхронізує модель даних із представленням у реальному часі. Вбудована система маршрутизації дозволяє організувати навігацію між різними частинами застосунку (наприклад, між сторінками менеджера, HR та співробітників) без перезавантаження сторінки, що покращує користувацький досвід. Крім того, реактивні форми у Angular надають інструменти для створення складних форм із перевіркою введених даних, таких як введення KPI чи коментарів до оцінки.

У якості мови програмування для клієнтської частини обрано TypeScript – надбудову над JavaScript із підтримкою статичної типізації [3]. TypeScript дозволяє виявляти помилки ще на етапі компіляції, а не під час виконання, що є важливим для великих проєктів із кількома розробниками. Наприклад, якщо розробник помилково передасть рядок замість числа в функцію обробки KPI, TypeScript одразу вкаже на цю помилку. Мова підтримує об'єктно-орієнтоване програмування, дозволяючи використовувати класи, інтерфейси та спадкування, що сприяє структуризації коду. Оскільки Angular створений із використанням TypeScript, це забезпечує повну сумісність і покращену підтримку в сучасних середовищах розробки.

Для стилізації інтерфейсу використано SCSS – препроцесор CSS, який розширює можливості стандартного CSS [4]. SCSS дозволяє визначати змінні для зберігання кольорів, шрифтів чи розмірів (наприклад, основний колір бренду компанії чи розмір шрифту для заголовків), що спрощує їх зміну в одному місці без необхідності редагувати весь код стилів. Завдяки вкладеності стилі можна писати в ієрархічному вигляді, що робить код більш читабельним і логічно структурованим. SCSS допомагає підтримувати єдину дизайн-систему, що забезпечує послідовний вигляд інтерфейсу для всіх ролей користувачів – менеджерів, HR та співробітників.

Для прискорення розробки інтерфейсу використано бібліотеку UI-компонентів PrimeNG, розроблену спеціально для Angular [5]. Вона надає широкий набір готових компонентів, таких як таблиці для відображення списків співробітників, діалогові вікна для редагування даних, динамічні форми для введення KPI, календарі для вибору дат і графіки для візуалізації професійної результативності. Завдяки PrimeNG розробники можуть зосередитися на реалізації бізнес-логіки, а не на створенні базових елементів із нуля. Компоненти PrimeNG є адаптивними, що гарантує коректне відображення застосунку на різних пристроях, включаючи смартфони й планшети. Підтримка тем дозволяє легко адаптувати зовнішній вигляд під корпоративний стиль компанії, наприклад, змінюючи кольорову палітру чи шрифти.

4.2 Серверна частина

Серверна частина додатку реалізована на платформі ASP.NET Core – кросплатформеному фреймворку від Microsoft, відомому своєю високою та безпекою [6]. ASP.NET Core має модульну архітектуру, що дозволяє включати лише необхідні компоненти, зменшуючи обсяг застосунку та підвищуючи його швидкодію. Наприклад, для реалізації API, який обробляє запити на отримання чи оновлення даних про KPI, використовуються лише потрібні бібліотеки, що оптимізує ресурси. Підтримка впровадження залежностей спрощує управління залежностями між компонентами системи та полегшує написання тестів. Завдяки багатоплатформності ASP.NET Core може працювати на таких операційних системах як Windows, Linux чи macOS, що робить його гнучким вибором для розгортання.

Мовою програмування для бекенду обрано C# – високорівневу, об’єктно-орієнтовану мову з сильною типізацією [7], яка є основою екосистеми .NET. C# дозволяє писати чистий і виразний код, що полегшує реалізацію складної логіки, наприклад, обчислення рейтингів співробітників чи обробки звітів. Підтримка асинхронного програмування забезпечує високу продуктивність вебсервісів, дозволяючи обробляти численні запити одночасно без блокування потоків. Велика екосистема бібліотек і фреймворків у .NET додатково спрощує розробку корпоративних застосунків [8].

Для роботи з базою даних використано Entity Framework Core (EF Core) – ORM-фреймворк, який дозволяє взаємодіяти з базою даних через об’єкти C#, замість написання SQL-запитів вручну. Наприклад, створення запису про нового співробітника чи оновлення його KPI відбувається через зручні методи C#, а не складні SQL-запити. EF Core підтримує міграції, що спрощує внесення змін до схеми бази даних, наприклад, додавання нових полів для зберігання додаткових метрик.

База даних зберігається в Microsoft SQL Server – реляційній СУБД корпоративного рівня, яка забезпечує стабільність і надійність [9]. Підтримка

транзакцій гарантує цілісність даних, що критично важливо при одночасному оновленні оцінок кількома користувачами. Індексація прискорює виконання запитів, таких як пошук оцінювань за певним критерієм. SQL Server також має інструменти для резервного копіювання та відновлення, що забезпечує захист даних від збоїв.

Для адміністрування бази даних використовується SQL Server Management Studio (SSMS) – графічний інструмент, який спрощує роботу з SQL Server. SSMS дозволяє створювати та виконувати SQL-запити, переглядати й редагувати дані, налаштовувати параметри сервера та слідкувати за його продуктивністю. Наприклад, адміністратор може швидко перевірити структуру таблиць із даними про KPI чи проаналізувати журнали запитів для оптимізації.

4.3 Додаткові інструменти

Для генерації фейкових даних під час тестування використано бібліотеку Bogus для C#. Вона дозволяє створювати реалістичні тестові дані, такі як імена співробітників, дати оцінювання чи значення KPI, що відповідають реальним сценаріям. Це корисно на етапі розробки, коли доступ до реальних даних обмежений. Bogus підтримує налаштування правил генерації, що дозволяє імітувати специфічні умови, наприклад, розподіл оцінок за певною шкалою.

У процесі розробки модуля генерації звітів було використано бібліотеку DinkToPdf, яка забезпечує конвертацію HTML-контенту у формат PDF. Це рішення було обране через його гнучкість та можливість використовувати повноцінну HTML-розмітку разом зі стилями CSS, що дозволило реалізувати структуровані та візуально привабливі звіти з високою якістю відображення. Формування звіту реалізовано на основі HTML-шаблону, який динамічно заповнюється даними та конвертується у PDF, що підходить для завантаження або друку.

Для управління версіями коду використано Git – розподілену систему контролю версій. Git дозволяє відстежувати зміни в коді, створювати гілки для паралельної роботи над різними функціями (наприклад, розроблення фронтенду та

бекенду), а також зберігати історію проєкту. У разі помилок розробники можуть легко повернутися до попередньої версії. Інтеграція з платформами, такими як GitHub чи GitLab, підтримує командну роботу, перегляд коду і налаштування різних процесів для автоматичного розгортання застосунку.

Висновок до розділу 4

Вибір технологій для розробки вебзастосунку оцінювання якості роботи співробітників ґрунтується на їхніх перевагах, підтримці спільноти та відповідності функціональним і нефункціональним вимогам проєкту. На клієнтській стороні використано Angular, TypeScript, SCSS що забезпечило створення інтуїтивно зрозумілого, адаптивного та функціонального інтерфейсу користувача з можливістю зміни стилів і розширення функціоналу.

Для серверної частини обрано стек ASP.NET Core, C#, EF Core та Microsoft SQL Server, який гарантує високу продуктивність, безпеку обробки даних, підтримку транзакцій та легкість інтеграції з іншими службами. Це дозволило реалізувати масштабовану архітектуру з чітким розділенням шарів.

Додаткові інструменти, такі як Bogus для генерації тестових даних, DinkToPdf для створення професійних PDF-звітів та бібліотека компонентів PrimeNG значно покращили процес розробки та тестування.

Такий набір технологій дозволяє легко розширювати функціонал у майбутньому відповідно до зростаючих потреб.

5 РОЗРОБЛЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

5.1 Опис функціональної моделі

Функціональна модель розроблюваної інформаційної системи передбачає побудову централізованого вебзастосунку, що стане інструментом для систематичного та об’єктивного оцінювання працівників у команді. У межах цього застосунку буде передбачено роботу з такими сутностями, як працівники, команди, оцінювачі, періоди оцінювання, а також набір метрик, за якими буде оцінюватись співробітник. Система дозволить створювати та редагувати шаблони оцінювання для різних ролей, враховувати вагові коефіцієнти метрик для кожної ролі, фіксувати оцінки за визначеною шкалою та зберігати ці дані в централізованій базі.

Основний функціональний потік передбачає ініціацію періоду оцінювання, розподіл працівників між оцінювачами, введення оцінок і збереження їх у системі. Далі передбачається виконання обчислень – визначення середніх балів, зважених коефіцієнтів, відображення змін динаміки по відношенню до попередніх оцінювань. Додатково система забезпечуватиме генерацію звітів для ведення бухгалтерії та іншої корпоративної документації.

Уся логіка застосунку розділена за ролями: менеджер керує структурою організації, лідер команди – оцінює підлеглих, HR-менеджер може переглядати результати та формувати статистику. Завдяки цьому забезпечується гнучкість і масштабованість рішення. Система розробляється як вебзастосунок, що дозволяє мати доступ до неї з будь-якого пристрою в будь-який час, що важливо для компаній із віддаленим форматом роботи.

Детальніше із можливими варіантами використання системи, включаючи сценарії створення сесій оцінювання, введення оцінок, генерації звітів та рекомендацій, а також управління ролями і командами, можна ознайомитись на кресленику ІС11.040БАК.005 Д1.

Також, послідовність взаємодії між користувачем, клієнтською частиною та серверним API, що відображає процес оцінювання, можна подивитись на кресленику ІС11.040БАК.005 Д2.

5.2 Модель бази даних

Кожна сутність у системі має чітко визначені призначення, що сприяє інтеграції даних і підтримці функціональності застосунку.

Сутності системи та їхнє призначення:

– Users (користувачі системи) – відповідає за зберігання інформації про зареєстрованих користувачів, включаючи їхні унікальні ідентифікатори, електронні пошти, захешовані паролі та зв’язок із ролями, що визначають доступ до функцій;

– Employees (співробітники) – містить персональні дані про працівників, такі як ім’я, прізвище, номер телефону, дати народження та прийому на роботу, а також зв’язок із відповідною командою та користувацьким акаунтом;

– Teams (команди) – відображає організаційну структуру компаній, зберігаючи назви команд та ідентифікатори їхніх керівників, що є співробітниками. Кожна команда може мати лідера команди;

– Roles (ролі) – зберігає набір ролей у системі;

– Permissions (дозволи) – зберігає перелік доступних прав включаючи їхні назви та описи для визначення функціональних можливостей;

– Role_Permissions (ролі та дозволи) – забезпечує зв’язок між ролями та дозволами, визначаючи, які права надаються кожній ролі;

– KPIMetrics (ключові показники ефективності) – містить перелік ключових показників ефективності, які використовуються для оцінки працівників;

– RoleKPIs (ролі та метрики) – зв’язує ролі з відповідними метриками, вказуючи ваги, діапазони балів, описи та правила оцінки;

– EvaluationSessions (сесії оцінювання) – відображає періоди оцінювання, пов’язані з конкретними співробітниками, датами початку та завершення, а також необов’язковими даними, такими як зважений бал чи звіт у вигляді файлу;

– Evaluations (оцінки) – зберігає деталізовані оцінки за кожною метрикою в межах сесії, включаючи бали, коментарі оцінювача та зв’язок із сесією та метриками;

– EmployeeClasses (класи співробітників) – визначає категорії результативності роботи працівника з описами, рекомендованими діями та діапазонами балів;

– Recommendations (рекомендації) – містить персоналізовані рекомендації для співробітників, включаючи текст, дату створення та прапор видимості для працівника.

Структура таблиць бази даних наведений у таблиці 5.1.

Таблиця 5.1 – Структура таблиць бази даних

Таблиця	Атрибут	Значення
Users	user_id	Унікальний ідентифікатор користувача
	email	Електронна пошта користувача, унікальна для кожного користувача у системі
	password_hash	Захешований пароль
	role_id	Зовнішній ключ, що посилається на ідентифікатор ролі в таблиці Roles
	user_id	Унікальний ідентифікатор користувача
Employees	first_name	Ім’я співробітника
	last_name	Прізвище співробітника
	phone_number	Номер телефону співробітника
	employee_id	Унікальний ідентифікатор співробітника

Продовження таблиці 5.1

	user_id	Зовнішній ключ, що посилається на ідентифікатор користувача в Users, унікальний для кожного співробітника
	birth_date	Дата народження співробітника
	hire_date	Дата прийому співробітника на роботу
	team_id	Зовнішній ключ, що посилається на ідентифікатор команди в Teams
	avatar	Бінарний файл із фотографією співробітника (опціонально)
Teams	team_id	Унікальний ідентифікатор команди
	team_name	Назва команди
	team_lead_id	Зовнішній ключ, що посилається на ідентифікатор керівника в Employees
Roles	role_id	Унікальний ідентифікатор ролі
	role_name	Назва ролі

Продовження таблиці 5.1

Permissions	permission_id	Унікальний ідентифікатор дозволу
	permission_name	Назва дозволу
	description	Текстовий опис дозволу
Role_Permissions	role_id	Зовнішній ключ, що посилається на ідентифікатор ролі в Roles
	permission_id	Зовнішній ключ, що посилається на ідентифікатор дозволу в Permissions
KPIMetrics	kpi_id	Унікальний ідентифікатор метрики
	kpi_name	Назва метрики (наприклад, "якість коду"), унікальна
RoleKPIs	role_id	Зовнішній ключ, що посилається на ідентифікатор ролі в Roles
	kpi_id	Зовнішній ключ, що посилається на ідентифікатор метрики в KPIMetrics
	weight	Вага метрики
	min_score	Мінімальний можливий бал для метрики
	max_score	Максимальний можливий бал для метрики

Продовження таблиці 5.1

	is_allowed_to_evaluate_except_lead	Прапор, що вказує, чи може оцінювати не тільки лідер
	score_range_description	Опис діапазону балів
EvaluationSessions	session_id	Унікальний ідентифікатор сесії оцінювання
	employee_id	Зовнішній ключ, що посилається на ідентифікатор співробітника в Employees
	class_id	Зовнішній ключ, що посилається на ідентифікатор класу в EmployeeClasses (опціонально)
	start_date	Дата початку сесії оцінювання
	end_date	Дата завершення сесії оцінювання
	evaluation_finished_date	Дата завершення оцінювання (опціонально)
	weighted_score	Зважений бал сесії (опціонально)
	report_file	Бінарний файл зі звітом (опціонально)

Продовження таблиці 5.1

Evaluations	evaluation_id	Унікальний ідентифікатор оцінки
	evaluation_session_id	Зовнішній ключ, що посилається на ідентифікатор сесії в EvaluationSessions
	kpi_id	Зовнішній ключ, що посилається на ідентифікатор метрики в RoleKPIs
	role_id	Зовнішній ключ, що посилається на ідентифікатор ролі в RoleKPIs
	evaluator_id	Зовнішній ключ, що посилається на ідентифікатор оцінювача в Employees
	score	Бал, наданий за метрику
	comment	Текстовий коментар до оцінки (опціонально)
EmployeeClasses	class_id	Унікальний ідентифікатор класу
	class_name	Назва класу продуктивності (наприклад, "експерт"), унікальна
	description	Опис класу

Продовження таблиці 5.1

	recommended_actions	Рекомендовані дії для класу
	min_score	Мінімальний бал для класу
	max_score	Максимальний бал для класу
Recommendations	recommendation_id	Унікальний ідентифікатор рекомендації
	employee_id	Зовнішній ключ, що посилається на ідентифікатор співробітника в Employees
	recommendation_text	Текст рекомендації
	created_at	Дата створення рекомендації
	isVisibleToEmployee	Прапор, що вказує, чи видно рекомендацію співробітнику

Схему бази даних системи з усіма таблицями відображено на кресленику ІС11.040БАК.005 Д3.

5.3 Структура програмного забезпечення системи

Для реалізації системи оцінювання співробітників на основі KPI було обрано клієнт-серверну архітектуру, яка є однією з найбільш поширених моделей побудови сучасних вебзастосунків. Суть клієнт-серверної архітектури полягає в розділенні обов’язків між клієнтом та сервером. Клієнт ініціює запити, а сервер відповідає на них, опрацьовуючи інформацію, виконуючи необхідні обчислення та повертаючи результати. Програмне забезпечення включає

в себе наступні основні компоненти:

– клієнтська частина (Frontend) відповідає за інтерфейс користувача та взаємодію з кінцевим користувачем. На цьому рівні відображаються форми для оцінювання, перегляду звітів, керування ролями та метриками, а також інші функціональні елементи системи. Клієнтська частина здійснює запити до серверної частини через HTTP-протокол, отримує відповіді у форматі JSON та динамічно оновлює інтерфейс без необхідності перезавантаження сторінки;

– серверна частина реалізує бізнес-логіку застосунку. Вона відповідає за обробку запитів, перевірку прав доступу, збереження та обробку оцінок, обчислення середньозважених балів, формування рекомендацій, генерацію звітів у форматі PDF тощо. Важливою складовою серверної частини є система контролю ролей і дозволів, яка регламентує, хто має право створювати метрики, оцінювати працівників або переглядати статистику;

– рівень бази даних відповідає за зберігання усієї інформації, необхідної для функціонування системи. Це включає: дані про користувачів (профілі, ролі, команди), список метрик, результати оцінювання, історію змін, а також згенеровані звіти. База даних побудована за реляційною моделлю, що дозволяє ефективно зберігати взаємозв’язані дані та реалізовувати запити будь-якої складності.

Компоненти системи взаємодіють між собою за допомогою чітко визначених інтерфейсів: клієнт надсилає HTTP-запити до API, сервер їх обробляє, у разі потреби звертається до бази даних і повертає результати клієнту. Така архітектура забезпечує модульність, тестованість і можливість повторного використання окремих частин, що полегшує масштабування системи.

Архітектура також враховує ключові аспекти безпеки: реалізовано автентифікацію, авторизацію та механізми обмеження доступу відповідно до ролей користувачів. Усі компоненти проєкту спроєктовано з урахуванням потенційної інтеграції з іншими сервісами, що надає рішенню гнучкості у подальшому розвитку.

Також, на кресленику ІС11.040БАК.005 Д4 представлена діаграма розгортання, яка описує розміщення компонентів системи на фізичних вузлах.

5.4 Архітектура серверної частини

5.4.1 Архітектурний стиль

Для побудови серверної частини застосунку було обрано підхід Clean Architecture [10], запропонований Робертом Мартіном. Цей архітектурний стиль дозволяє побудувати систему таким чином, щоб бізнес-логіка була ізольована від будь-яких зовнішніх залежностей, зокрема бази даних, фреймворків чи вебсерверів. Основна ідея полягає в інверсії залежностей: внутрішні шари не мають залежати від зовнішніх, а навпаки – зовнішні залежать від внутрішніх [11].

На рисунку 5.1 зображено структуру серверної частини додатку.

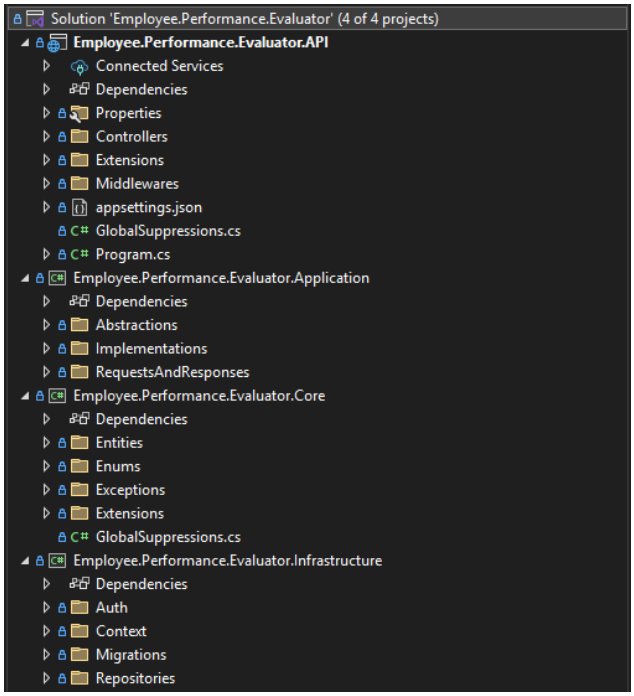


Рисунок 5.1 – Структура бекенд проєкту

У структурі рішення, представленій на рисунку 4.1, проєкт поділено на чотири основні рівні:

– Core – найнижчий шар, що містить сутності доменної моделі (Entities), переліки (Enums), винятки (Exceptions) та інші фундаментальні конструкції. Цей шар повністю ізольований від інфраструктури, не містить жодних зовнішніх залежностей і не має прив’язки до реалізації;

– Application – реалізує бізнес-логіку у вигляді сервісів, інтерфейсів, а також моделей запитів і відповідей. У цьому шарі описуються сценарії використання системи, логіка оцінювання, генерації рекомендацій тощо. Шар залежить лише від Core і містить реалізації, що використовуються API або інфраструктурним рівнем;

– Infrastructure – відповідає за реалізацію інтерфейсів доступу до зовнішніх ресурсів, зокрема до бази даних (через Entity Framework Core), системи автентифікації, а також реалізує сервіси, які залежні від зовнішніх технологій. Шар залежить від Application, але не впливає на нього. Таким чином, інфраструктура є змінною частиною, яку за потреби можна замінити без впливу на бізнес-логіку;

– API – верхній шар, який відповідає за обробку HTTP-запитів. Контролери, реалізовані в цьому шарі, приймають запити з клієнта, викликають відповідні сервіси з шару Application, формують і повертають відповіді. Його основна функція – бути "входом" у систему, при цьому він не містить бізнес-логіки.

На відміну від традиційної N-Layer або N-Tier архітектури, де шари часто мають спрямовані залежності один від одного, Clean Architecture забезпечує інверсію залежностей: інфраструктура залежить від бізнес-логіки, а не навпаки. У N-Tier підході бізнес-логіка часто прив’язана до конкретних реалізацій, що знижує гнучкість і ускладнює модульне тестування. У Clean Architecture кожен шар має чітку відповідальність, а взаємодія здійснюється через абстракції.

На рисунку 5.2 зображено архітектурний підхід Clean Architecture.

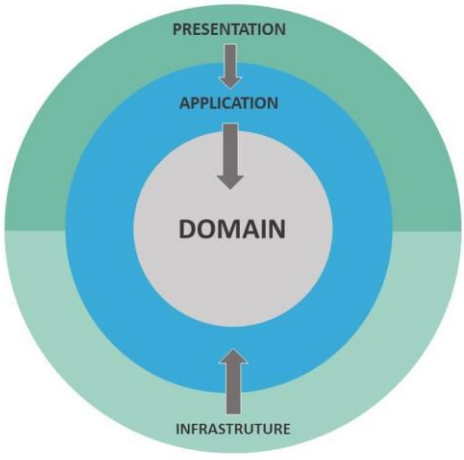


Рисунок 5.2 – Архітектурний підхід Clean Architecture

Отже основними перевагами Clean Architecture є:

– ізоляція бізнес-логіки від інфраструктури;

– зменшення зв’язності між компонентами;

– полегшене модульне тестування;

– гнучкість щодо зміни бази даних, зовнішніх API або технологій.

5.4.2 Специфікація класів

У даній системі реалізовано чітку багаторівневу архітектуру, яка дотримується принципів розділення відповідальностей. Для прикладу розглянуто контролер, відповідальний за роботу з оцінюваннями – EvaluationsController.

Контролер реалізує такі основні функції:

– GetAllBySessionIdAsync – повертає всі оцінки, що належать до вказаної сесії оцінювання. Дані включають інформацію про KPI, оцінюваного та оцінювача.

– CreateEvaluationAsync – приймає запит на створення оцінки. Контролер перевіряє права доступу та передає дані на обробку в сервісний рівень.

Контролер не містить бізнес-логіки, а делегує її виконання відповідному сервісу. Додатково, через атрибути (наприклад, [HasPermission(...)]), здійснюється перевірка ролей та дозволів користувача, що ініціює запит, див. рисунок 5.3.

```
[Route("api/[controller]")]
[ApiController]
1 reference
public class EvaluationsController(
    ILogger<EvaluationsController> logger,
    IEvaluationsService evaluationsService) : ControllerBase
{
    [HttpGet("session/{sessionId}")]
    [HasPermission(UserPermission.ViewAllEvaluations)]
    [ProducesResponseType(typeof(IEnumerable<EvaluationViewModel>), StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    [ProducesResponseType(StatusCodes.Status500InternalServerError)]
    0 references
    public async Task<IActionResult> GetAllBySessionIdAsync(int sessionId, CancellationToken cancellationToken)
    {
        try
        {
            var evaluations = await evaluationsService.GetAllBySessionIdAsync(sessionId, cancellationToken);

            if (evaluations == null || evaluations.Count == 0)
            {
                return NoContent();
            }

            return Ok(evaluations);
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "Error occurred while getting evaluations.");
            return StatusCode((int)HttpStatusCode.InternalServerError, ex);
        }
    }
```

Рисунок 5.3 – Частина коду класу EvaluationsController

Після обробки запиту контролером, дані передаються до сервісного шару, зокрема до реалізації інтерфейсу IEvaluationsService, див. рисунок 5.4.

```
3 references
public interface IEvaluationsService
{
    2 references
    Task<List<EvaluationViewModel>> GetAllBySessionIdAsync(int sessionId, CancellationToken cancellationToken);

    2 references
    Task CreateEvaluationAsync(AddEvaluationRequest addEvaluationRequest, CancellationToken cancellationToken);
}
```

Рисунок 5.4 – Код інтерфейсу IEvaluationsService

У цьому шарі відбувається основна бізнес-логіка, включаючи перевірку прав доступу, валідацію даних та взаємодію з репозиторіями. Наприклад, метод CreateEvaluationAsync виконує такі кроки:

– отримує сесію оцінювання за ідентифікатором та перевіряє її наявність;

– перевіряє відповідність ролі оцінюваного працівника;

– отримує інформацію про поточного користувача та його роль;

– отримує доступні KPI для оцінювання та перевіряє їх відповідність;

– створює нову оцінку та зберігає її через відповідний репозиторій.

Сервісний шар взаємодіє з репозиторіями, які реалізують доступ до бази даних, див. рисунок 5.5.

```
1 reference
public class EvaluationsService(
    IUserGetter userGetter,
    IEmployeesRepository employeesRepository,
    IRoleKPIsService roleKPIsService,
    IEvaluationSessionsRepository evaluationSessionsRepository,
    IEvaluationsRepository evaluationsRepository) : IEvaluationsService
{
    2 references
    public async Task<List<EvaluationViewModel>> GetAllBySessionIdAsync(int sessionId, CancellationToken cancellationToken)
    {
        var evaluations = await evaluationsRepository.GetAllBySessionIdAsync(sessionId, cancellationToken);

        return [.. evaluations.Select(EvaluationViewModel.MapFromDbModel)];
    }

    2 references
    public async Task CreateEvaluationAsync(AddEvaluationRequest addEvaluationRequest, CancellationToken cancellationToken)
    {
        var evaluationSession = await evaluationSessionsRepository.GetByIdWithDetailsAsync(
            addEvaluationRequest.EvaluationSessionId, cancellationToken);
        if (evaluationSession == null)
        {
            throw new InvalidOperationException($"Evaluation session with Id={addEvaluationRequest.EvaluationSessionId} not found.");
        }
        if (evaluationSession.EndDate < DateTimeOffset.Now)
        {
            throw new InvalidOperationException($"Evaluation session with Id={evaluationSession.Id} has already ended.");
        }
        if (evaluationSession.Employee!.User!.RoleId != addEvaluationRequest.RoleId)
        {
            throw new InvalidOperationException($"Employee with Id={evaluationSession.Employee.Id} does not have RoleId={addEvaluationRequest.RoleId}.");
        }

        var evaluatorUser = userGetter.GetCurrentUserOrThrow();
        var evaluatorEmployee = await employeesRepository.GetByUserIdAsync(evaluatorUser.Id, cancellationToken);
```

Рисунок 5.5 – Частина коду реалізації IEvaluationsService

Репозиторії, такі як EvaluationsRepository, реалізують інтерфейси, визначені в іншому шарі застосунку, забезпечуючи абстракцію від конкретної реалізації зберігання даних.

Наприклад, метод GetAllBySessionIdAsync у EvaluationsRepository виконує запит до бази даних з використанням Entity Framework Core, включаючи необхідні зв'язки між сутностями, див. рисунок 5.6.

```
2 references
public class EvaluationsRepository : BaseRepository<Evaluation>, IEvaluationsRepository
{
    0 references
    public EvaluationsRepository(AppDbContext context) : base(context)
    {
    }

    3 references
    public Task<List<Evaluation>> GetAllBySessionIdAsync(int sessionId, CancellationToken cancellationToken)
    {
        return GetAllWithDetailsInternal()
            .AsNoTracking()
            .Where(e => e.EvaluationSessionId == sessionId)
            .ToListAsync(cancellationToken);
    }

    1 reference
    private IQueryable<Evaluation> GetAllWithDetailsInternal()
    {
        return _dbSet
            .Include(e => e.Evaluator)
                .ThenInclude(ev => ev!.User)
                    .ThenInclude(u => u!.Role)
            .Include(e => e.RoleKpi)
                .ThenInclude(rk => rk!.Role)
            .Include(e => e.RoleKpi)
                .ThenInclude(rk => rk!.KpiMetric);
    }
}
```

Рисунок 5.6 – Код реалізації класу EvaluationsRepository

Базовий репозиторій BaseRepository<T> надає загальні методи для роботи з будь-якою сутністю, такі як GetByIdAsync, AddAsync, Update, Delete та SaveChangesAsync, див. рисунок 5.7. Це забезпечує повторне використання коду та спрощує реалізацію конкретних репозиторіїв. Крім того, методи підтримують асинхронне виконання з підтримкою токенів скасування (CancellationToken), що дозволяє ефективно керувати ресурсами в багатопоточних середовищах.

Також реалізовано метод ExistsAsync для перевірки наявності запису за ідентифікатором в базі даних, що підвищує зручність розробки при реалізації бізнес-логіки.

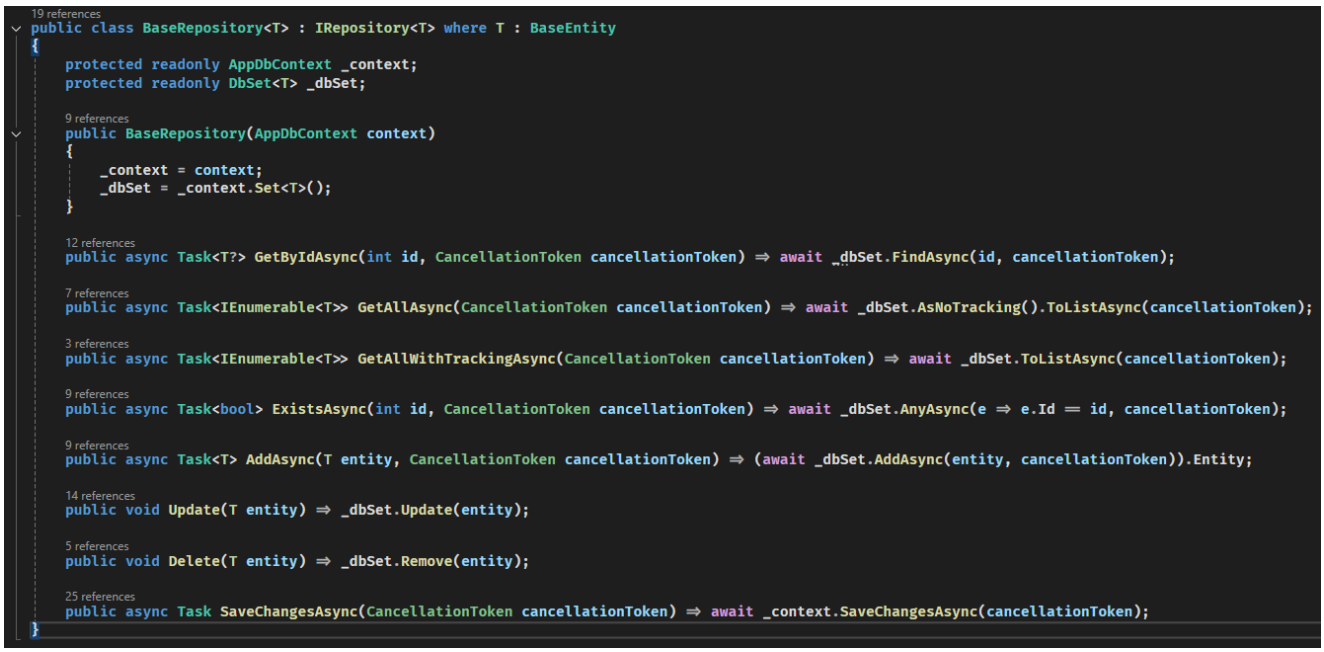


Рисунок 5.7 – Код реалізації базового репозиторію

5.4.3 Реалізація доступів та дозволів

Реалізація системи доступу до окремих функцій або ресурсів здійснюється за допомогою політик авторизації на основі призначених дозволів (permissions), які задаються кожному користувачу. Це дозволяє забезпечити контрольований та гнучкий доступ до можливостей системи залежно від ролі користувача та його прав.

Основним компонентом, який відповідає за застосування політики доступу, є атрибут HasPermissionAttribute, див. рисунок 5.8. Цей атрибут призначається до методів контролерів або класів, які потребують перевірки дозволу, і забезпечує авторизацію на основі політики, яка відповідає певному значенню UserPermission.

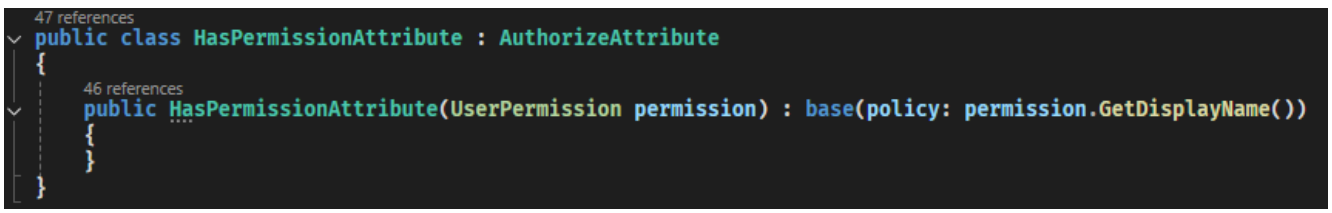


Рисунок 5.8 – Код реалізації атрибута HasPermissionAttribute

Реалізація самої перевірки політики виконується через обробник PermissionAuthorizationHandler, див. рисунок 5.9.

```
2 references
public class PermissionAuthorizationHandler(ILogger<PermissionAuthorizationHandler> logger)
    : AuthorizationHandler<PermissionRequirement>
{
    0 references
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context, PermissionRequirement requirement)
    {
        string? userId = context.User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier)?.Value;

        if (!int.TryParse(userId, out int id))
        {
            logger.LogWarning("UserId claim is missing or invalid. User Id={UserId}", userId);
            context.Fail();

            return Task.CompletedTask;
        }

        var userPermissions = context.User.Claims
            .Where(c => c.Type == CustomClaims.PermissionsClaim)
            .Select(c => c.Value)
            .ToHashSet();

        if (userPermissions.Contains(requirement.Permission))
        {
            context.Succeed(requirement);
        }
        else
        {
            logger.LogWarning("User with Id={UserId} does not have permission {Permission} to access this resource.",
                id, requirement.Permission);

            context.Fail();
        }

        return Task.CompletedTask;
    }
}
```

Рисунок 5.9 – Код реалізації PermissionAuthorizationHandler

Цей клас реалізує стандартний механізм авторизації в ASP.NET Core і перевіряє, чи має поточний користувач у своєму токені відповідний дозвіл. Обробник отримує список дозволів з клеймів користувача (Claims), зокрема з клейму PermissionsClaim, і порівнює їх із необхідним значенням, заданим у політиці (Authorization Policy).

Отже, використання цього технічогого рішення дозволяє централізовано керувати дозволами у вигляді політик, і використовувати їх у вигляді декларативних атрибутів у коді, тому такий підхід підвищує читабельність і підтримуваність коду, адже логіка авторизації відокремлена від бізнес-логіки.

Список дозволів додається до токена під час його генерації в класі JwtProvider, див. рисунок 5.10. Під час створення JWT-токена в методі GenerateTokenAsync система додає в токен всі дозволи користувача, які отримуються з бази даних за допомогою IUsersRepository. Завдяки цьому токен

містить всю необхідну інформацію для подальшої авторизації без необхідності звернення до бази при кожному запиті.

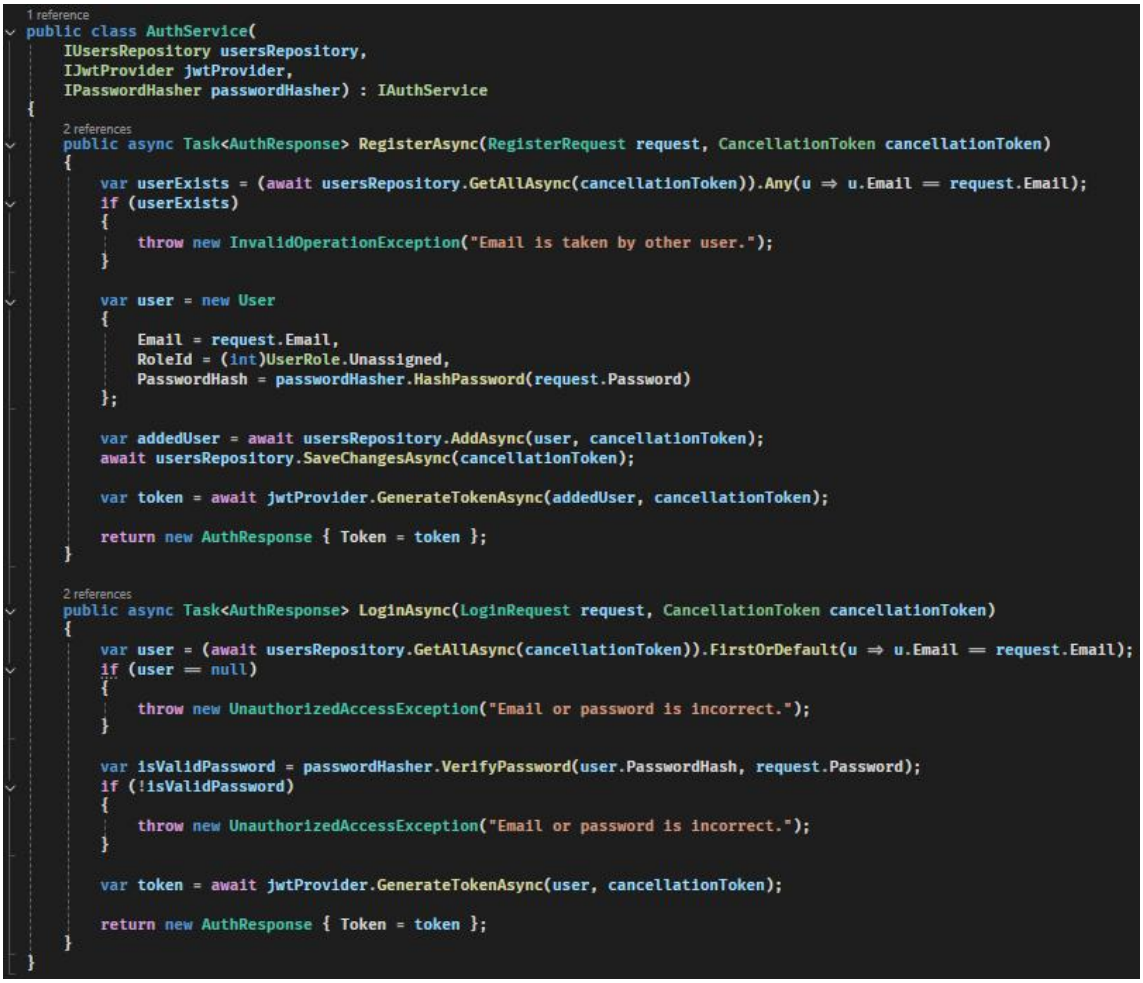
```
public class JwtProvider : IJwtProvider
{
    private readonly JwtOptions _jwtOptions;
    private readonly IUsersRepository _usersRepository;


    public JwtProvider(IOptions<JwtOptions> jwtOptions, IUsersRepository usersRepository)
    {
        _jwtOptions = jwtOptions.Value;
        _usersRepository = usersRepository;
    }


    public async Task<string> GenerateTokenAsync(User user, CancellationToken cancellationToken)
    {
        var claims = new List<Claim>
        {
            new(JwtRegisteredClaimNames.Sub, user.Id.ToString()),
            new(JwtRegisteredClaimNames.Name, user.Email),
        };

        var userPermissions = await _usersRepository.GetPermissionsAsync(user.Id, cancellationToken);
        foreach (var permission in userPermissions)
        {
            claims.Add(new Claim(CustomClaims.PermissionsClaim, permission.ToString()));
        }

        var secretKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_jwtOptions.Key));
        var signingCredentials = new SigningCredentials(secretKey, SecurityAlgorithms.HmacSha256);
        var expirationTimeInHours = _jwtOptions.ExpirationTimeInHours;
        var expires = DateTime.UtcNow.AddHours(expirationTimeInHours);

        var token = new JwtSecurityToken(
            _jwtOptions.Issuer,
            _jwtOptions.Audience,
            claims,
            expires: expires,
            signingCredentials: signingCredentials);

        return new JwtSecurityTokenHandler().WriteToken(token);
    }
}
```

Рисунок 5.10 – Код реалізації JwtProvider

Для отримання токена користувач повинен авторизуватися через API-контролер AuthController. Контролер містить методи Login та Register, які викликають відповідні методи сервісу IAuthService.

Безпосередньо логіку автентифікації реалізовано в класі AuthService, див. рисунок 5.11. Під час реєстрації новий користувач створюється з незаданою роллю, а його пароль хешується за допомогою PasswordHasher. Під час входу здійснюється перевірка наявності користувача та правильності введеного пароля.

У випадку успішної перевірки генерується JWT-токен, який дозволяє клієнту надалі взаємодіяти з захищеними ресурсами. Завдяки такому підходу вдається підтримувати надійний рівень безпеки користувачів, а також централізовано керувати правами доступу та авторизацією.



```
1 reference
public class AuthService(
    IUsersRepository usersRepository,
    IJwtProvider jwtProvider,
    IPasswordHasher passwordHasher) : IAuthService
{
    2 references
    public async Task<AuthResponse> RegisterAsync(RegisterRequest request, CancellationToken cancellationToken)
    {
        var userExists = (await usersRepository.GetAllAsync(cancellationToken)).Any(u => u.Email == request.Email);
        if (userExists)
        {
            throw new InvalidOperationException("Email is taken by other user.");
        }

        var user = new User
        {
            Email = request.Email,
            RoleId = (int)UserRole.Unassigned,
            PasswordHash = passwordHasher.HashPassword(request.Password)
        };

        var addedUser = await usersRepository.AddAsync(user, cancellationToken);
        await usersRepository.SaveChangesAsync(cancellationToken);

        var token = await jwtProvider.GenerateTokenAsync(addedUser, cancellationToken);

        return new AuthResponse { Token = token };
    }

    2 references
    public async Task<AuthResponse> LoginAsync(LoginRequest request, CancellationToken cancellationToken)
    {
        var user = (await usersRepository.GetAllAsync(cancellationToken)).FirstOrDefault(u => u.Email == request.Email);
        if (user == null)
        {
            throw new UnauthorizedAccessException("Email or password is incorrect.");
        }

        var isValidPassword = passwordHasher.VerifyPassword(user.PasswordHash, request.Password);
        if (!isValidPassword)
        {
            throw new UnauthorizedAccessException("Email or password is incorrect.");
        }

        var token = await jwtProvider.GenerateTokenAsync(user, cancellationToken);

        return new AuthResponse { Token = token };
    }
}
```

Рисунок 5.11 – Код реалізації AuthService

За хешування паролів відповідає окремий сервіс PasswordHasher. Він використовує алгоритм PBKDF2 з HMACSHA256 для безпечного хешування, що забезпечує стійкість до атак перебором.

5.5 Архітектура клієнтської частини

Клієнтська частина застосунку реалізована з використанням Angular – сучасного фреймворку, який дозволяє створювати масштабовані односторінкові додатки. Архітектура застосунку організована за модульним принципом, що забезпечує гнучке розділення функціональності, повторне використання компонентів, а також зручність супроводу та масштабування.

Загальну структуру проєкту можна побачити на рисунку 5.12.

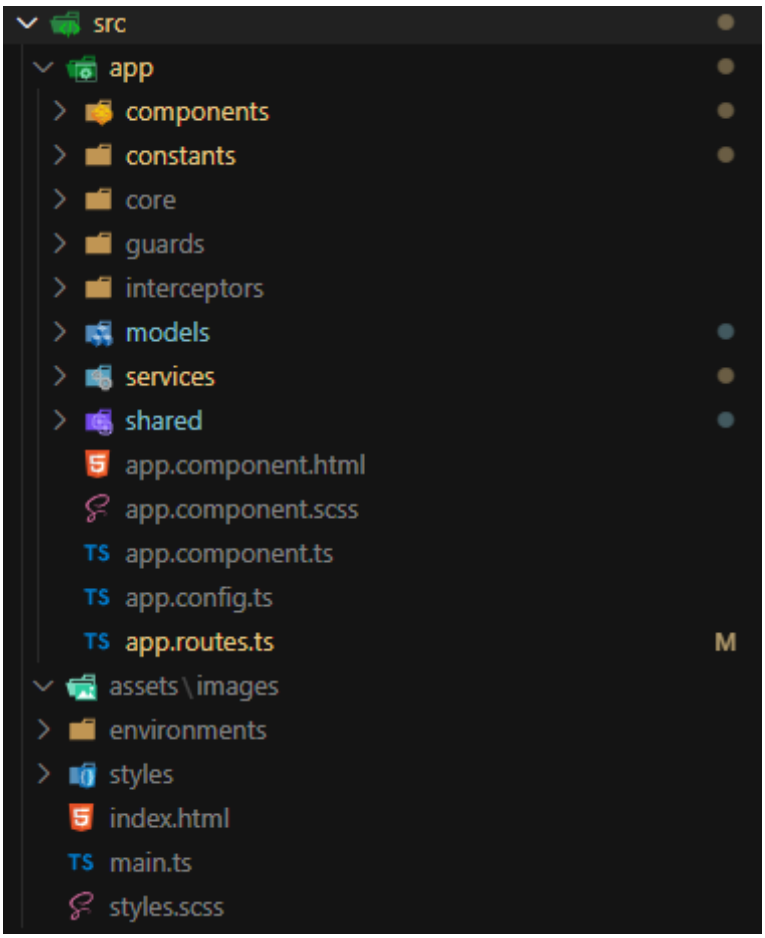


Рисунок 5.12 – Загальна структура проєкту

Уся логіка інтерфейсу згрупована в окрему директорію app, яка, у свою чергу, містить підкаталоги за функціональними доменами. Центральне місце займає папка components, у якій розміщені функціональні модулі застосунку – як-от інтерфейси для керування профілями співробітників, сесіями оцінювання, KPI-метриками, рекомендаціями тощо. У проєкті використовується підхід standalone-компонентів, що дозволяє кожному компоненту існувати незалежно без обов'язкового включення до окремого модуля. Такий підхід спрощує структуру коду, полегшує повторне використання компонентів і пришвидшує розробку.

Кожен функціональний розділ складається з набору незалежних компонентів, які самостійно імпортують усі необхідні залежності. Наприклад, папка evaluations відповідає за інтерфейс керування оцінюваннями співробітників і містить як візуальні елементи, так і логіку взаємодії з відповідними сервісами.

Структура такої директорії показана на рисунку 5.13.

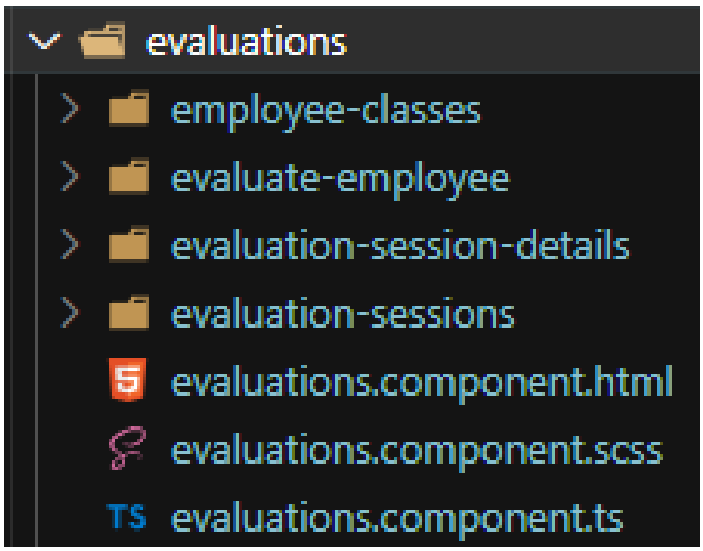


Рисунок 5.13 – Структура модуля оцінок

Крім основних компонентів, проєкт містить додаткові службові частини. Каталоги core та shared містять спільні сервіси, утиліти, компоненти та модулі, які використовуються у різних частинах застосунку.

Для реалізації механізмів авторизації та захисту маршрутів використовуються Angular guards, які розміщені в окремому каталозі guards. Вони перевіряють чи у користувач автентифікований та чи має відповідні дозволи. Каталог interceptors містить перехоплювачі запитів (HTTP-interceptors), що автоматично додають заголовки до запитів або обробляють помилки мережі. Усі моделі даних зібрані в директорії models, де оголошуються інтерфейси для типізації об’єктів, отриманих із серверної частини.

Логіка взаємодії з API винесена в окремі сервіси, що розміщуються у каталозі services. Кожен сервіс відповідає за певний домен (наприклад, EvaluationService або AuthService) та містить методи для виконання HTTP-запитів до відповідних контролерів на сервері.

Можна зробити висновок, що архітектура клієнтської частини побудована таким чином, щоб кожен модуль міг працювати незалежно від інших, дотримуючись принципів розділення обов’язків. Це дозволяє просто тестувати, супроводжувати та розширювати функціональність застосунку без потреби змінювати інші частини системи.

5.6 Настанова користувача

Коли співробітник відкрив застосунок за посиланням, він автоматично переходить до сторінки авторизації, яка зображена на рисунку 5.14. Щоб зареєструватись, потрібно ввести свою електронну пошту та пароль двічі.

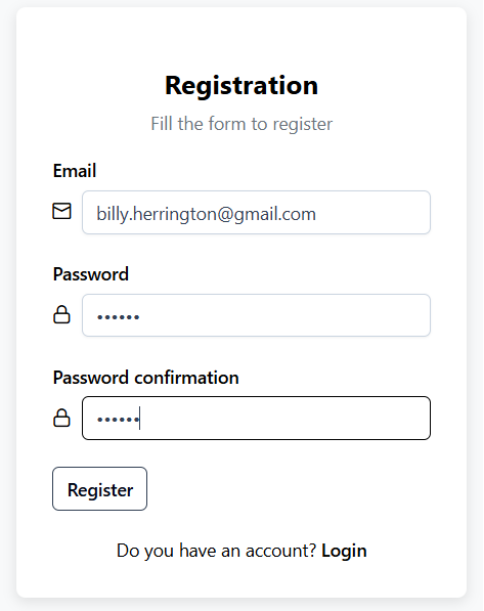


Рисунок 5.14 – Сторінка авторизації

Далі співробітник з відповідним доступом – менеджер чи адміністратор – має створити профіль для нового користувача у системі. Для цього потрібно вибрати «New Users» що знаходиться у вкладці «Administration» у меню зліва, та на натиснути «Create Profile» для відповідного співробітника, див. рисунок 5.15.

Після натискання кнопки «Create Profile» відкриється форма створення профілю, де необхідно вказати основну інформацію про співробітника, таку як ім’я, прізвище, посада та роль у компанії. Присвоєння ролі є обов’язковим етапом, адже від неї залежить рівень доступу користувача до функціоналу системи. Після заповнення всіх обов’язкових полів треба підтвердити створення профілю, натиснувши кнопку «Save», див. рисунок 5.16. Успішно створений профіль буде автоматично переміщено до загального списку співробітників. Таким чином, новий користувач отримує доступ до системи відповідно до призначеної ролі.

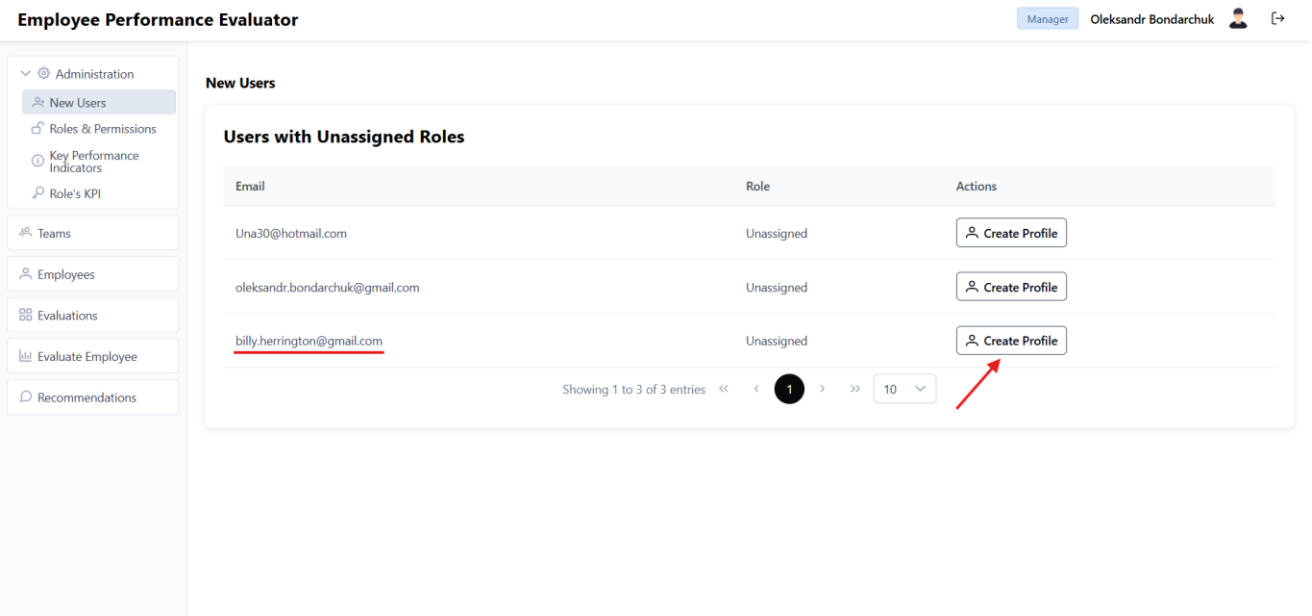


Рисунок 5.15 – Сторінка додавання зареєстрованих співробітників до системи

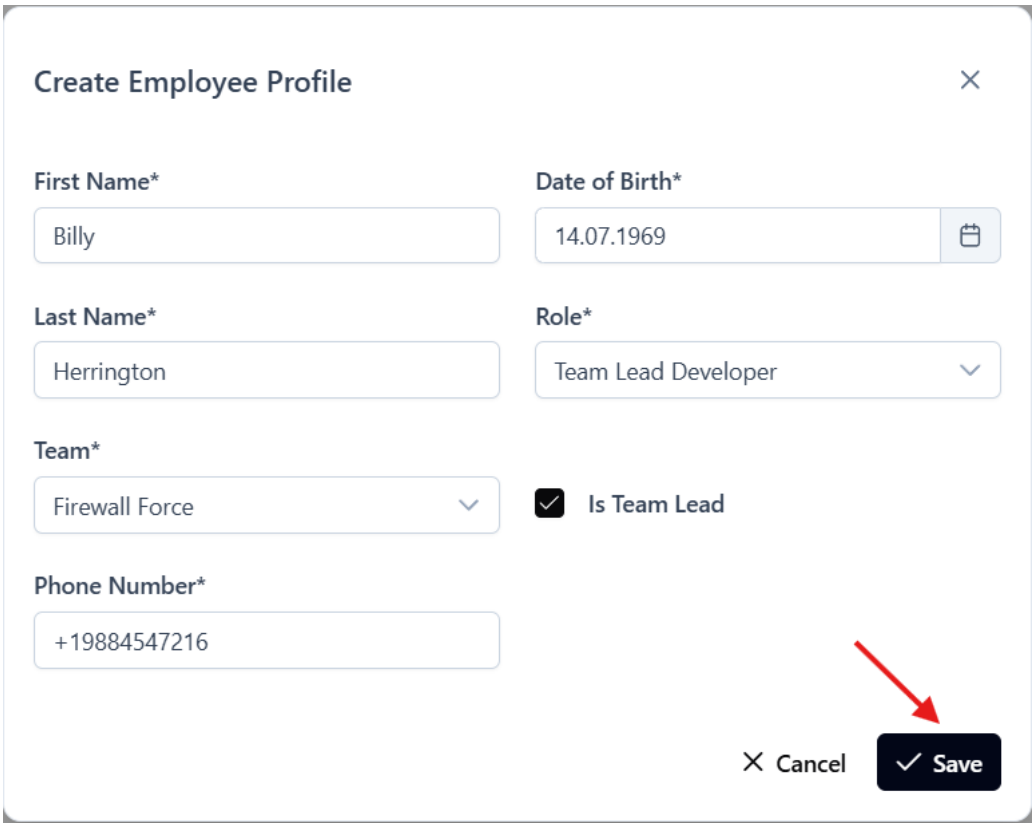


Рисунок 5.16 – Форма створення профілю співробітника

Далі можна переглянути інформацію про співробітника. Для цього варто вибрати пункт з меню «Employees», знайти необхідну людину та натиснути на її ім’я, див. рисунок 5.17.

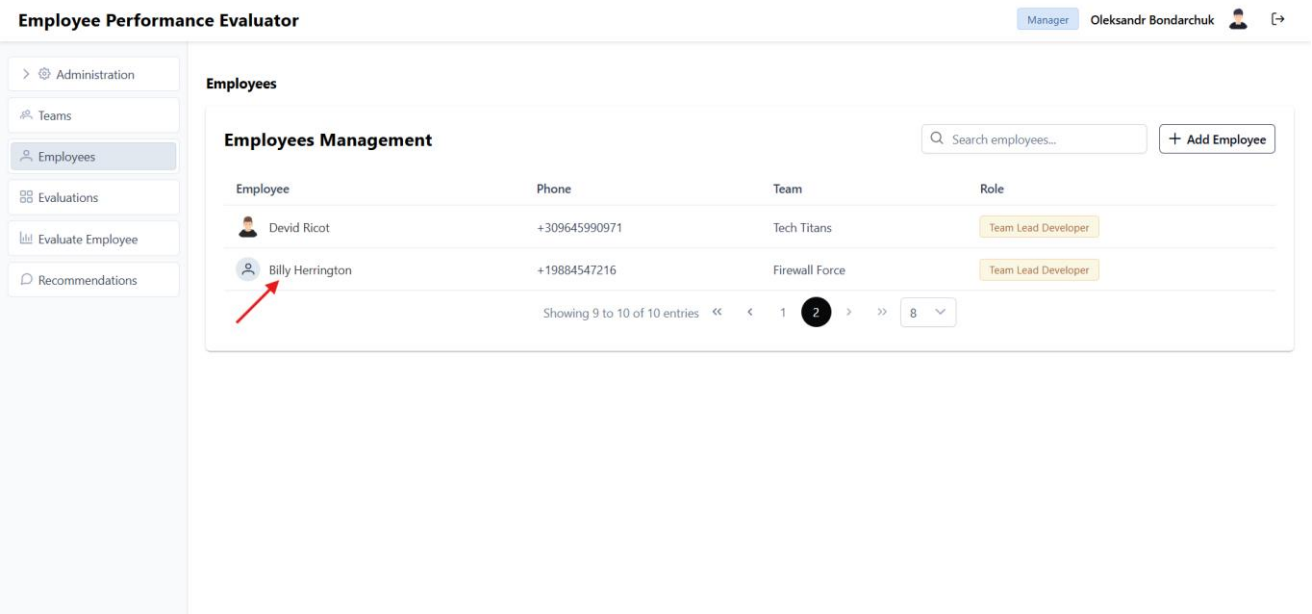


Рисунок 5.17 – Сторінка з усіма співробітниками

Перейшовши на профіль співробітника, користувач з відповідними правами може редагувати його особисту інформацію, натиснувши на «Edit Profile». Додатково він може додати йому фото, навівшись на сірий круг з іконкою користувача розташований біля імені співробітника, див. рисунок 5.18. Варто зауважити що функціонал редагування особистих даних прихований для користувачів без відповідного дозволу.

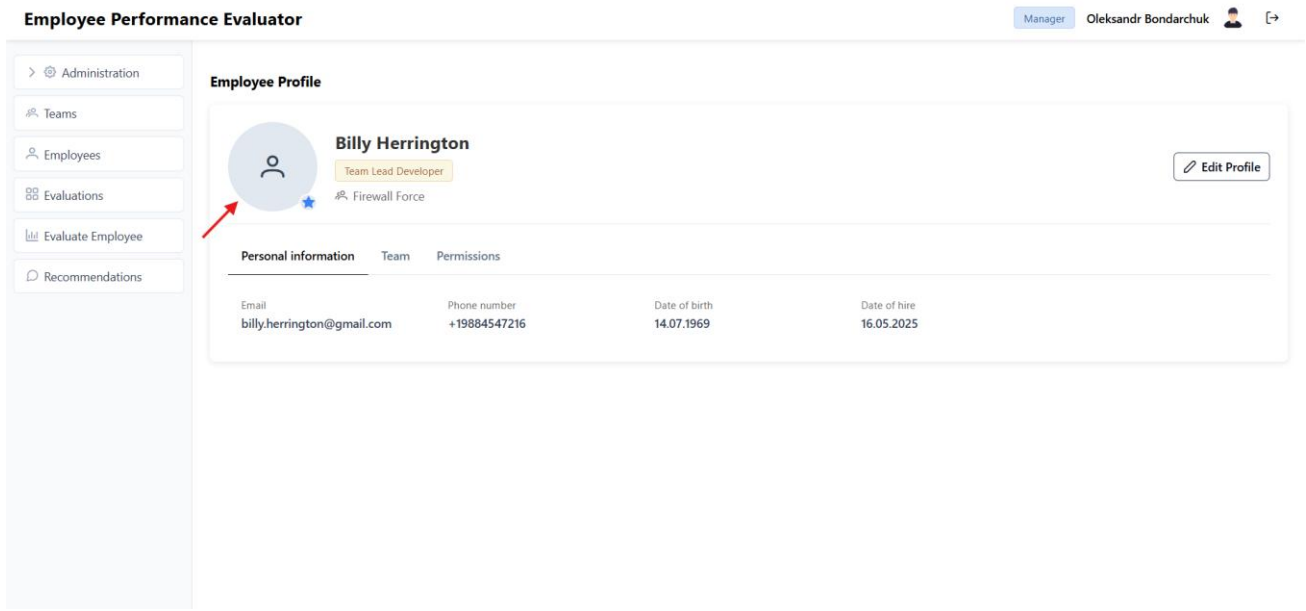


Рисунок 5.18 – Сторінка профілю співробітника з можливістю редагування

Система також надає ряд інших адмініструючих функцій. Менеджер системи може налаштовувати ролі, для цього потрібно вибрати «Roles & Permission» що знаходиться у вкладці «Administration» див. рисунок 5.19.

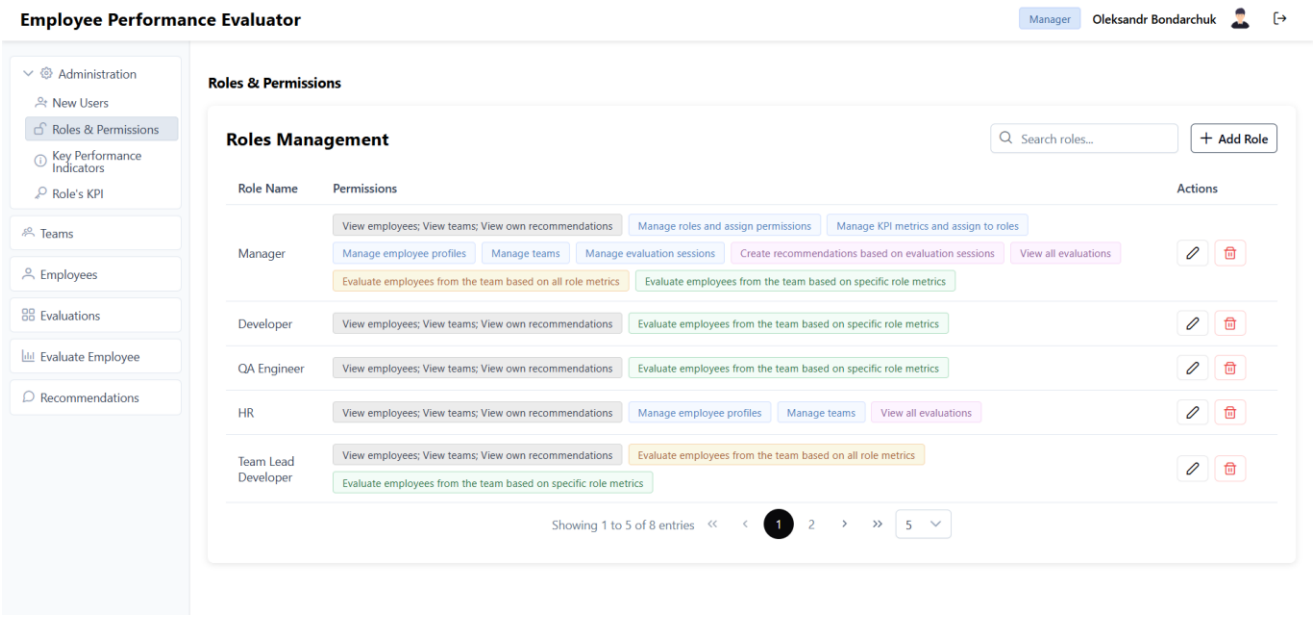


Рисунок 5.19 – Сторінка керування ролями та їх дозволами у системі

Також доступні для редагування ключові показники ефективності та їх призначення до ролі. Для цього необхідно вибрати опції «Key Performance Indicators» та «Role’s KPI» відповідно, див. рисунки 5.20 та 5.21.

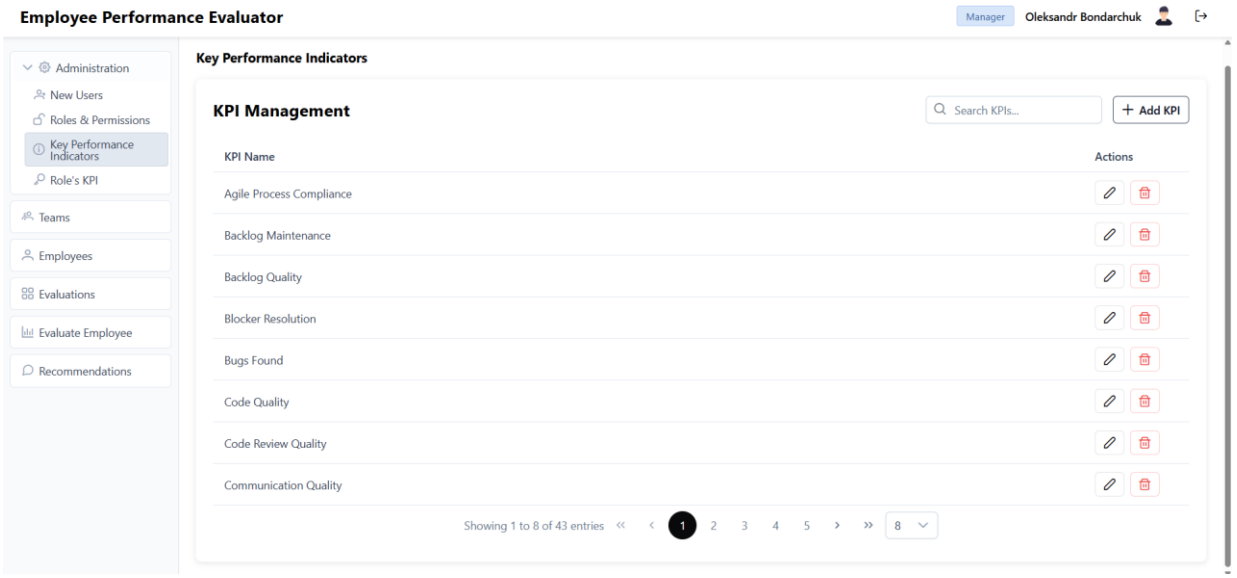


Рисунок 5.20 – Сторінка керування ключовими показниками ефективності

Варто зазначити, що функціонал призначення KPI до певної ролі доступний тільки тим користувачам, у кого є дозвіл і на керування ролями, і на керування ключовими показниками ефективності.

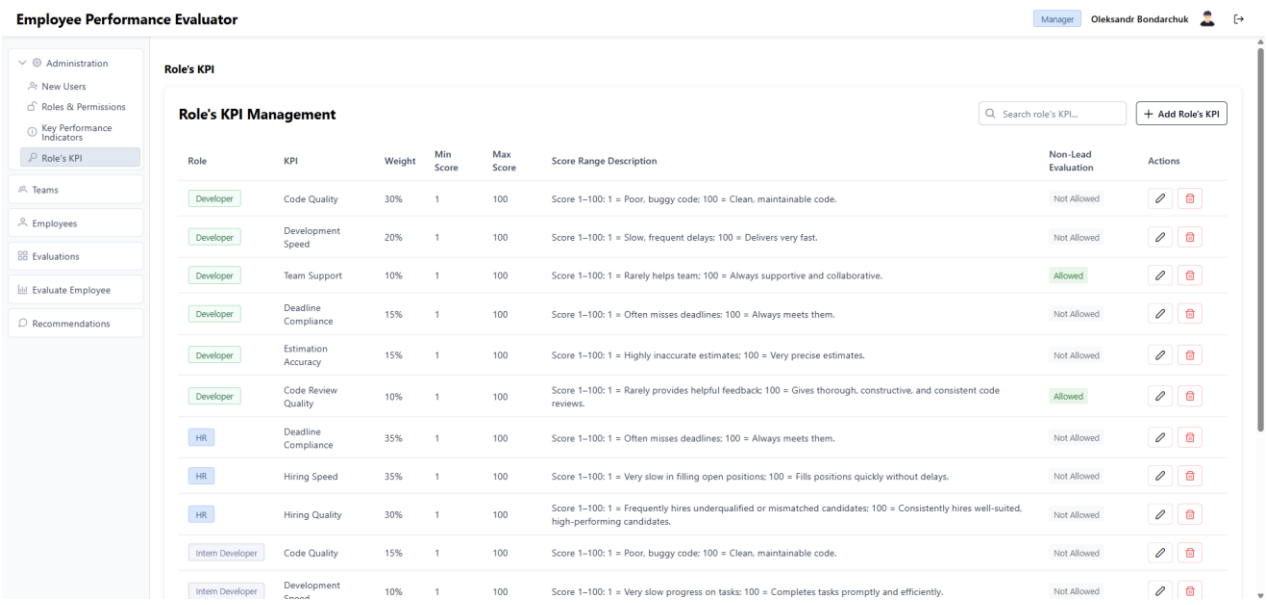


Рисунок 5.21 – Сторінка призначення KPI до певної ролі

Для керування командами, система надає функціонал, доступний у меню зліва, натиснувши на «Teams». Співробітники з певним дозволом можуть додавати, редагувати та видаляти команди, див. рисунок 5.22.

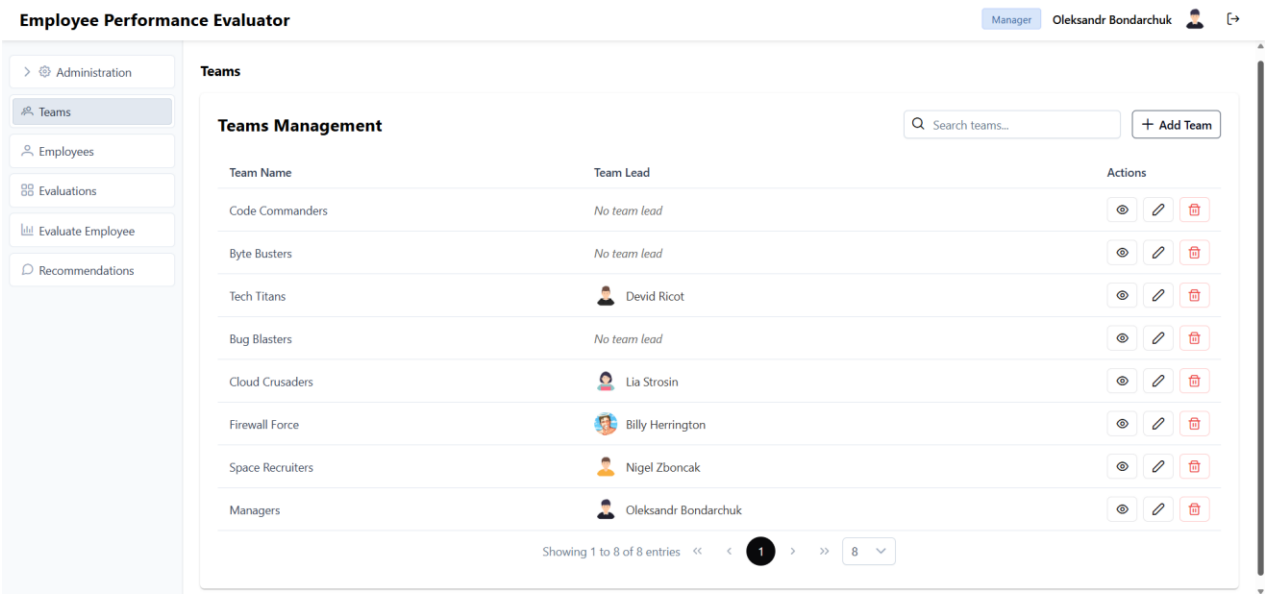


Рисунок 5.22 – Сторінка керування командами

Інші користувачі без цього дозволу можуть лише переглядати список команд та працівників, як знадяться в тій чи іншій команді.

Користувачі з відповідним доступом можуть переглядати сесії оцінювання працівників. Для цього потрібно вибрати «Evaluations» на вкладці меню, див. рисунок 5.23. На цій сторінці доступний перегляд оцінок усіх співробітників з можливістю фільтрації та створення нової сесії оцінювання, натиснувши на «Start New Evaluation Session».

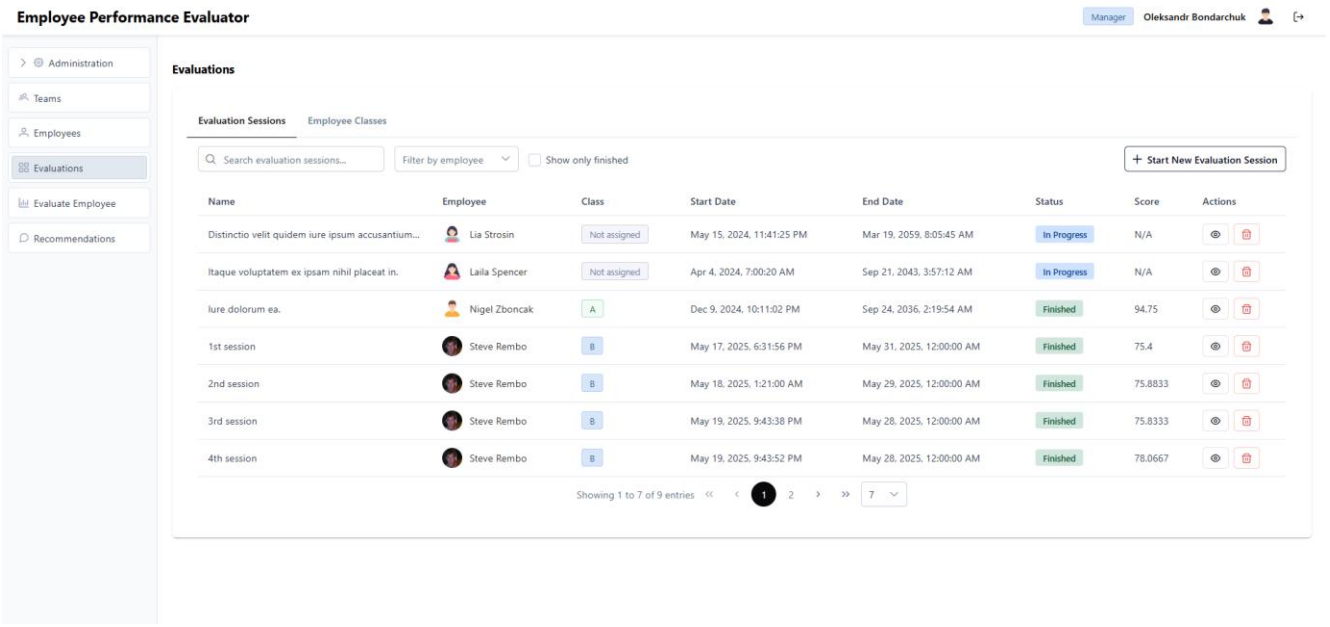


Рисунок 5.23 – Сторінка керування сесіями оцінювань співробітників

Щоб детально подивитись інформацію про сесію, потрібно натиснути на відповідну дію зі стовпця «Actions» у таблиці сесій оцінювань, та перейти на сторінку керування цією сесією, див. рисунок 5.24. На цій сторінці можна:

– подивитись дату початку та кінця сесії оцінювання, її статус та обчислений середньозважений бал на основі внесених оцінок;

– подивитись оцінку кожної метрики (KPI);

– завантажити звіт натиснувши «Download Report», де будуть виведені усі оцінки за цю сесію та прогноз на наступний період (див. рисунок А.2 у додатку А);

– згенерувати рекомендацію;

– перейти на профіль оцінювача, натиснувши на ім’я співробітника.

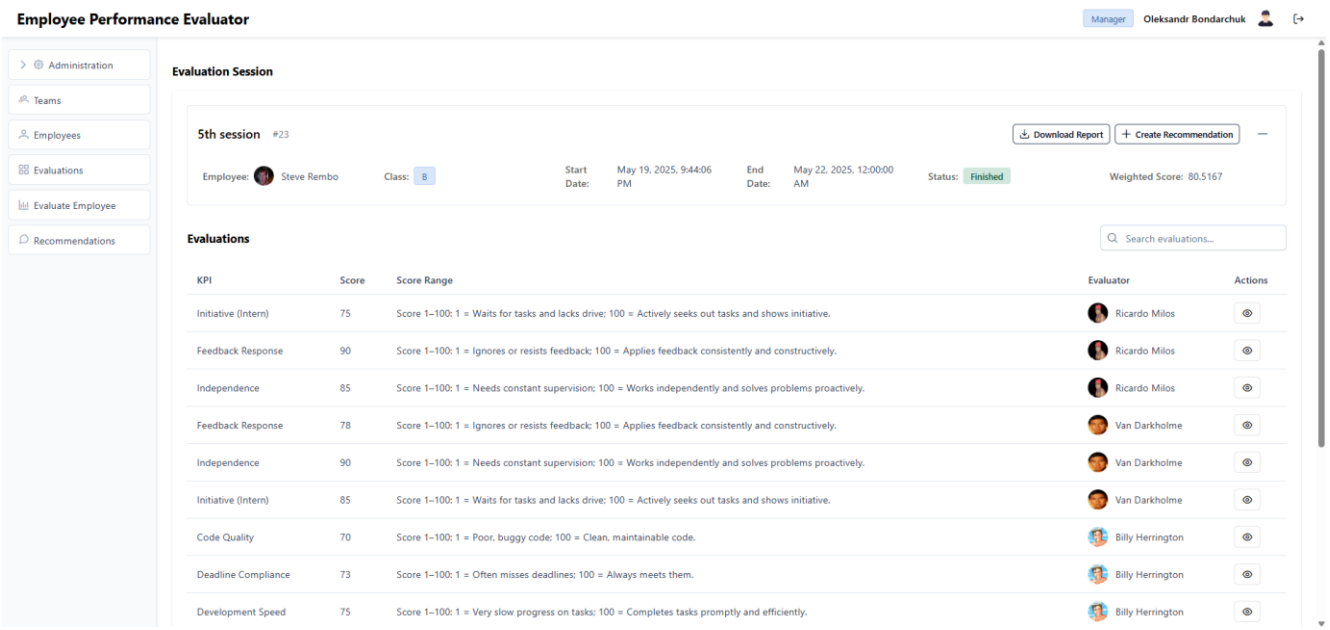


Рисунок 5.24 – Сторінка керування сесією оцінювання

Також потрібно враховувати, що починати, закінчувати та видаляти сесію оцінювання може тільки користувач з розширеним доступом до керування сесіями.

Детальний перегляд тієї чи іншої оцінки можливий, натиснувши на відповідну дію зі стовпця «Actions», див. рисунок 5.25.

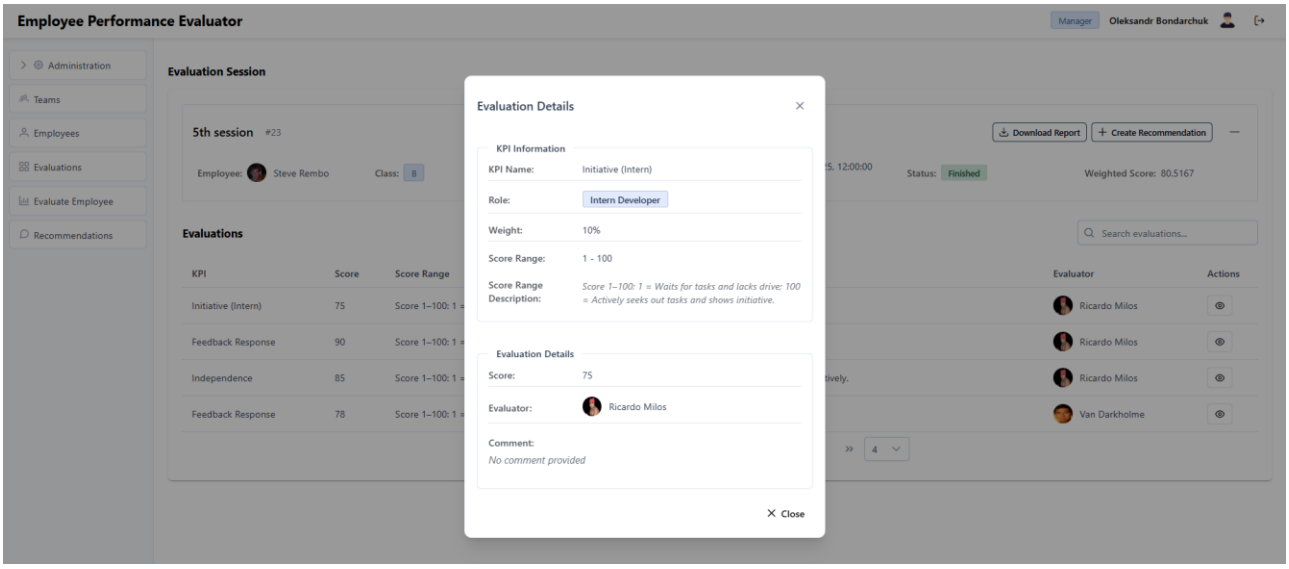


Рисунок 5.25 – Вікно перегляду оцінки певної метрики (KPI)

Щоб створити рекомендацію, потрібно натиснути на «Create Recommendation», див. рисунок 5.26.

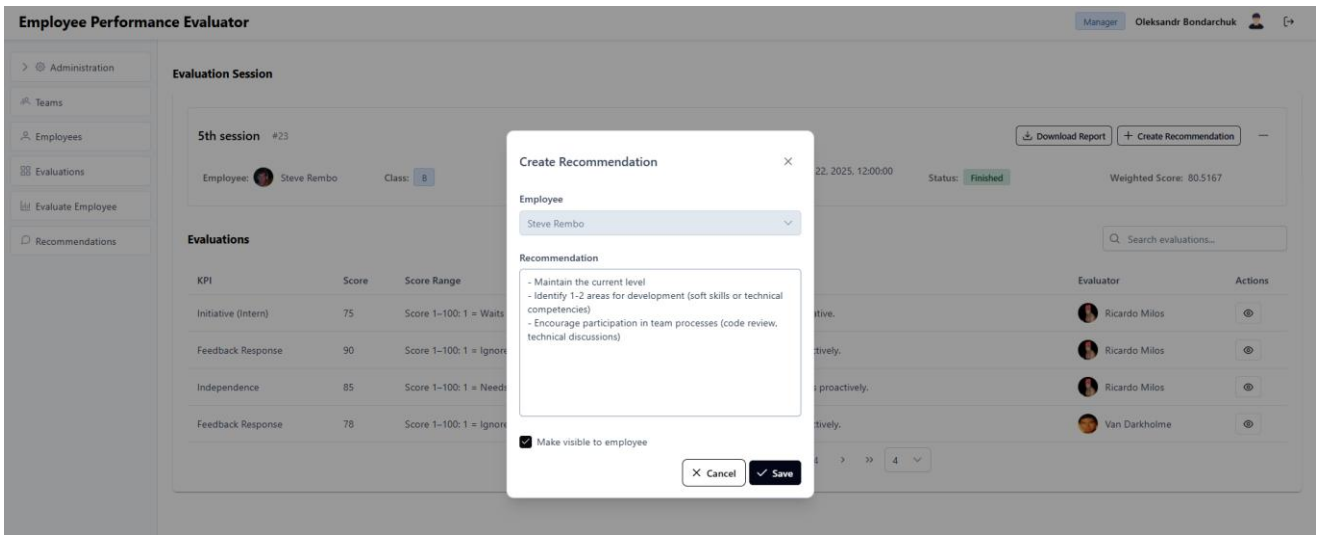


Рисунок 5.26 – Вікно створення рекомендації для співробітника

У наведеному прикладі працівнику рекомендується зберегти поточний рівень результативності, водночас визначити 1–2 напрями для подальшого розвитку — це можуть бути як м’які навички, так і технічні компетенції.

Такі поради допомагають співробітнику бачити зони росту та сприяють професійному розвитку. Опція «Make visible to employee» дає змогу зробити рекомендацію доступною для самого працівника у його інтерфейсі, які він зможе переглядати одразу після створення рекомендації.

Для оцінювання співробітника, користувач з відповідним доступом має перейти до сторінки «Evaluate Employee», обрати доступну сесію, метрику, та оцінити її за шкалою від 1 до 100, залишивши коментар до оцінки, якщо це потрібно, та натиснути «Submit», див. рисунки 5.27 - 5.29.

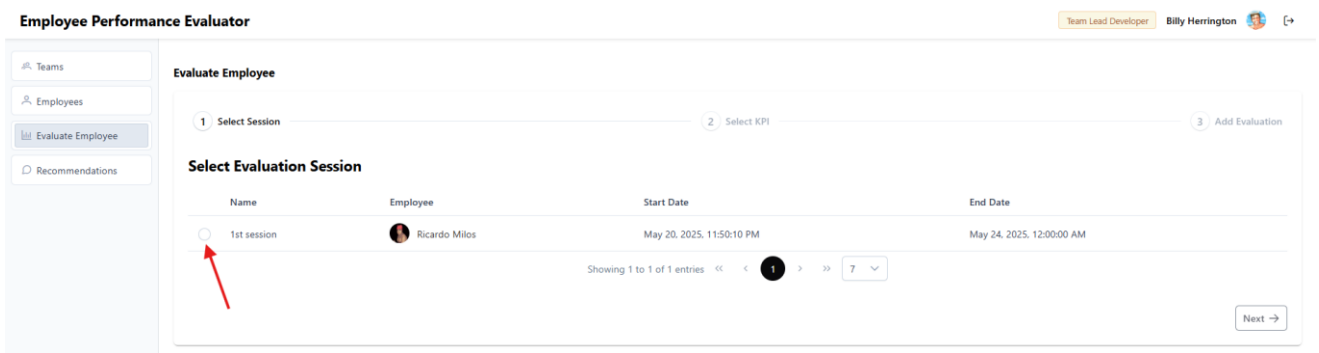


Рисунок 5.27 – Вікно оцінювання співробітника з вибором сесії

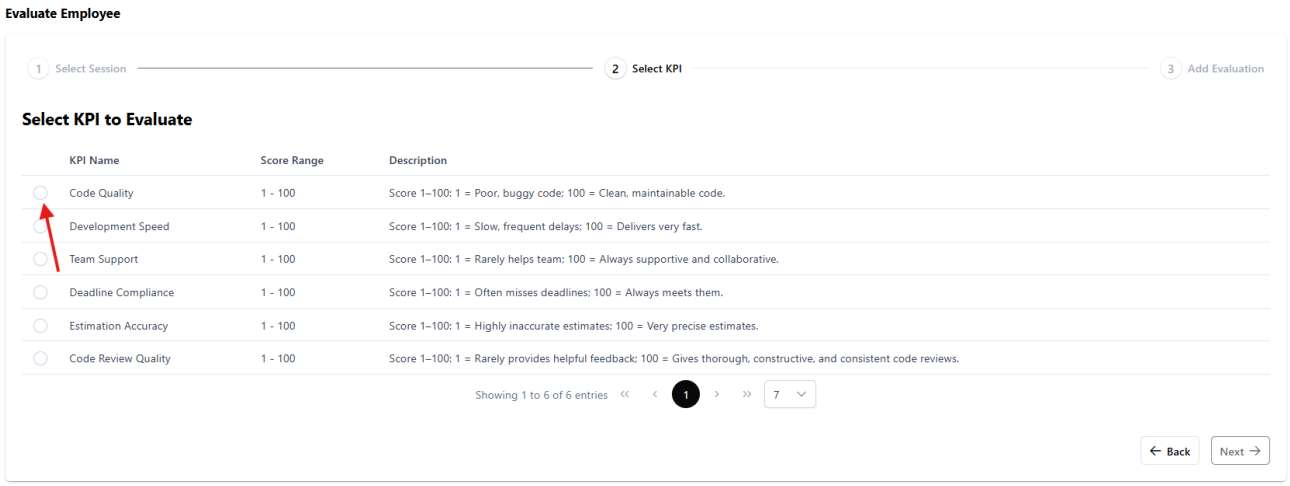


Рисунок 5.28 – Вікно оцінювання співробітника з вибором метрики для оцінки

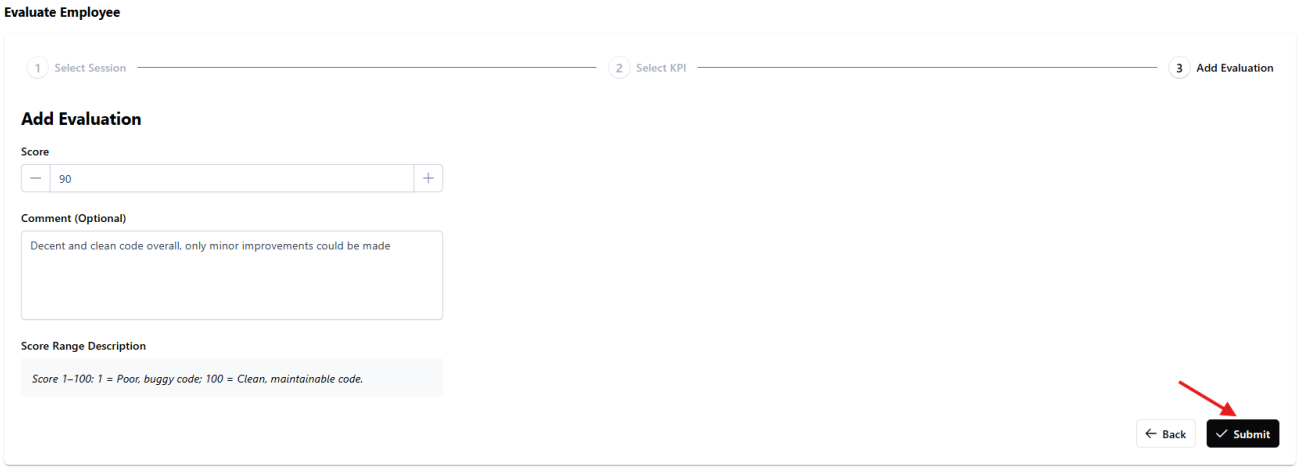


Рисунок 5.29 – Вікно оцінювання співробітника з додаванням оцінки

Для перегляду власних рекомендацій, співробітник має перейти на вкладку з меню «Recommendations», див. рисунки 5.30 та 5.31.

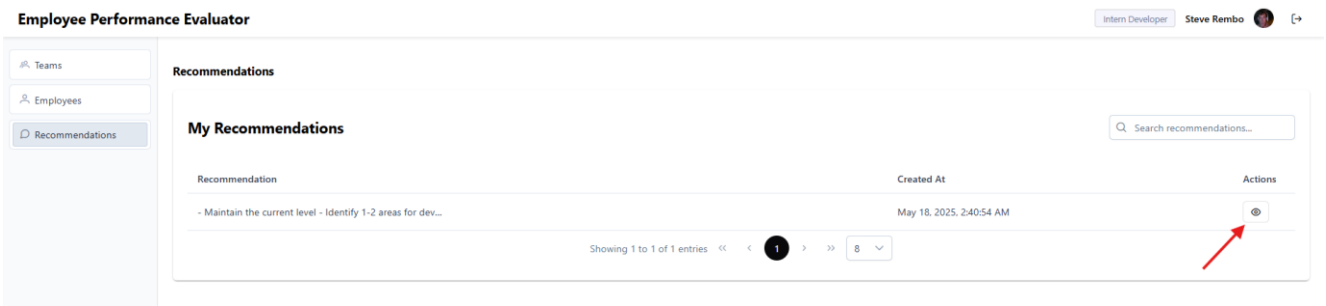


Рисунок 5.30 – Вікно власних рекомендацій

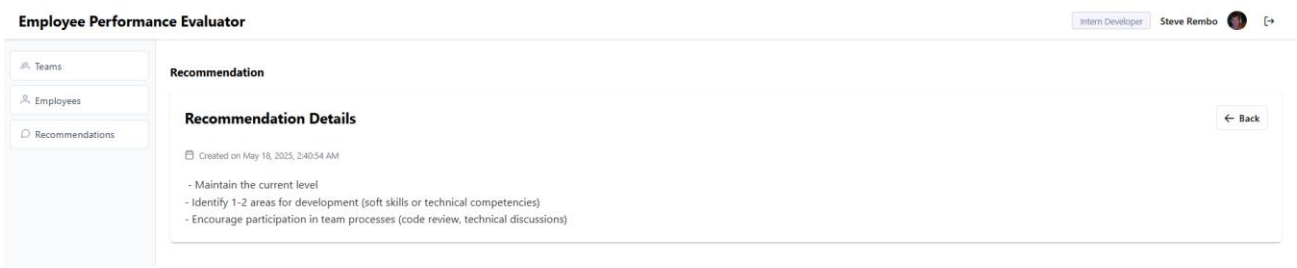


Рисунок 5.31 – Вікно вибраної рекомендації

Варто зауважити, що користувачі з відповідним доступом можуть переглядати та редагувати усі рекомендації, див. рисунок 5.32.

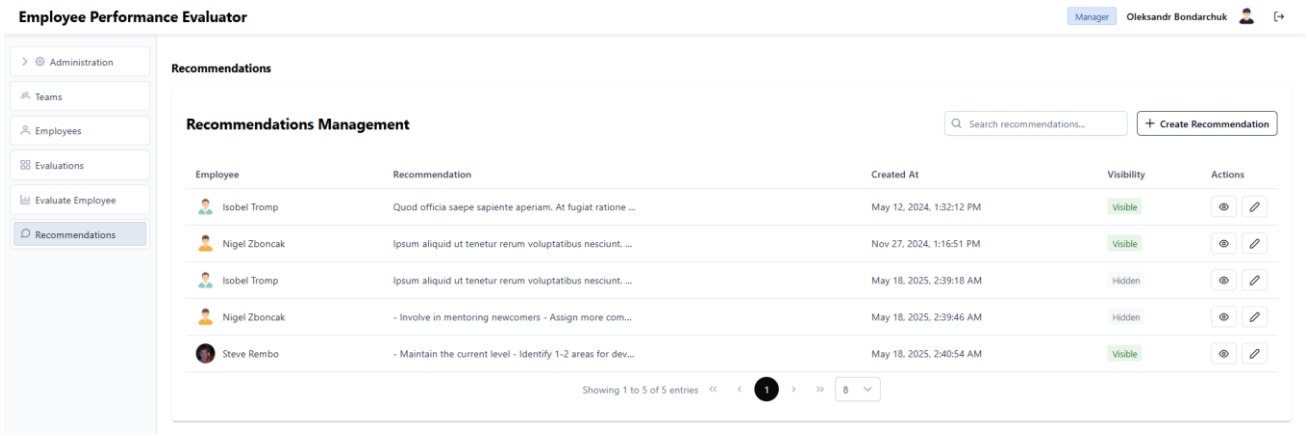


Рисунок 5.32 – Вікно рекомендацій з розширеним функціоналом

Висновок до розділу 5

У цьому розділі було здійснено детальний аналіз і реалізацію інформаційної системи для оцінювання якості роботи працівників. Повний код проєкту доступний за посиланням на рисунку А.1 (див. додаток А). Система побудована у вигляді централізованого вебзастосунку з чітким розподілом функціональності відповідно до ролей користувачів. Було розроблено клієнт-серверну архітектуру, яка включає окремі рівні: клієнтський інтерфейс, серверну логіку та рівень бази даних. Така структура забезпечує ефективне управління користувачами, оцінками та звітами, а також дозволяє масштабувати рішення за потреби.

6 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

6.1 Постановка задачі

6.1.1 Змістовна постановка задачі

У даному розділі розглядається математичне забезпечення вебзастосунку для оцінки якості роботи співробітників у IT та інших командо-орієнтованих компаніях. Основною метою цього розділу є розроблення та обґрунтування математичних моделей, алгоритмів і методів, що забезпечують аналіз попередніх оцінок працівників, визначення трендів їхнього професійного розвитку та прогнозування результатів діяльності в наступному періоді. Як зазначалося раніше, основним завданням системи є автоматизація процесу оцінювання якості роботи співробітників на основі KPI, а також прогнозування їхнього рівня професійної результативності.

Для вирішення цього завдання використовуються історичні дані оцінок за KPI, визначеними керуючим персоналом, з урахуванням вагового внеску кожної метрики, які охоплюють кілька періодів, протягом яких проводилися оцінювання. Ці дані дозволяють проаналізувати динаміку загальної результативності роботи працівників, та здійснити прогноз оцінок для наступного періоду з акцентом на останніх, більш релевантних часових відрізках.

Таким чином, ключовими задачами математичного забезпечення системи є:

– визначення трендів розвитку працівників на основі KPI;

– розроблення методу прогнозування професійної результативності працівників у наступному періоді з урахуванням ваг останніх періодів та історичних даних.

6.1.2 Математична постановка задачі

Нехай маємо:

– n – кількість періодів, за які були проведення оцінювання;

– m – кількість оціночних метрик (KPI);

– $S_{i,j}$ – оцінка результативності роботи працівника у i-му періоді за j-ю метрикою, де $i = 1,2,\ldots,n$ – номери оціночних періодів, $j = 1,2,\ldots,m$ – номери метрик ролі, $1 \leq S_{i,j} \leq 100$, де $S_{i,j} \in \mathbb{N}$;

– W_j – вага j-ї метрики, при умові, що $\sum_{j=1}^{m} W_j = 1$;

– P_i – загальна результативність роботи працівника у i-му періоді, яка обчислюється як зважена сума оцінок за формулою:

$$P_i = \sum_{j=1}^{m} W_j \cdot S_{i,j}; \quad (6.1)$$

– T_j – тренд розвитку для j-ї метрики (наприклад, числове значення, що відображає зміну оцінки з часом);

– P_{n+1} – прогнозована загальна результативність роботи у наступному періоді $n + 1$.

Метою задачі є:

1) на основі історичних даних оцінок працівника $S_{i,j}$ для $i = 1,2,\ldots,n$, $j = 1,2,\ldots,m$ визначити тренди T_j для кожної метрики;

2) використовуючи тренди для кожної метрики T_j та поточні оцінки $S_{n,j}$, спрогнозувати значення $S_{n+1,j}$ для кожної метрики;

3) обчислити прогнозовану загальну результативність роботи P_{n+1} у наступному періоді $(n + 1)$ за формулою:

$$P_{n+1} = \sum_{j=1}^{m} W_j \cdot S_{n+1,j}, \quad (6.2)$$

де $S_{n+1,j}$ – прогнозована оцінка працівника у наступному періоді $(n + 1)$ за j-ю метрикою.

Наявними обмеженнями є: тренди T_j повинні враховувати як короткострокові, так і довгострокові зміни, прогноз P_{n+1} має бути реалістичним і враховувати можливі коливання результативності роботи.

Таким чином, задача зводиться до аналізу рядів оцінок для кожної метрики, визначення їхньої динаміки та побудови прогнозу з урахуванням кожної метрики.

6.2 Обґрунтування методу розв’язання

Оскільки попередніх оцінювань співробітника може бути більше, ніж 2–3 (наприклад, 5–10), і при цьому останні оцінювання мають більшу вагу порівняно з давнішими, необхідно переглянути вибір методу з урахуванням цих уточнень.

6.2.1 Лінійна регресія

З переваг цього методу можна виділити простоту використання, добре працює з великою кількістю даних (5–10 періодів), виявляє загальний тренд. Із недоліків – не враховує більшу вагу останніх періодів, чутливий до викидів, припускає лінійність змін [12]. У лінійній регресії будується модель залежності значень показника для кожного критерію від номера періоду у вигляді:

$$S_{i,j} = a_i i + b_j, \tag{6.3}$$

де $S_{i,j}$ – значення j-го критерію в i-му періоді;

a_i, b_j – параметри лінійної регресії для j-го критерію;

i – номер періоду, $i = 1,2,\dots,n$;

j – номер критерію, $j = 1,2,\dots,m$.

Прогноз на наступний період $(n + 1)$ обчислюється як

$$S_{n+1,j} = a_i(n + 1) + b_j. \tag{6.4}$$

Отже, метод підходить для стабільних даних, але не відповідає вимозі акценту на останніх періодах.

6.2.2 Експоненційне згладжування

Експоненційне згладжування – це метод прогнозування часових рядів, який широко застосовується для аналізу та передбачення даних, що змінюються з часом. Експоненційне згладжування базується на рекурсивному обчисленні згладженого значення, яке є комбінацією поточного спостереження та попереднього згладженого значення [13]. Згладжене значення визначається як

$$S'_{i,j} = \alpha \cdot S_{i,j} + (1 - \alpha) \cdot S'_{i-1,j}, \tag{6.5}$$

де $S'_{i,j}$ – реальне значення j-го критерію в і-му періоді;

$S_{i,j}$ – згладжене значення;

$\alpha \in (0,1)$ – коефіцієнт згладжування, який визначає важливість нових даних. Формула для прогнозу для періоду $n + 1$ визначається як

$$S_{n+1,j} = S'_{i,j}. \tag{6.6}$$

Фактично, прогноз для наступного періоду $(n + 1)$ просто дорівнює останньому згладженому значенню. Експоненційне згладжування можна розглядати як зважене середнє всіх попередніх спостережень, де ваги зменшуються експоненційно з часом.

Отже, метод добре підходить для акценту на останніх періодах, але менш точний для прогнозування трендів при великій кількості даних.

6.2.3 Метод ковзного середнього з трендом

Метод базується на двох послідовних етапах: спочатку виконується згладжування часового ряду для усунення короткострокових коливань, а потім оцінюється напрямок і швидкість змін даних, тобто тренд [14]. Мета згладжування – усереднити останні k періодів, щоб виділити основну тенденцію без викидів.

Зазвичай обирається значення $k = 3$, тобто беруться до уваги останні три спостереження для кожного критерію. На основі цих згладжених значень обчислюється тренд T_j, який являє собою середню зміну між послідовними ковзними середніми для j-го критерію. Це дозволяє виявити стабільне зростання або спад показника. Після цього здійснюється прогнозування на наступний період $n + 1$. Формула прогнозу має вигляд:

$$S_{n+1,j} = S_{n,j} + T_j, \tag{6.7}$$

де $S_{n,j}$ – згладжене значення j-го критерію у поточному (останньому) періоді n;

T_j – середня зміна згладжених значень за останні періоди, що характеризує темп зміни показника.

Із недоліків, метод не надає явної переваги останнім періодам, якщо не використовувати зважене середнє, спрощує динаміку. Не враховує нелінійні тренди чи різкі зміни без додаткових модифікацій.

Отже, метод менш ефективний для задачі з акцентом на актуальність останніх даних.

6.2.4 Модель Хольта

Метод Хольта, також відомий як подвійне експоненційне згладжування, є розширенням простого експоненційного згладжування, яке враховує не лише рівень даних, а й їхній тренд [15]. Він корисний у ситуаціях, коли дані показують поступові зміни в одному напрямку, але не мають вираженої сезонності.

Формула для прогнозу для наступного періоду:

$$S_{n+1,j} = L_{n,j} + T_{n,j}, \tag{6.8}$$

де L_n – згладжене значення часового ряду, яке відображає його базовий рівень у момент періоду n;

T_n – згладжена оцінка зміни цього рівня, яка показує напрямок і швидкість тенденції.

Метод Хольта враховує тренди, акцентує увагу на останніх періодах завдяки α (коефіцієнт згладжування рівня) та β (коефіцієнт згладжування тренду) і є простим для реалізації. Він дозволяє прогнозувати майбутні тренди на основі історичних оцінок, однак він вимагає точного налаштування α і β, для уникнення можливих похибок.

6.2.5 Порівняння методів

Результати порівнянь розглянутих методів за точністю, складністю реалізації, адаптивністю до змін, врахуванні трендів, акценті на останніх періодах та кількості періодів наведено в таблиці 6.1.

Таблиця 6.1 – Порівняння методів.

Метод	Точність	Складність реалізації	Адапти-вність до змін	Враху-вання трендів	Акцент на останні періоди
Лінійна регресія	Низька	Низька	Низька	Так	Ні
Експоненційне згладжування	Середня	Висока	Середня	Частково	Так
Ковзне середнє	Середня	Висока	Низька	Так	Ні
Модель Хольта	Висока	Середня	Висока	Так	Так

Модель Хольта залишається найкращим варіантом у ситуаціях, коли кількість періодів велика (від 5 до 10 і більше), а останні значення мають більшу вагу. На відміну від простого експоненційного згладжування, яке враховує лише

рівень ряду, модель Хольта дозволяє врахувати як загальний напрям зміни даних у часі, так і нещодавні коливання. Завдяки цьому вона краще пристосовується до зміни тенденцій і точніше прогнозує дані в умовах, коли важливо враховувати як довгострокову динаміку, так і актуальні зміни.

На відміну від лінійної регресії, яка передбачає фіксований лінійний зв'язок, модель Хольта оновлює оцінки на кожному кроці та адаптується до змін. Зважене згладжування з трендом хоча й може бути ефективним, але потребує складнішого налаштування параметрів. Просте експоненційне згладжування не враховує зміну тенденцій, що знижує його точність при довших часових рядах.

6.3 Опис методу розв'язання

6.3.1 Загальні положення

Модель Хольта базується на двох основних компонентах: рівні ($L_{i,j}$) – згладжене значення оцінки для j-ї метрики у i-му періоді, що визначається за формулою:

$$L_{i,j} = \alpha \cdot S_{i,j} + (1 - \alpha)(L_{i-1,j} + T_{i-1,j}), \quad (6.9)$$

де α $(0 < \alpha < 1)$ – коефіцієнт згладжування рівня (більше значення – більша вага останніх даних);

$S_{i,j}$ – оцінка j-ї метрики у i-му періоді,

та тренді ($T_{i,j}$) – згладжена зміна оцінки, що відображає напрямок розвитку, що визначається за формулою:

$$T_{i,j} = \beta(L_{i,j} - L_{i-1,j}) + (1 - \beta) \cdot T_{i-1,j}, \quad (6.10)$$

де β $(0 < \beta < 1)$ – коефіцієнт згладжування тренду.

Загальна результативність роботи працівника обчислюється як зважена сума оцінок за всіма метриками з урахуванням їхніх ваг, див. формулу 6.2.

Схема обчислення прогнозу професійної результативності включає 4 етапи.

Етап 1. Провести ініціалізацію для першого періоду ($i = 1$). Рівень $L_{1,j}$ для кожної метрики j встановити рівним відповідній оцінці $S_{1,j}$, отриманої з історичних даних. Початковий тренд $T_{1,j}$ може бути прийнятий як нуль ($T_{1,j} = 0$), якщо немає додаткової інформації про динаміку, або обчислений як різниця між оцінками другого та першого періодів ($S_{2,j} - S_{1,j}$), якщо дані другого періоду доступні. Цей крок забезпечує базові значення для подальших рекурсивних обчислень.

Етап 2. Послідовно обчислити рівень та тренд для кожного наступного періоду ($i = 2,3,\ldots,n$). Для кожної метрики j у періоді i рівень $L_{i,j}$ слід розрахувати за формулою 6.9. Потім обчислити тренд $T_{i,j}$ за формулою 6.10. Ці обчислення повторюються для всіх періодів і метрик, створюючи згладжені значення, які враховують як поточні оцінки, так і їхню динаміку.

Етап 3. Після обробки всіх історичних даних виконати прогнозування для наступного періоду ($n + 1$). Для кожної метрики j прогнозовану оцінку $S_{n+1,j}$ визначити як суму останнього рівня та тренду:

$$S_{n+1,j} = L_{n,j} + T_{n,j}. \tag{6.10}$$

Етап 4. Обчислити загальну результативність роботи у прогнозованому періоді P_{n+1}, використовуючи формулу 6.2. Цей крок завершує прогноз, надаючи менеджерам числову оцінку очікуваної професійної результативності працівника.

Щоб модель працювала точно, потрібно правильно підібрати значення коефіцієнтів α і β. Це можна зробити, наприклад, мінімізуючи середньоквадратичну похибку – різницю між прогнозованими та реальними значеннями. Такий підбір дозволяє краще враховувати особливості конкретних даних. Зазвичай параметри підбирають автоматично за допомогою спеціальних методів, наприклад, перебору значень або спрощених алгоритмів оптимізації.

6.3.2 Приклад застосування

Наведемо приклад обчислення прогнозу професійної результативності розробника коду в IT-компанії за п’ять періодів (півріч), тобто $n = 5$. Для оцінки використаємо десять ключових показників ефективності, див таблицю 6.2. Кожному критерію призначено вагу (у відсотках), яка відображає його відносну важливість у загальній оцінці, причому сума всіх ваг дорівнює 100%. Ці ваги згенеровано для прикладу, щоб продемонструвати роботу алгоритму прогнозування, але вони можуть бути адаптовані до реальних потреб компанії.

Дані оцінок і ваги використовуються для аналізу історичної результативності роботи розробника коду та прогнозування його результатів у наступному, шостому півріччі.

Таблиця 6.2 – Історичні дані оцінок розробника коду.

Метрика	Вага (W_j)	Період 1	Період 2	Період 3	Період 4	Період 5
Якість коду	20%	80	85	82	88	90
Тайм-менеджмент	15%	70	72	75	78	80
Командна робота	10%	85	88	90	87	85
Ініціативність	5%	60	65	70	75	80
Дотримання стандартів коду	15%	75	78	80	82	85
Самостійність	10%	70	75	78	80	82
Комунікаційні навички	5%	90	92	88	85	87
Відповідальність	10%	80	82	85	88	90
Готовність до навчання	5%	65	70	75	80	85
Якість виконаних завдань	10%	78	80	82	85	88

Для демонстрації роботи алгоритму методу Хольта розглядається прогнозування результативності роботи розробника на основі історичних оцінок за п’ять періодів (півріччя) для десяти метрик, таких як якість коду, тайм-менеджмент, командна робота тощо. Використовуються параметри $\alpha = 0.7$, що забезпечує значний акцент на останніх оцінках, і $\beta = 0.3$, що відповідає помірному згладжуванню тренду. Обчислення проводяться послідовно для кожної метрики, після чого визначається загальна результативність роботи працівника. Нижче описано розширений процес обчислень на прикладі метрики "Якість коду" та узагальнено для інших метрик, завершуючи прогнозом загальної результативності роботи.

Спочатку виконується ініціалізація для першого періоду ($i = 1$). Для метрики "Якість коду" ($j = 1$) рівень встановлюється як оцінка першого періоду: $L_{1,1} = S_{1,1} = 80$. Початковий тренд обчислюється як різниця між оцінками другого та першого періодів: $T_{1,1} = S_{2,1} - S_{1,1} = 85 - 80 = 5$. Цей тренд відображає початкову тенденцію зростання оцінки і використовується як базис для подальших ітерацій.

Далі обчислення проводяться для періодів 2–5. Для другого періоду ($i = 2$) рівень $L_{2,1}$ розраховується за формулою $L_{2,1} = 0.7 \cdot 85 + (1 - 0.7)(80 + 5) = 85$, що поєднує нову оцінку з прогнозом на основі попереднього рівня та тренду. Тренд $T_{2,1}$ оновлюється як $T_{2,1} = 0.3(85 - 80) + (1 - 0.3) \cdot 5 = 5$, зберігаючи тенденцію зростання. Для третього періоду ($i = 3$) рівень становить $L_{3,1} = 0.7 \cdot 82 + 0.3(85 + 5) = 84.4$, а тренд знижується до $T_{3,1} = 0.3(84.4 - 85) + 0.7 \cdot 5 = 3.32$, відображаючи уповільнення зростання. У четвертому періоді ($i = 4$) обчислення дають $L_{4,1} = 0.7 \cdot 88 + 0.3(84.4 + 3.32) = 87.826$ і $T_{4,1} = 0.3(87.826 - 84.4) + 0.7 \cdot 3.32 = 3.3518$. Нарешті, для п’ятого періоду ($i = 5$) рівень становить $L_{5,1} = 0.7 \cdot 90 + 0.3(87.826 + 3.3518) = 90.3534$, а тренд — $T_{5,1} = 0.3(90.3534 - 87.826) + 0.7 \cdot 3.3518 = 3.1045$. Ці обчислення відображають поступове згладжування оцінок із урахуванням їхньої динаміки.

Для прогнозування оцінки в шостому періоді ($i = 6$) для "Якості коду"

додаються останній рівень і тренд: $S_{6,1} = L_{5,1} + T_{5,1} = 90.3534 + 3.1045 = 93.4579 \approx 93.46$. Аналогічні обчислення проводяться для решти метрик (тайм-менеджмент, командна робота тощо), враховуючи ті ж значення α і β. Для спрощення результати для інших метрик наведено як прогнозовані оцінки, наприклад, 82.50 для тайм-менеджменту, 84.20 для командної роботи, 84.80 для ініціативності тощо, отримані шляхом повторення описаного процесу. Результати прогнозів для кожної метрики наведено в таблиці 6.3.

Таблиця 6.3 – Результати прогнозів результативності роботи.

Метрика	$S_{6,j}$ (прогноз)
Якість коду	93.46
Тайм-менеджмент	82.50
Командна робота	84.20
Ініціативність	84.80
Дотримання стандартів коду	87.30
Самостійність	83.90
Комунікаційні навички	88.10
Відповідальність	92.00
Готовність до навчання	89.50
Якість виконаних завдань	90.20

На завершальному етапі обчислюється загальна результативність роботи. Для п’ятого періоду (P_5) вона розраховується як зважена сума оцінок: $P_5 = 0.2 \cdot 90 + 0.15 \cdot 80 + 0.1 \cdot 85 + 0.05 \cdot 80 + 0.15 \cdot 85 + 0.1 \cdot 82 + 0.05 \cdot 87 + 0.1 \cdot 90 + 0.05 \cdot 85 + 0.1 \cdot 88 = 85.95$. Для шостого періоду ($P_6$) використовуються прогнозовані оцінки для кожної метрики з таблиці 6.3: $P_6 = 0.2 \cdot 93.46 + 0.15 \cdot 82.5 + 0.1 \cdot 84.2 + 0.05 \cdot 84.8 + 0.15 \cdot 87.3 + 0.1 \cdot 83.9 + 0.05 \cdot 88.1 + 0.1 \cdot 92 + 0.05 \cdot 89.5 + 0.1 \cdot 90.2 = 88.24$.

Отримане значення 88.24 для P_6 порівняно з 85.95 для P_5 вказує на зростання результативності роботи, що підтверджує позитивний тренд розвитку працівника.

Висновок до розділу 6

У цьому розділі було розглянуто методи прогнозування результативності роботи працівників, зокрема експоненційне згладжування, лінійну регресію, ковзне середнє з трендом та метод Хольта. Порівняння проводилося за такими характеристиками: точність, адаптивність до змін, врахування трендів і простота реалізації, за результатами якого основним методом обрано метод Хольта. Цей алгоритм є ефективним для аналізу історичних оцінок, акцентує увагу на останніх даних і легко впроваджується у вебзастосунок.

Наведено приклад прогнозування результативності працівника на основі загальновідомих метрик для ролі розробника. Метод дає змогу проводити базову аналітику змін оцінок і прогнозувати розвиток працівника. Це створює основу для подальшої інтеграції з модулями стратегічного планування та управління персоналом. У перспективі прогнозну модель можна вдосконалити за допомогою алгоритмів машинного навчання. Такий підхід підвищить точність оцінювання та дозволить виявляти приховані закономірності в динаміці ефективності працівників.

7 ТЕСТУВАННЯ СИСТЕМИ

У цьому розділі наводиться опис тестів і порядок їх виконання для перевірки відповідності програмного забезпечення системи функціональним вимогам, представленим у розділі 3.

7.1 Мета випробувань

Мета випробувань полягає у забезпеченні якісної перевірки функціональності вебзастосунку для оцінки якості роботи працівників, щоб переконатися, що всі ключові компоненти системи працюють відповідно до заданих вимог. Основна увага приділяється перевірці коректності обробки даних про оцінки співробітників, обчислень зважених балів, генерації рекомендацій та звітів, а також правильного розподілу прав доступу між користувачами з різними ролями (менеджери, лідери команд, HR). Випробування також спрямовані на виявлення можливих помилок у логіці API, інтерфейсу та взаємодії з базою даних, щоб гарантувати стабільність і надійність системи в реальних умовах використання. Для досягнення цієї мети тестування проводилося з використанням автоматично згенерованих даних, що дозволило моделювати різноманітні сценарії роботи системи.

7.2 Загальні положення

У контексті програмного забезпечення тестування є критичним етапом життєвого циклу розробки, що включає перевірку функціональності, продуктивності, безпеки та зручності використання. У даному випадку застосовувалися методи ручного end-to-end (E2E) тестування та тестування API за допомогою Postman (див. рисунок 7.1), що дозволило перевірити логіку роботи системи на всіх рівнях – від користувацького інтерфейсу до серверної обробки запитів.

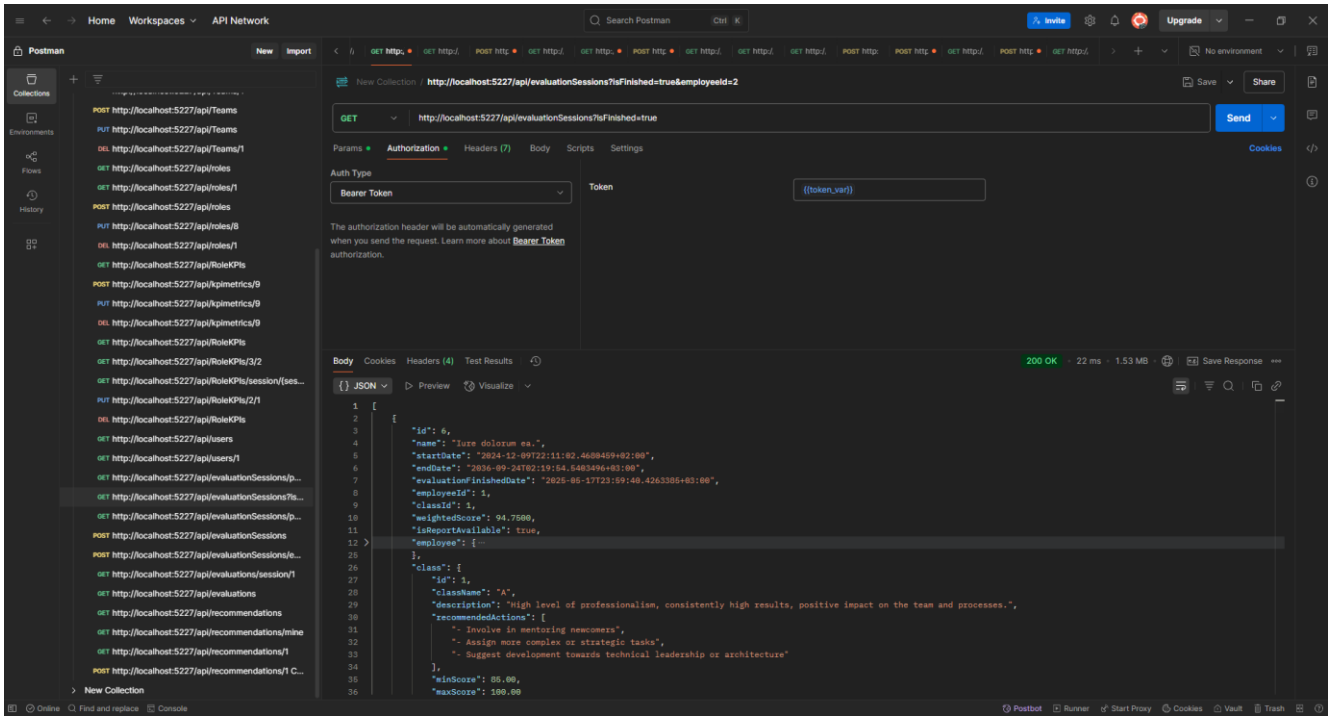


Рисунок 7.1 – Колекція запитів у Postman для тестування

Дані для тестування генерувалися за допомогою бібліотеки Bogus, яка створює реалістичні набори даних (наприклад, імена співробітників, оцінки за метриками, дати), що імітують реальну базу даних. Процес включав підготовку тестових сценаріїв, виконання запитів до API (наприклад, створення сесії оцінювання, додавання оцінок), а також ручну перевірку результатів у вебінтерфейсі.

7.3 Результати випробувань

Тестування системи проводилося з використанням Postman для перевірки API та ручного end-to-end (E2E) тестування через вебінтерфейс, охоплюючи всі ключові контролери, які реалізують функціональність системи: AuthController (керування авторизацією), EmployeeClassesController (управління класами результативності роботи), EmployeesController (додавання та редагування даних про співробітників), EvaluationsController (обробка оцінок за метриками), EvaluationSessionsController (створення, редагування та завершення сесій оцінювання), KPIMetricsController (управління ключовими показниками

ефективності), RecommendationsController (генерація та перегляд рекомендацій), RoleKPIsController (налаштування ваг і діапазонів балів для ролей), RolesController (керування ролями користувачів), TeamsController (управління командами) та UsersController (керування профілями користувачів). Тестування проводилося з використанням даних, згенерованих бібліотекою Bogus, що включало профілі 20 унікальних співробітників із різними іменами та прізвищами, 10 команд із випадковими назвами (наприклад, "Team Alpha", "Team Beta"), 45 ключових показників ефективності, а також 10 сесій оцінювання з різними датами початку та завершення.

Перевірка API включала широкий спектр запитів до кожного контролера. У більшості випадків запити завершувалися з кодом 200 OK, що свідчило про коректну обробку даних. Ручне E2E-тестування проводилося через вебінтерфейс із моделюванням типових сценаріїв: створення нового співробітника, ініціювання сесії оцінювання, введення оцінок, перегляд рекомендацій та генерація звітів у PDF. Перевірка ролей підтвердила, що менеджер може додавати команди, лідер – оцінювати підлеглих, а HR – переглядати статистику. Незначні візуальні затримки (до 2 секунд) спостерігалися при завантаженні великих наборів даних, але вони не вплинули на функціональність. Загалом система продемонструвала високу відповідність вимогам, із виправленнями, які значно підвищили її практичну цінність.

Нижче наведено розширені результати тестування у таблицях 7.1 – 7.7.

Таблиця 7.1 – Перевірка авторизації користувача

Назва параметру	Значення
Ціль	Перевірка можливості входу в систему для менеджера
Передумова	- дані для авторизації менеджера вже існують і занесені до бази даних; - користувач перебуває на сторінці авторизації
Вхідні дані	Дані авторизації: email = "manager@example.com", password = "securepassword123".

Продовження таблиці 7.1

Послідовність виконання	- у поле "Email" ввести "manager@example.com"; - у поле "Password" ввести "securepassword123"; - натиснути кнопку "Login"
Очікуваний результат	Користувач авторизований і перенаправлений на сторінку управління командами
Отриманий результат	Користувач успішно авторизувався з кодом 200 OK і отримав доступ до системи

Таблиця 7.2 – Перевірка створення сесії оцінювання

Назва параметру	Значення
Ціль	Перевірка можливості створення нової сесії оцінювання для співробітника
Передумова	- користувач із роллю менеджера авторизований; - дані про співробітника (employeeId = 5) існують у базі
Вхідні дані	Дані сесії: employeeId = 5, endDate = "2025-06-30", name = "1st session"
Послідовність виконання	- відправити POST-запит до /evaluationSessions з JSON: {"employeeId": 5, "endDate": "2025-06-30", name: "1st session"}; - перевірити відповідь у Postman
Очікуваний результат	Сесія створена з унікальним sessionId і кінцевою датою "2025-06-30"
Отриманий результат	Сесія створена з sessionId = 21, кінцевою датою "2025-06-30", відповідь 200 OK

Таблиця 7.3 – Перевірка додавання оцінки

Назва параметру	Значення
Ціль	Перевірка додавання оцінки за метрикою для сесії

Продовження таблиці 7.3

Передумова	- сесія оцінювання (sessionId = 21) створена; - користувач авторизований; - користувач має роль лідера команди
Вхідні дані	Дані оцінки: sessionId = 21, kpiId = 1, roleId = 5, score = 85, comment = "Good performance"
Послідовність виконання	- відправити POST-запит до /evaluations з JSON: {"sessionId": 21, "kpiId": 1, "score": 85, "comment": "Good performance"}; - перевірити збереження в базі
Очікуваний результат	Оцінка додана з коректним score і comment, відображено в системі
Отриманий результат	Оцінка додана з evaluationId = 15, score = 85, comment = "Good performance" збережено, відповідь 200 OK

Таблиця 7.4 – Перевірка доданих рекомендацій

Назва параметру	Значення
Ціль	Перевірка доданих персональних рекомендацій на основі оцінок
Передумова	- існує щонайменше одна сесія оцінювання для співробітника (employeeId = 5); - користувач авторизований; - користувач має роль оцінювача
Вхідні дані	Запит GET /recommendations/mine
Послідовність виконання	- відправити GET-запит до /recommendations/mine; - перевірити відповідь у Postman
Очікуваний результат	Повернуто текст рекомендації, наприклад, "Improve time management"
Отриманий результат	Повернуто рекомендацію: " Improve time management", код 200 OK, текст збережений

Таблиця 7.5 – Перевірка завершення сесії оцінювання

Назва параметру	Значення
Ціль	Перевірка можливості завершення сесії з оцінками
Передумова	- сесія (sessionId = 21) містить принаймні одну оцінку; - користувач із роллю менеджера авторизований
Вхідні дані	Запит POST /evaluationSessions/end/:id
Послідовність виконання	- відправити POST-запит до /evaluationSessions/end/:id; - перевірити оновлення статусу
Очікуваний результат	Сесія завершена з оновленим полем EvaluationFinishedDate
Отриманий результат	Сесія завершена, поле EvaluationFinishedDate має актуальну дату, відповідь 200 OK після перевірки наявності оцінок

Таблиця 7.6 – Перевірка фільтрації сесій оцінювання

Назва параметру	Значення
Ціль	Перевірка коректності фільтрації доступних сесій за датою завершення
Передумова	- існують сесії з різними датами EvaluationFinishedDate (наприклад, "2025-05-20" чи "null"); - користувач із роллю оцінювача авторизований
Вхідні дані	Запит GET /evaluationSessions/pending-to-be-evaluated та поточною датою 02:00 AM EEST 24 травня 2025 року
Послідовність виконання	- відправити запит до /evaluationSessions/pending-to-be-evaluated - перевірити відповідь у Postman
Очікуваний результат	Повернуто лише незавершені сесії
Отриманий результат	Повернута лише сесія з EvaluationFinishedDate = "null", сесія з "2025-05-20" відфільтрована, код 200 OK

Таблиця 7.7 – Перевірка генерації звіту

Назва параметру	Значення
Ціль	Перевірка можливості генерації звіту для завершеної сесії оцінювання
Передумова	- сесія (sessionId = 21) завершена і містить оцінки; - користувач із роллю менеджера авторизований
Вхідні дані	Запит GET /evaluationSessions/generate-report/21 з токеном авторизації
Послідовність виконання	- відправити GET-запит до /evaluationSessions/generate-report/21; - завантажити файл звіту відправивши GET-запит до /evaluationSessions/report/21
Очікуваний результат	Повернуто файл PDF зі зведеними оцінками, зваженим балом і рекомендаціями
Отриманий результат	Повернуто PDF-файл із оцінками (наприклад, 85 за "Якість коду"), зваженим балом 88.24 і рекомендацією "Покращити тайм-менеджмент", код 200 OK

Під час тестування було виявлено кілька проблем, які потребували виправлення.

По-перше, при отриманні доступних сесій для оцінювання через GET /evaluationSessions контролер повертав також завершені сесії з датами закінчення в минулому, що вводило в оману користувачів. Ця проблема була усунута шляхом додавання фільтрації на серверному рівні, де відображалися лише сесії з датою завершення, що перевищує поточну, що покращило зручність використання.

По-друге, виявлено можливість завершення сесії через POST /evaluationSessions/end/:id без наявності хоча б однієї оцінки, що суперечило логіці системи. Виправлення полягало в додаванні перевірки на наявність пов’язаних записів у таблиці Evaluations перед оновленням статусу сесії.

По-третє, у генерації прогнозів результативності роботи виникла потреба в

уточненні умов: спочатку прогнозування виконувалося навіть за наявності однієї або двох сесій, що не давало репрезентативних результатів. Було додано перевірку, яка вимагає щонайменше трьох сесій оцінювання для обчислення трендів методом Хольта, що підвищило надійність прогнозів.

По-четверте, було виявлено недостачу перевірки вводу даних у формах на клієнтській стороні, коли введення некоректних даних (наприклад, від'ємних балів чи тексту замість чисел) призводило до на клієнтській стороні; ця проблема була вирішена шляхом додавання клієнтської валідації з використанням Angular-форм, що перевіряє правильність введених даних перед відправкою запиту.

Висновок до розділу 7

У розділі було проведено комплексне тестування вебзастосунку для оцінки продуктивності працівників, яке включало перевірку API за допомогою Postman та ручне E2E-тестування з використанням даних, згенерованих бібліотекою Bogus. Результати тестування, деталізовані у таблицях 7.1–7.7, підтвердили відповідність системи функціональним вимогам, зокрема коректність обчислень зважених балів, генерації звітів у форматі PDF, управління доступом за ролями, фільтрації сесій оцінювання та прогнозування продуктивності методом Хольта. Під час тестування було виявлено та усунуто кілька недоліків: додана фільтрація сесій за датою завершення, введена перевірка наявності оцінок перед завершенням сесії, встановлена мінімальна кількість сесій для прогнозування, а також впроваджена клієнтська валідація форм для уникнення введення некоректних даних.

Хоча система продемонструвала стабільність, було помічено потребу в оптимізації часу генерації звітів (затримки до 2 секунд при великих обсягах даних) та вдосконаленні обробки помилок для підвищення зручності використання.

Загалом, проведене тестування підтвердило готовність застосунку до впровадження в реальних умовах командно-орієнтованих компаній, зокрема в IT-секторі, із рекомендацією подальшого вдосконалення продуктивності при масштабуванні та додавання механізмів для обробки великих обсягів даних.

ВИСНОВКИ

Розроблена інформаційна система оцінювання якості роботи співробітників успішно реалізує поставлені в дипломному проєкті завдання: вона автоматизує процес збору, обробки та збереження результатів оцінювання на основі заздалегідь визначених KPI, а також генерує структуровані PDF-звіти з детальною інформацією про сесію оцінювання, зважений підсумковий бал і рекомендації щодо подальшого розвитку. Завдяки впровадженню багаторівневої моделі ролей та прав доступу система гарантує безпеку даних і відповідність принципам розмежування обов'язків у корпоративному середовищі.

Функціональні можливості модуля формування сесій оцінювання охоплюють налаштування набору метрик та вагових коефіцієнтів для кожної ролі, створення та завершення сесій, а також введення й збереження оцінок з коментарями. Інтеграція клієнтської частини на базі Angular/TypeScript із серверним API на ASP.NET Core/C# та EF Core забезпечує високу продуктивність запитів і зручність інтерфейсу.

Нефункціональні вимоги до системи (надійність, продуктивність, безпека, розширюваність) були досягнуті завдяки застосуванню сучасних практик розробки та архітектурних підходів: шарова структура серверної частини дозволила чітко відокремити бізнес-логіку від реалізації доступу до даних та гарантувати легкість підтримки та розширення функціоналу в майбутньому. Для генерації PDF-звітів обрано бібліотеку DinkToPdf, яка надала можливість використовувати повноцінні HTML/CSS-шаблони, що суттєво полегшило створення візуально привабливих звітних документів і дало змогу швидко налаштовувати стиль звітів, якщо з'явиться така потреба.

Тестування системи охопило інтеграційні тести для перевірки взаємодії усіх компонентів системи та тестування API через Postman. Результати показали коректне виконання всіх операцій: створення, оновлення й видалення будь-яких сутностей у системі, обробка виключень і коректне накладання прав доступу. Виявлені в процесі тестування незначні недоліки (помилки фільтрації сесій

оцінювання, прогнозування з недостатньою кількістю оцінок, відсутність валідації на рівні інтерфейсу) були виправлені.

Для подальшого вдосконалення системи можна розглянути можливість інтеграції з корпоративними системами Single Sign-On для централізованого керування доступом, додавання візуалізації динаміки результативності роботи та оптимізації створення звітів з великою кількістю оцінок. Це дозволить забезпечити ще вищий рівень масштабованості, зменшити час генерації звітів та збільшить загальну ефективність процесів оцінювання.

Отже, виконана робота створює міцну основу для подальшого розвитку автоматизованих HR-рішень і вдало демонструє застосування сучасних технологій у контексті дипломного проєкту.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Angular documentation. [Електронний ресурс]. Режим доступу: https://angular.dev/overview

2. RxJS documentation. [Електронний ресурс]. Режим доступу: https://rxjs.dev/guide/overview

3. TypeScript documentation. [Електронний ресурс]. Режим доступу: https://www.typescriptlang.org/docs/

4. SCSS documentation. [Електронний ресурс]. Режим доступу: https://sass-lang.com/documentation/

5. PrimeNG documentation. [Електронний ресурс]. Режим доступу: https://primeng.org/

6. ASP.NET Core documentation. [Електронний ресурс]. Режим доступу: https://learn.microsoft.com/uk-ua/aspnet/core/?view=aspnetcore-8.0

7. C# documentation. [Електронний ресурс]. Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/

8. .NET Core documentation. [Електронний ресурс]. Режим доступу: https://learn.microsoft.com/en-us/dotnet/core/introduction/

9. MS SQL Server documentation. [Електронний ресурс]. Режим доступу: https://docs.microsoft.com/en-us/sql/sql-server/

10. Microsoft Documentation: Clean architecture with ASP.NET Core. [Електронний ресурс]. Режим доступу: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures

11. Clean Architecture by Robert C. Martin. [Електронний ресурс]. Режим доступу: https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html

12. Окара Д. В., та ін. Економетрія: навчальний посібник. Одеса: ОДАБА, 2018. 144 с. С. 18–29.

13. Терентьєв, О., & Дуда, В. (2025). Метод відновлення пропусків у даних на основі комбінованої моделі експоненційного згладжування. Екологічна безпека та природокористування, 53(1), 125–131.

14. Широкопетлєва М. С., Пономаренко О. А., Дудар З. В. Порівняння методів прогнозування часових рядів. Біоніка інтелекту, 2018, №2(91). С.41-47.

15. Васильєв О. С., Лилка О. С. Використання методу Хольта для аналізу часових рядів. Проблеми інформатизації та управління, 2010, №3(31). С.26–29. [Електронний ресурс]. Режим доступу: https://jrnl.nau.edu.ua/index.php/PIU/article/view/6424/7226