КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет автоматизації і інформаційних технологій

Кафедра інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

на тему:

Розробка веб-системи автоматизованої оцінки дефектів будівельних конструкцій

Рапін Олександр Олександрович

(прізвище, ім’я та по батькові здобувача повністю)

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет автоматизації і інформаційних технологій

Кафедра інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувачка кафедри ІТ
д.т.н., професор Гончаренко Т. А.
_____
„___” _____2025року

КВАЛІФІКАЦІЙНА РОБОТА
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

Розробка веб-системи автоматизованої оцінки дефектів будівельних конструкцій

(тема)

Я як здобувач вищої освіти КНУБА розумію і підтримую політику закладу з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Здобувач Рапін Олександр Олександрович
(прізвище, ім’я та по-батькові повністю)
122 «Комп’ютерні науки»
(спеціальність)
Інформаційні управляючі системи та технології
(освітня програма)
Групи КН-21-1
Керівник Бородавка Є. В.
(прізвище та ініціали)
Доктор технічних наук, професор
(вчене звання, науковий ступінь)
Рецензент Гуменний Д.О. к.т.н., доцент
(Прізвище та ініціали)

Ідентичність підтверджую

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет автоматизації і інформаційних технологій
Випускова кафедра інформаційних технологій
Освітній ступінь: бакалавр
Спеціальність: 122 Комп'ютерні науки
Освітня програма: Інформаційні управляючі системи та технології

ЗАТВЕРДЖУЮ
Завідувачка кафедри ІТ
д.т.н., професор Гончаренко Т. А.
_____
„___" _____2025року

З А В Д А Н Н Я
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

_____Рапін Олександр Олександрович
(прізвище, ім'я та по батькові здобувача)

1. Тема роботи Розробка веб-системи автоматизованої оцінки дефектів будівельних конструкцій
затверджена наказом ректора КНУБА № 235 від 14 лютого 2025 року.

2. Керівник роботи Бородавка Є. В. доктор технічних наук, професор
(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)

3. Строк подання Здобувачем роботи до захисту

4. Зміст пояснювальної записки за розділами:

Р.1 Аналіз існуючих методів оцінки стану будівельних конструкцій

Р.2 Проєктування архітектури та функціоналу веб-системи

Р.3 Програмна реалізація та тестування системи

Р.4 Ергономіка та економічне обгрунтування веб-системи

5. Графічний матеріал за розділом:

Р.1 Слайди 2, 3, 4

Р.2 Слайди 5, 6, 7, 8

Р.3 Слайди 9, 10, 11, 12, 13

Р.4 Слайд 14

6. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Розділ 1	04.02.2025
Розділ 2	10.04.2025
Розділ 3	25.04.2025
Розділ 4	23.05.2025
Остаточне оформлення роботи	26.05.2025
Направлення роботи для перевірки на плагіат	26.05.2025
Попередній захист роботи на випусковій кафедрі	27.05.2025
Направлення роботи на рецензування	28.05.2025

7. Консультанти розділів атестаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта	Перевірив	
		дата	підпис
Розділ 1	Бородавка Є. В.	04.02.2025	
Розділ 2	Гончаренко Т. А.	15.05.2025	
Розділ 3	Мацієвський О. О.	19.05.2025	
Розділ 4	Рябчун Ю.В.	21.05.2025	

8. Дата видачі завдання 14 лютого 2025 року

Завідувачка _____ Гончаренко Т. А.
(підпис) (прізвище та ініціали)

Керівник _____ Бородавка Є. В.
(підпис) (прізвище та ініціали)

Здобувач _____ Рапін О. О.
(підпис) (прізвище та ініціали)

РЕЗЮМЕ (SUMMARY) *до атестаційної випускної роботи Здобувача:*	Рапін Олександр Олександрович Rapin Olexandr		
ЗВО	Київський національний університет будівництва і архітектури		
Тема (українською та англійською)	Розробка веб-системи автоматизованої оцінки будівельних конструкцій Development of a web system for automated assessment of building structures		
Освітній ступінь	Бакалавр		
Факультет	Автоматизації і інформаційних технологій		
Випускаюча кафедра	Інформаційних технологій		
Спеціальність	122 «Комп'ютерні науки»		
Освітня програма	Інформаційні управляючі системи та технології		
Керівник	Бородавка Євгеній Володимирович		
Обсяг роботи:	пояснювальна записка, стор.	розділів	креслень формату А
	127	4	
Розділ 1.	Аналіз існуючих методів оцінки стану будівельних конструкцій		
Розділ 2.	Проєктування архітектури та функціоналу веб-системи		
Розділ 3.	Програмна реалізація та тестування системи		
Розділ 4.	Ергономіка та економічне обгрунтування веб-системи		
Ключові слова: *Keywords:*	оцінка стану будівель, автоматизовані системи, експертні методи, пріоритетність ремонту, веб-розробка, алгоритми прийняття рішень building condition assessment, automated systems, expert methods, repair prioritization, web development, decision-making algorithms.		

Здобувач: _____/ Олександр РАПІН /

Керівник: _____/ Євгеній БОРОДАВКА /

"___" _____2025р.

АНОТАЦІЯ

Рапін О. О. Розробка веб-системи для автоматизованої оцінки стану будівельних конструкцій.

Кваліфікаційна випускна робота бакалавра за спеціальністю: 122 «Комп'ютерні науки», освітньо-професійна програма: «Інформаційні управляючі системи і технології». – Київський національний університет будівництва та архітектури. – Київ, 2025.

Робота присвячена розробці веб-системи для автоматизованої оцінки стану будівельних конструкцій із використанням методів експертного аналізу. Описано сучасні методи оцінки, підходи до автоматизації процесу прийняття рішень, а також архітектуру та реалізацію системи. Представлено результати тестування, які підтверджують ефективність розробленого програмного забезпечення у визначенні пріоритетності ремонтних робіт.

Ключові слова: оцінка стану будівель, автоматизовані системи, експертні методи, пріоритетність ремонту, веб-розробка, алгоритми прийняття рішень.

SUMMARY

Rapin O. O. Development of a web system for automated assessment of the condition of building structures.

Bachelor's thesis for a bachelor's degree in specialty: 122 “Computer Science”, specialization: "Information Management Systems and Technologies" - Kyiv National University of Construction and Architecture - Kyiv, 2025.

The work is devoted to the development of a web system for the automated assessment of the condition of building structures using expert analysis methods. Modern assessment methods, approaches to automating decision-making processes, as well as the system’s architecture and implementation are described. The test results confirm the effectiveness of the developed software in determining the priority of repair work.

Keywords: building condition assessment, automated systems, expert methods, repair prioritization, web development, decision-making algorithms.

ЗМІСТ

ЗМІСТ **7**
ВСТУП **9**
РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ОЦІНКИ СТАНУ БУДІВЕЛЬНИХ КОНСТРУКЦІЙ **11**
1.1 Аналіз причин пошкоджень будівельних конструкцій **11**
1.2 Сучасні підходи до оцінки стану будівельних конструкцій **14**
1.3 Методи експертних оцінок у будівельній сфері **19**
1.4 Проблеми існуючих підходів та обґрунтування необхідності автоматизації .. **24**
1.5 Висновок до першого розділу **27**
РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ФУНКЦІОНАЛУ ВЕБ-СИСТЕМИ **29**
2.1 Обґрунтування архітектурних рішень веб-системи **29**
2.2 Побудова логічної структури веб-системи **32**
2.3 Опис основних функціональних компонентів системи **36**
2.4 Розробка взаємодії між компонентами системи **45**
2.5 Формування вимог до даних та схеми їх обробки **49**
2.6 Висновок до другого розділу **53**
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ **55**
3.1 Вибір інструментів і технологій розробки **55**
3.2 Вибір моделі життєвого циклу **58**
3.3 Загальна архітектура системи **61**
3.4 Логічна структура та модульна побудова **62**
3.5 Функціональні переходи та алгоритми обробки оцінок **64**
3.6 Реалізація клієнтської частини (frontend) **69**
3.7 Реалізація серверної частини (backend) **74**
3.8 База даних та сховище інформації **80**
3.9 Тестування системи **83**
3.10 Висновки до третього розділу **90**

РОЗДІЛ 4. ЕРГОНОМІКА ТА ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ВЕБ-СИСТЕМИ 91
4.1 Ергономіка системи 91
4.2 Економічне обґрунтування 95
4.3 Висновки до четвертого розділу 100
ЗАГАЛЬНІ ВИСНОВКИ 101
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 103
ДОДАТОК А 107
ДОДАТОК Б 120

ВСТУП

Сучасні виклики, зумовлені широкомасштабними руйнуваннями інфраструктури України внаслідок воєнних дій, стихійних лих та інших катастроф, потребують швидких та ефективних рішень у сфері технічного обстеження об'єктів. Значні обсяги пошкоджених будівель висувають високі вимоги до точності, швидкості та об'єктивності оцінки їхнього стану. У таких умовах автоматизація процесів експертного аналізу стає критично важливою для ефективного управління процесом відновлення.

Метою цієї роботи є створення веб-системи, яка автоматизує процес оцінювання пошкоджень будівельних конструкцій із залученням групи експертів. Програмний засіб покликаний забезпечити об'єктивність, узгодженість та прозорість рішень щодо визначення пріоритетів відновлення будівель.

Запропоноване рішення спрямоване на мінімізацію людського фактору в процесі аналізу стану об'єктів, скорочення часу на прийняття рішень і забезпечення масштабованості системи для використання у державних і приватних структурах, що займаються інженерними обстеженнями.

Актуальність теми підтверджується тим, що більшість існуючих методик ґрунтуються на ручному введенні, обробці та аналізі даних, що уповільнює процес і підвищує ймовірність помилок. Автоматизація цієї сфери має стратегічне значення для України в умовах післявоєнного відновлення.

Об'єктом дослідження є *процес експертного оцінювання технічного стану будівельних конструкцій*.

Предметом дослідження є *моделі, методи і засоби автоматизації експертної оцінки в межах веб-системи*.

Основними методами дослідження виступають *методи експертного оцінювання (метод Дельфі, парних порівнянь, аналіз ієрархій), програмна інженерія, методи проєктування клієнт-серверних інформаційних систем*.

У рамках дипломного проєкту було поставлено такі основні завдання:

- провести аналіз існуючих підходів до технічної оцінки будівельних конструкцій;
- сформувати набір критеріїв для оцінювання стану конструкцій;
- адаптувати обрані методи експертного аналізу до автоматизованої обробки;
- розробити архітектуру веб-системи з урахуванням масштабованості та модульності;
- реалізувати інтерфейс введення та перегляду оцінок, логіку обробки та базу даних;
- провести тестування системи в умовах, наближених до реальних, та оцінити її ефективність.

Сфера практичного застосування системи охоплює органи місцевого самоврядування, технічні служби, державні установи та приватні компанії, залучені до оцінки технічного стану об'єктів, а також потенційно може використовуватись в освітньому та науковому середовищі для навчальних чи дослідницьких цілей.

Короткий зміст розділів дипломної роботи:

- У *першому розділі* виконано огляд і класифікацію методів оцінки стану будівель, обґрунтовано необхідність автоматизації.
- *Другий розділ* присвячений проєктуванню архітектури системи, логічної структури, взаємодії між компонентами та вимог до даних.
- У *третьому розділі* описано реалізацію веб-додатку, його функціональність, алгоритми обробки та тестування.
- *Четвертий розділ* містить аналіз ергономіки інтерфейсу та техніко-економічне обґрунтування доцільності розробки.

РОЗДІЛ 1. АНАЛІЗ ІСТУЮЧИХ МЕТОДІВ ОЦІНКИ СТАНУ БУДІВЕЛЬНИХ КОНСТРУКЦІЙ

1.1 Аналіз причин пошкоджень будівельних конструкцій

Будівельні конструкції піддаються різноманітним впливам, які можуть спричинити їх пошкодження або навіть повне руйнування. Розуміння причин пошкоджень є критично важливим для їхньої оцінки, планування ремонту та підвищення стійкості будівель у майбутньому. У цьому розділі розглянемо основні причини пошкоджень будівельних конструкцій, серед яких можна виділити природні, техногенні та експлуатаційні фактори.

Природні фактори

- Стихійні лиха
  - Землетруси: Ударні хвилі, які виникають під час землетрусів, можуть призвести до руйнування фундаментів, тріщин у стінах і зсуву конструктивних елементів. Стійкість до землетрусів залежить від проєктування будівлі та її географічного розташування [1, с. 25].
  - Повені: Тривалий контакт із водою викликає ослаблення фундаментів, корозію металевих елементів та розмив ґрунту навколо конструкцій [2].
  - Урагани: Сильні вітри й удари об'єктів, принесених вітром, можуть пошкодити фасади, дахові конструкції та віконні системи.
- Кліматичні умови
  - Різкі зміни температури: Цикли заморожування та відтавання викликають утворення мікротріщин у бетоні, що поступово призводить до його руйнування [3].
  - Ультрафіолетове випромінювання: Тривала дія сонячного світла може знижувати міцність полімерних матеріалів та покрівельних покриттів.
  - Опади: Постійний вплив дощу або снігу призводить до корозії металів та зносу матеріалів, особливо у недостатньо захищених ділянках .

Техногенні фактори

- Помилки проєктування та будівництва
  - Недотримання будівельних норм: Використання низькоякісних матеріалів або невідповідність будівельних робіт проєктній документації [4].
  - Неправильні розрахунки навантажень: Недооцінка ваги конструкцій або зовнішніх навантажень, таких як сніг чи вітер, може спричинити перевантаження будівлі.
  - Порушення технології будівництва: Неправильне укладання бетонних сумішей, недостатня армування чи інші технологічні помилки.
- Вплив промислової діяльності
  - Вібрації: Будівлі поблизу виробничих підприємств або залізничних колій можуть зазнавати пошкоджень через постійні вібрації [5].
  - Викиди хімічних речовин: Агресивні гази та рідини можуть викликати корозію металів, пошкодження бетону та зниження міцності інших матеріалів.

Експлуатаційні фактори

- Неправильна експлуатація
  - Перевантаження конструкцій: Використання будівель не за призначенням, наприклад, складування важких предметів у невідповідних зонах.
  - Відсутність обслуговування: Несвоєчасна заміна пошкоджених елементів чи профілактичний ремонт можуть призводити до накопичення пошкоджень.
- Старіння матеріалів
  - Природне старіння: Матеріали мають обмежений термін служби, після якого вони втрачають свої експлуатаційні властивості.
  - Вплив часу: Навіть за відсутності активного використання конструкції можуть деградувати через постійний вплив навколишнього середовища [6].

Для довшої експлуатації будівельних конструкцій використовують різні методи мінімізації пошкоджень (табл. 1.1)

Таблиця 1.1

Методи мінімізації пошкоджень

Методи	Переваги методів мінімізації пошкоджень	
Удосконалення проєктування	Використання сучасних розрахункових методів для прогнозування поведінки конструкцій у різних умовах.	Врахування потенційних ризиків на етапі проєктування.
Якісні матеріали та технології	Застосування інноваційних матеріалів з підвищеною стійкістю до впливу зовнішніх факторів.	Контроль за дотриманням технологій під час будівництва.
Регулярне обслуговування	Проведення регулярних інспекцій для виявлення та усунення початкових пошкоджень.	Плановий ремонт і профілактика.
Навчання персоналу	Підвищення кваліфікації спеціалістів, відповідальних за будівництво та обслуговування будівель.	Поширення знань про новітні технології та методи оцінки стану конструкцій.

Аналіз причин пошкоджень будівельних конструкцій дозволяє не лише зрозуміти природу руйнувань, але й знайти ефективні способи їхньої мінімізації. Комплексний підхід, який враховує природні, техногенні та експлуатаційні фактори, є ключем до створення стійких і довговічних будівель, які здатні протистояти викликам сучасного світу.

1.2 Сучасні підходи до оцінки стану будівельних конструкцій

Оцінка стану будівельних конструкцій є основою для забезпечення їхньої надійності, безпеки та довговічності. Сучасні підходи охоплюють широкий спектр методів і технологій, які дозволяють отримати точні й об'єктивні результати для подальшого прийняття рішень. Вони включають традиційні, інструментальні, аналітичні, інноваційні та комплексні методи, що спрямовані на ефективне дослідження стану будівель.

1. Традиційні методи оцінки

- Візуальний огляд
  - Це перший етап обстеження, що виконується для виявлення явних пошкоджень, таких як тріщини, корозія, деформації, відшарування матеріалів чи зміщення елементів.
  - Метод застосовується для оцінки зовнішнього стану конструкцій і дозволяє визначити потребу в подальших дослідженнях.
  - Обмеження: виявлення лише поверхневих дефектів; суб’єктивний характер оцінки.
- Ручні вимірювання (рис. 1.1)
  - Використовуються вимірювальні інструменти (рулетки, рівні, теодоліти), щоб оцінити геометричні параметри конструкцій.
  - Дає змогу визначити відхилення від проєктних параметрів і деформації.

Рис. 1.1 Ручні вимірювання пошкоджень

- Фізичні випробування
  - Застосовуються для оцінки міцності матеріалів і конструктивних елементів шляхом експериментального впливу.
  - Приклади: випробування на вигин, стиск чи розтяг.
  - Часто є руйнівними і потребують часткової заміни випробуваних зразків.

2. Інструментальні методи

Сучасні інструментальні методи дозволяють виявляти внутрішні дефекти конструкцій без руйнування матеріалів.

- Ультразвукові методи (рис. 1.2) [7]
  - Застосування ультразвукових хвиль для виявлення тріщин, пустот та інших внутрішніх дефектів.
  - Переваги: неруйнівність, висока точність, можливість дослідження значної площі.
  - Області використання: залізобетонні, металеві та кам’яні конструкції.

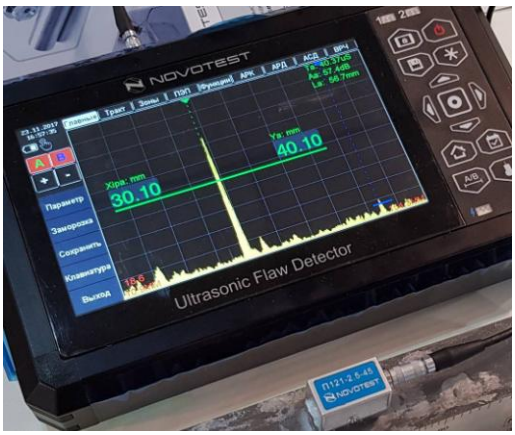


Рис 1.2 Прилад для ультразвукової оцінки

- Рентгенівська та гамма-радіографія
  - Методи дають змогу отримати зображення внутрішньої структури матеріалів, що дозволяє оцінити якість зварних з'єднань, стан арматури та наявність тріщин.
  - Вимоги: спеціальне обладнання та захисні заходи для роботи з радіоактивними матеріалами.

- Тепловізійна діагностика [8]
  - Використовує інфрачервоне випромінювання для виявлення зон тепловтрат, вологості або протікання.
  - Часто використовується для оцінки фасадів, покрівель і теплотрас.
- Методи акустичної емісії
  - Дозволяють фіксувати звукові сигнали, що виникають у конструкції при утворенні тріщин або інших дефектів.
  - Використовуються для моніторингу стану великих споруд, таких як мости чи дамби.

3. Аналітичні методи

- Математичне моделювання [9]
  - Застосовується для прогнозування поведінки конструкцій під час дії різних навантажень.
  - Найбільш поширений метод – метод кінцевих елементів (FEM), який дозволяє розраховувати напруження, деформації та резерви міцності.
- Аналіз довговічності
  - Передбачає розрахунок залишкового ресурсу конструкції на основі її віку, матеріалів, умов експлуатації та наявних дефектів.
- Прогнозування ризиків
  - Моделювання можливих сценаріїв руйнування конструкцій з урахуванням зовнішніх і внутрішніх факторів.

4. Інноваційні технології

Сучасні інноваційні підходи значно підвищують ефективність та точність оцінки стану конструкцій.

- Безпілотні літальні апарати (БПЛА) [10]
  - Використовуються для обстеження важкодоступних об’єктів, таких як висотні будівлі, мости чи вежі.

  - Обладнані камерами високої роздільної здатності або тепловізорами для отримання детальних зображень.
- 3D-сканування
  - Створює тривимірну модель об’єкта для оцінки його геометричних параметрів та пошуку деформацій.
  - Використовується для контролю стану історичних пам’яток і складних інженерних конструкцій.
- Інтернет речей (IoT)
  - Датчики, встановлені в конструкціях, дозволяють у реальному часі збирати дані про навантаження, температуру, вібрацію та інші параметри.
  - Забезпечує постійний моніторинг і оперативне виявлення відхилень.
- Штучний інтелект (ШІ) [16]
  - Застосовується для аналізу великих обсягів даних, отриманих під час обстежень, та для розробки рекомендацій з обслуговування конструкцій .

- Відстеження масштабних руйнувань, спричинених стихійними лихами, може здійснюватися за допомогою супутникових технологій. Супутники дозволяють отримувати актуальні зображення зон руйнувань, аналізувати масштаби пошкоджень та планувати дії з ліквідації наслідків (рис. 1.3) [11].

Рис 1.3 Фото наслідків пожежі

5. Комплексні підходи

Поєднання методів дозволяє отримати максимально повну інформацію про стан конструкції. Наприклад:

- Інтеграція даних, зібраних дронами, з результатами ультразвукових та тепловізійних досліджень.
- Створення динамічних моделей об’єктів із врахуванням даних з IoT-датчиків.

6. Регулярний моніторинг та профілактика

- Регулярні огляди – Забезпечення періодичного обстеження стану конструкцій для своєчасного виявлення дефектів.
- Профілактичні заходи – Виконання планового ремонту та заміни зношених елементів. – Використання захисних покриттів для зменшення впливу корозії чи інших негативних факторів.

Сучасні підходи до оцінки стану будівельних конструкцій поєднують традиційні методи з інноваційними технологіями, що дозволяє забезпечити їхню надійність і безпеку. Використання дронів, IoT, 3D-сканування та математичного моделювання забезпечує не лише високу точність діагностики, але й економію часу та ресурсів. Інтеграція цих технологій дозволяє створити ефективну систему моніторингу, яка забезпечує своєчасне прийняття рішень щодо ремонту чи реконструкції об'єктів.

1.3 Методи експертних оцінок у будівельній сфері

Експертні оцінки відіграють ключову роль у прийнятті рішень у будівельній сфері. Вони застосовуються для аналізу стану будівель, вибору технологій ремонту, визначення пріоритетності відновлювальних робіт і оцінки можливих ризиків. У цьому розділі розглянемо основні методи експертних оцінок, їх переваги, обмеження та особливості застосування.

1. Загальна характеристика експертних оцінок

Експертна оцінка — це процес, у якому група кваліфікованих фахівців аналізує складні ситуації або об’єкти, використовуючи свої знання, досвід і спеціальні методи. Основні етапи проведення експертної оцінки включають [12]:

1. Визначення мети оцінки. Чітке формулювання завдань, які потрібно вирішити.
2. Формування експертної групи. Вибір фахівців із відповідною кваліфікацією та досвідом.
3. Збір і аналіз даних. Отримання інформації про об'єкт оцінки за допомогою спостережень, вимірювань і консультацій.
4. Застосування методів оцінки. Використання конкретних підходів, які найкраще відповідають меті дослідження.
5. Обробка результатів. Узагальнення, аналіз і представлення висновків у зручній для використання формі.

2. Класифікація методів експертних оцінок

Методи експертних оцінок можна поділити на кілька основних груп:

1. Кількісні методи

   – Оцінка на основі числових показників, які відображають стан об'єкта чи процесу.

   – Приклади: ранжування, бальна шкала, матриця порівнянь.

2. Якісні методи

   – Висновки формуються на основі описів, аналізу ситуацій і думок експертів.

   – Використовуються у випадках, коли числові дані важко отримати.

3. Індивідуальні методи

– Оцінка здійснюється одним експертом.

– Обмеження: суб'єктивність і залежність від рівня кваліфікації експерта.

4. Колективні методи

– Оцінка проводиться групою експертів, що дозволяє зменшити суб’єктивність результатів.

– Приклади: метод Дельфі, комісійна оцінка.

3. Основні методи експертних оцінок

3.1. Метод Дельфі [13]

Метод Дельфі передбачає отримання узгоджених висновків від групи експертів через серію анонімних опитувань. Основні етапи:

1. Формулювання запитань. Експертам пропонуються питання, що потребують оцінки чи прогнозу.
2. Індивідуальні відповіді. Кожен експерт надає свої відповіді незалежно від інших.
3. Аналіз відповідей. Узагальнення результатів першого туру та створення зворотного зв'язку.
4. Повторне опитування. Експерти переглядають свої відповіді, враховуючи результати попереднього туру.
5. Досягнення консенсусу. Завершення процесу після кількох турів, коли думки експертів збігаються.

Переваги методу Дельфі:

- Анонімність знижує вплив авторитету окремих осіб
- Можливість отримати узгоджені висновки

Недоліки:

- Тривалість процесу
- Залежність від компетентності експертів

3.2. Метод парних порівнянь

Цей метод передбачає порівняння об'єктів або критеріїв попарно для визначення їхньої відносної важливості. Результати оформлюються у вигляді матриці порівнянь, де кожен елемент відображає перевагу одного об'єкта над іншим.

Етапи:

- Формування списку об'єктів
- Проведення попарних порівнянь експертами
- Розрахунок коефіцієнтів важливості

Переваги:

- Простота реалізації
- Дає чіткі результати

Недоліки:

- Обмежена кількість об’єктів для порівняння
- Чутливість до помилок в оцінках

3.3. Метод ранжування

Експерти розташовують об'єкти або критерії в порядку їхньої важливості. Наприклад, будівлі з найгіршим станом отримують найвищий пріоритет для ремонту.

Переваги:

- Простота й швидкість виконання
- Підходить для попередньої оцінки

Недоліки:

- Відсутність чітких числових результатів
- Висока суб’єктивність

3.4. Метод аналізу ієрархій

Цей метод використовується для прийняття рішень у складних ситуаціях. Він базується на побудові ієрархії критеріїв та альтернатив і визначенні їхньої важливості через матриці парних порівнянь.

Етапи:

- Побудова ієрархії (мета, критерії, альтернативи)
- Проведення парних порівнянь
- Розрахунок ваг критеріїв та альтернатив
- Прийняття рішення на основі отриманих ваг

Переваги:

- Структурований підхід до прийняття
- Можливість врахування численних факторів

Недоліки:

- Складність реалізації при великій кількості
- Вимога математичних розрахунків

4. Інноваційні підходи до експертних оцінок

1. Використання інформаційних систем – Спеціалізовані програмні продукти дозволяють автоматизувати процес оцінки, зменшуючи кількість помилок.
2. Моделі штучного інтелекту – Використання штучного інтелекту для аналізу великих обсягів даних і формування прогнозів.
3. Big Data – Аналіз великих даних дозволяє знаходити закономірності та робити більш точні прогнози.

Для вибору оптимального методу експертних оцінок необхідно порівняти їх за основними критеріями. Важливими аспектами аналізу є точність, об'єктивність, простота реалізації, швидкість виконання, масштабованість.

У таблиці нижче представлено порівняння чотирьох основних методів експертних оцінок: метод Дельфі, метод парних порівнянь, метод ранжування та

метод аналізу ієрархій . Оцінювання виконане на основі переваг і недоліків кожного методу, де позначка «+» вказує на позитивний аспект, а «-» – на обмеження (табл 1.2).

Таблиця 1.2

Таблиця порівняння методів експертних оцінок

Критерій / Метод	Точність	Об'єктивність	Простота реалізації	Швидкість виконання	Маштабованість
Метод Дельфі	+	+	-	-	-
Метод парних порівнянь	+	+	+	-	+
Метод зважених коефіцієнтів	-	-	+	+	+
Метод аналізу ієрархій	+	+	-	-	+

Проведений аналіз показує, що вибір методу експертного оцінювання залежить від поставленої задачі, специфіки завдання, доступних ресурсів та складності ситуації.

Таким чином, для оцінки складних об'єктів із великою кількістю критеріїв доцільно використовувати метод аналізу ієрархій, а для попередніх швидких оцінок підходить метод ранжування. У випадках, коли важливо досягти консенсусу серед експертів, ефективним є метод Дельфі.

Методи експертних оцінок у будівельній сфері забезпечують обґрунтованість рішень, сприяючи ефективному плануванню ремонтних і відновлювальних робіт.

1.4 Проблеми існуючих підходів та обгрунтування необхідності автоматизації

Існуючі підходи до оцінки стану будівельних конструкцій мають низку недоліків, які знижують їх ефективність, точність та зручність використання. У багатьох випадках ці проблеми є наслідком застарілих методик, відсутності стандартизації та обмеженого застосування сучасних цифрових інструментів. Окрім того, збільшення обсягів пошкоджених будівель унаслідок природних катастроф, військових дій або інших зовнішніх впливів висуває високі вимоги до швидкості й об'єктивності процесу оцінки. Нижче розглянуто основні проблеми існуючих підходів та обґрунтовано необхідність автоматизації.

Низька ефективність і трудомісткість процесу оцінки

Традиційні методи оцінки стану будівельних конструкцій значною мірою покладаються на ручну працю експертів, що вимагає великих витрат часу та ресурсів [14]. Експерти, які проводять оцінку, змушені аналізувати значний обсяг даних, оглядати пошкоджені об’єкти, заповнювати звіти та виконувати численні розрахунки . У разі великих масштабів пошкоджень, таких як наслідки воєнних дій чи стихійних лих, цей підхід є надзвичайно повільним і часто не дозволяє вчасно визначити пріоритети для проведення ремонтних робіт.

Висока суб’єктивність оцінки

Експертні оцінки значною мірою залежать від досвіду та компетентності окремих фахівців, що може призводити до різниці в результатах навіть за однакових умов. Суб’єктивність оцінювання може викликати суперечності між експертами та знижувати точність прийняття рішень. Відсутність єдиного стандарту для оцінювання пошкоджень також ускладнює процес порівняння та узгодження результатів .

Відсутність стандартизації

В Україні та багатьох інших країнах досі немає уніфікованих підходів до класифікації пошкоджень будівельних конструкцій . Це ускладнює процес передачі інформації між різними установами, а також створює бар’єри для використання сучасних технологій. Відсутність стандартизованих критеріїв оцінки також ускладнює навчання нових експертів і створює залежність від окремих досвідчених фахівців.

Недостатній рівень цифровізації

У багатьох випадках обробка даних здійснюється вручну, що підвищує ризик помилок і значно ускладнює аналіз великих обсягів інформації. Ручне введення даних у таблиці чи паперові форми, відсутність автоматичних алгоритмів обробки інформації, а також нездатність інтегрувати різні джерела даних уповільнюють процес прийняття рішень.

Відсутність інтеграції з сучасними технологіями

Сучасні інформаційні технології, такі як штучний інтелект, машинне навчання, геоінформаційні системи (ГІС) та хмарні платформи, можуть значно підвищити точність і швидкість оцінки стану будівель. Проте більшість існуючих методів не використовують цих можливостей, що обмежує їх потенціал. Крім того, відсутність інтеграції з мобільними пристроями та онлайн-платформами ускладнює процес збору даних і комунікації між експертами (рис. 1.8) [15].

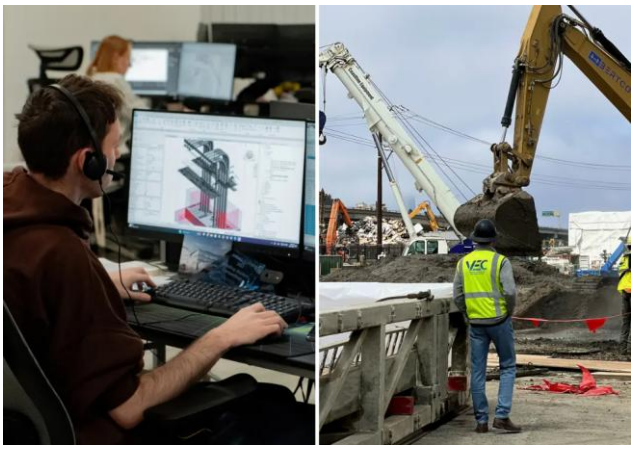


Рис. 1.8 – Інтеграція з технологіями

Складність управління великими обсягами даних

Оцінка стану будівельних конструкцій часто вимагає обробки великих обсягів інформації, включаючи фотографії, звіти, креслення та технічну документацію. Без застосування автоматизованих систем організація, зберігання та аналіз цих даних стає важким завданням. До того ж, пошук необхідної інформації в разі виникнення запитів від зацікавлених сторін може займати значний час.

Потреба у пріоритизації робіт

У разі великої кількості пошкоджених об'єктів, таких як унаслідок війни чи природної катастрофи, традиційні підходи не дозволяють швидко визначити, які об'єкти потребують першочергового ремонту. Це може призводити до непродуктивного використання ресурсів і збільшення часу відновлення інфраструктури .

Обґрунтування необхідності автоматизації

Автоматизація процесу оцінки стану будівельних конструкцій дозволяє вирішити більшість зазначених проблем.

По-перше, використання спеціалізованих програмних продуктів значно знижує трудомісткість і підвищує швидкість оцінки. Експерти можуть вводити дані в систему, яка автоматично обробляє їх, формує звіти та пропонує пріоритетність ремонтів.

По-друге, автоматизовані системи мінімізують суб’єктивність оцінки, оскільки використовують стандартизовані критерії та математичні моделі. Це сприяє підвищенню точності й об’єктивності результатів. Впровадження таких систем також дозволяє знизити залежність від досвіду окремих фахівців.

По-третє, автоматизація сприяє кращій організації й аналізу великих обсягів даних. Інтеграція баз даних, хмарних сховищ і ГІС дає змогу зберігати й аналізувати інформацію в єдиному середовищі, забезпечуючи швидкий доступ до потрібних даних.

Нарешті, автоматизація дозволяє використовувати сучасні технології, такі як алгоритми машинного навчання, для прогнозування стану конструкцій, аналізу ризиків і пріоритизації ремонтних робіт. Це значно підвищує ефективність процесу відновлення будівель і дозволяє уникнути помилок, які можуть виникати при використанні традиційних методів.

Проблеми, що існують у традиційних підходах до оцінки стану будівельних конструкцій, обумовлюють нагальну потребу в їх модернізації. Автоматизація процесу оцінки є ефективним рішенням, яке дозволяє підвищити швидкість, точність і об’єктивність прийняття рішень. Використання сучасних інформаційних технологій сприяє оптимізації ресурсів і забезпечує ефективне управління відновлювальними роботами.

1.5 Висновок до першого розділу

У ході першого розділу було визначено основні проблеми, пов’язані з оцінкою стану будівельних конструкцій та визначенням пріоритетності їх ремонту. Проаналізовано сучасні методи експертного оцінювання та обґрунтовано необхідність автоматизації цього процесу.

На основі проведеного аналізу сформульовано основні завдання, які необхідно вирішити для розробки веб-системи автоматизованої оцінки будівельних конструкцій:

1. Розробка методології оцінки – визначення та адаптація методів експертного аналізу (метод Дельфі, метод парних порівнянь, аналіз ієрархій) для використання у веб-системі.
2. Проєктування архітектури системи – розробка структурної моделі веб-додатку, що включає фронтенд, бекенд та базу даних, із забезпеченням масштабованості та гнучкості.
3. Реалізація програмного забезпечення – створення веб-системи на основі стеку технологій React, TypeScript та PostgreSQL для обробки та аналізу введених експертами даних.

4. Оптимізація взаємодії користувача із системою – розробка інтуїтивного інтерфейсу для ефективного введення даних та перегляду результатів.
5. Тестування та валідація – перевірка працездатності системи, аналіз її точності та ефективності у вирішенні поставлених задач.

Очікувані результати

1. Автоматизація оцінки стану будівель – зниження витрат часу на аналіз пошкоджень завдяки алгоритмічному обробленню експертних даних.
2. Підвищення точності рішень – використання об’єктивних математичних методів для мінімізації суб’єктивних похибок експертів.
3. Прозорість і надійність – система забезпечить збереження історії оцінок та надання обґрунтованих рекомендацій для розподілу ресурсів.
4. Масштабованість рішення – можливість адаптації системи для інших сфер експертного оцінювання.

Переваги впровадження

- Зменшення впливу людського фактору на ухвалення рішень.
- Скорочення часу на оцінку об’єктів та планування ремонтних робіт.
- Підвищення ефективності управління відбудовою завдяки чіткій систематизації даних.
- Сприяння відновленню інфраструктури України шляхом раціонального розподілу ресурсів.

Таким чином, реалізація веб-системи дозволить значно покращити якість та швидкість оцінки стану будівельних конструкцій, сприяючи ефективному управлінню процесами відновлення інфраструктури.

РОЗДІЛ 2. МЕТОДОЛОГІЯ ПРОЄКТУВАННЯ ТА ВИБІР ІНСТРУМЕНТІВ РЕАЛІЗАЦІЇ СИСТЕМИ

2.1 Обґрунтування архітектурних рішень веб-системи

Проєктування архітектури програмної системи є критично важливим етапом розробки, оскільки саме на цьому рівні визначаються ключові компоненти, їхні функції та взаємозв'язки. Для створення веб-системи автоматизованої оцінки дефектів будівельних конструкцій потрібно забезпечити ефективність обробки даних, підтримку багатокористувацької роботи, масштабованість і безпеку.

У результаті аналізу можливих варіантів було обрано **трирівневу клієнт-серверну архітектуру [**17, 18], яка відповідає поставленим вимогам. Вона дозволяє розподілити завдання між трьома рівнями системи, кожен з яких має свої функціональні обов'язки. Нижче наведено таблицю, яка демонструє змістовне наповнення кожного рівня системи та його роль у загальній архітектурі (табл 2.1).

Таблиця 2.1

Опис основних рівнів трирівневої архітектури

Рівень системи	Функціональне призначення
Клієнтський (Frontend)	Забезпечує взаємодію користувача з системою: введення даних, перегляд результатів, формування запитів
Серверний (Backend)	Виконує обробку даних, застосовує методи експертних оцінок, координує обмін інформацією між клієнтом та базою даних
Рівень бази даних (Database)	Відповідає за зберігання даних: інформація про об'єкти, оцінки експертів, результати аналізу

Ця таблиця підкреслює, що кожен рівень виконує чітко визначені функції, що дозволяє уникнути дублювання процесів та забезпечити логічну структурованість системи. Важливо зазначити, що розподіл завдань між рівнями спрощує як

розробку, так і подальшу підтримку системи, оскільки модифікації одного рівня не потребують змін в інших.

Вибір такої архітектури обґрунтовується низкою переваг, які дозволяють системі ефективно працювати в умовах реального використання. Для більшої наочності їх наведено в таблиці (табл 2.2).

Таблиця 2.2

Переваги трирівневої архітектури

Перевага	Опис
Розділення відповідальності	Кожен рівень займається окремим завданням, що знижує складність підтримки
Масштабованість	Можливість масштабувати рівні окремо залежно від навантаження
Підвищена безпека	Розмежування доступу дозволяє застосувати різні політики безпеки
Зручність супроводу	Полегшення пошуку та усунення помилок завдяки модульності
Підтримка багатокористувацького доступу	Система обробляє запити від багатьох користувачів одночасно

Як видно з таблиці 2.2, обрана архітектура дозволяє не лише розділити функціонал системи, але й забезпечує високий рівень надійності та гнучкості.

Наприклад, якщо в процесі експлуатації виникне необхідність оновлення інтерфейсу, це можна буде зробити без змін у серверній логіці чи базі даних. Такий підхід критично важливий для систем, які розраховані на довготривале використання з поступовим нарощенням функціоналу.

Щоб краще зрозуміти доцільність вибору саме трирівневої архітектури, було проведено аналіз альтернативних підходів (табл 2.3).

Таблиця 2.3

Порівняння варіантів архітектури

Варіант архітектури	**Переваги**	**Недоліки**
Монолітна	Простота реалізації, менші витрати на розробку	Складність масштабування, проблеми при оновленні
Мікросервісна	Висока гнучкість, незалежний розвиток модулів	Складність координації, вищі витрати на підтримку
Трирівнева	Баланс між складністю, масштабованістю та безпекою	Потребує координації між рівнями

Як видно з таблиці, монолітна архітектура могла б здатися привабливою через простоту реалізації, однак її обмеження в масштабованості роблять її менш придатною для систем, які планують розширюватися.

З іншого боку, мікросервісна архітектура хоч і надає високу гнучкість, проте є складною для впровадження в рамках обмеженого бюджету. Трирівнева архітектура виступає золотою серединою, що поєднує достатню масштабованість з прийнятною складністю реалізації.

Окрім основної структури, важливо також розглянути можливості майбутнього розвитку системи. Обрана архітектура дозволяє без суттєвих змін додавати нові функціональні модулі (табл 2.4).

Таблиця 2.4

Потенційні напрямки розширення системи

Напрямок	Призначення
API-інтерфейси	Забезпечення взаємодії з іншими програмними комплексами
Мобільний додаток	Надання доступу до системи з мобільних пристроїв
Аналітичний модуль	Формування звітів, візуалізація даних, статистичний аналіз
Система сповіщень	Інформування користувачів про оновлення чи нові дані

Наведена таблиця демонструє, що система є відкритою для розвитку. Наприклад, додавання API дозволить інтегрувати веб-систему з державними чи муніципальними реєстрами для автоматичного імпорту даних про об’єкти, а мобільний додаток забезпечить можливість введення інформації безпосередньо на місці огляду.

Таким чином, обрана архітектура веб-системи забезпечує **стійкий фундамент для реалізації ключових функцій, підтримки багатокористувацького режиму, забезпечення безпеки та гнучкості в розвитку**. Вона дозволяє не лише задовольнити поточні вимоги проєкту, але й передбачає простий шлях адаптації системи під майбутні задачі.

2.2 Побудова логічної структури веб-системи

Логічна структура веб-системи [19] — це концептуальне відображення взаємозв’язків між основними її компонентами, визначення ролей, функцій та потоків даних. Вона демонструє, як елементи системи взаємодіють між собою для досягнення спільної мети: автоматизованої оцінки дефектів будівельних конструкцій.

Проєктування логічної структури базується на результатах архітектурного обґрунтування і деталізує компоненти системи, їхні функції та зв’язки. Логічна структура [20] не залежить від конкретних технологій, що робить її універсальною для подальшої реалізації на різних програмних платформах.

Основні компоненти логічної структури

Веб-система складається з кількох ключових компонентів, кожен з яких виконує свою роль у процесі збору, обробки та відображення інформації (табл 2.5).

Таблиця 2.5

Основні компоненти логічної структури системи

Компонент	Функціональне призначення
Інтерфейс введення даних	Приймає інформацію про дефекти будівель від користувача (експерта)
Модуль перевірки даних	Перевіряє правильність та повноту введених даних
Блок обробки оцінок	Застосовує методи експертних оцінок (Дельфі, парних порівнянь, аналіз ієрархій)
Модуль формування результатів	Узагальнює результати, формує пріоритетність об’єктів для ремонту
Сховище даних	Зберігає дані про об’єкти, оцінки, результати
Інтерфейс перегляду результатів	Надає користувачеві доступ до підсумкових даних, звітів, графіків

Як видно з таблиці, кожен компонент виконує окрему функцію, однак усі вони взаємопов’язані та утворюють єдиний логічний ланцюг. Взаємодія між компонентами забезпечує безперервний потік даних від введення до отримання результатів.

Опис потоків даних

Рух даних у системі відбувається за наступною схемою:

1. Користувач через інтерфейс введення даних заповнює інформацію про об’єкт оцінки (характеристики будівлі, опис дефектів, фотофіксація тощо).
2. Дані передаються у **модуль перевірки**, де відбувається валідація (перевірка на заповненість, формат, відповідність вимогам).
3. Після перевірки дані надходять у **блок обробки оцінок**, де застосовуються обрані методи експертного аналізу.
4. Оброблені результати передаються у **модуль формування результатів**, який розраховує підсумкові оцінки та формує пріоритетність об’єктів.
5. Підсумкові дані зберігаються у **сховище даних**.
6. Користувач через **інтерфейс перегляду результатів** отримує доступ до інформації: таблиць, графіків, звітів.

Опис функціональних взаємозв’язків

Логічна структура також передбачає взаємодію між окремими компонентами для забезпечення цілісності системи. Для цього компоненти розподілені на **функціональні блоки**, кожен з яких виконує окреме завдання.

Таблиця 2.6

Функціональні блоки системи

Функціональний блок	**Складові модулі**
Блок введення	Інтерфейс введення даних, модуль перевірки даних
Блок обробки	Блок обробки оцінок, модуль формування результатів
Блок авторизаії	Блок для авторизації та входу у систему
Блок відображення	Інтерфейс перегляду результатів
Блок зберігання	Сховище даних

Такий поділ дозволяє **уніфікувати логіку роботи системи** та зробити її більш зручною для подальшої реалізації. Наприклад, блок введення може бути реалізований у вигляді веб-форми, тоді як блок відображення — у вигляді інтерактивної таблиці з можливістю фільтрації.

Принципи побудови логічної структури

Логічна структура веб-системи базується на наступних принципах:

- **модульність** — кожен компонент виконує окреме завдання;
- **масштабованість** — можливість додавати нові модулі без порушення роботи існуючих;
- **гнучкість** — можливість адаптації структури під зміну вимог користувачів;
- **незалежність від технологій** — логіка системи не залежить від конкретних засобів реалізації.

Завдяки цим принципам система має потенціал до розвитку та адаптації без необхідності радикальної перебудови.

Практичні сценарії використання логічної структури

Щоб підкреслити значення логічної структури, можна навести кілька прикладів її практичного використання:

1. **Використання на місцях обстеження будівель** — експерт за допомогою інтерфейсу введення даних фіксує пошкодження у польових умовах; дані одразу надходять на сервер для обробки.
2. **Робота в офісі** — інший користувач аналізує результати оцінки через інтерфейс перегляду, формує звіт для подання керівництву.
3. **Автоматичне оновлення даних** — дані про об’єкти оновлюються через API з інших систем, інтегруються в сховище та відображаються в системі.

Ці сценарії демонструють гнучкість логічної структури, яка дозволяє використовувати систему в різних умовах без зміни її основної логіки.

Таким чином, побудова логічної структури веб-системи є критично важливим етапом, який закладає основу для її подальшої реалізації, незалежно від обраних технологій. Запропонована структура забезпечує чіткий поділ функцій, послідовність обробки даних та можливість подальшого розширення функціоналу системи.

2.3 Опис основних функціональних компонентів системи

Функціональні компоненти веб-системи автоматизованої оцінки дефектів будівельних конструкцій забезпечують виконання ключових завдань від введення даних до формування результатів оцінки. Кожен компонент системи має чітко визначену роль, а їх взаємодія формує цілісний процес обробки даних [22].

Далі представлено загальний вигляд основних компонентів та їх взаємозв'язків (рис 2.1).

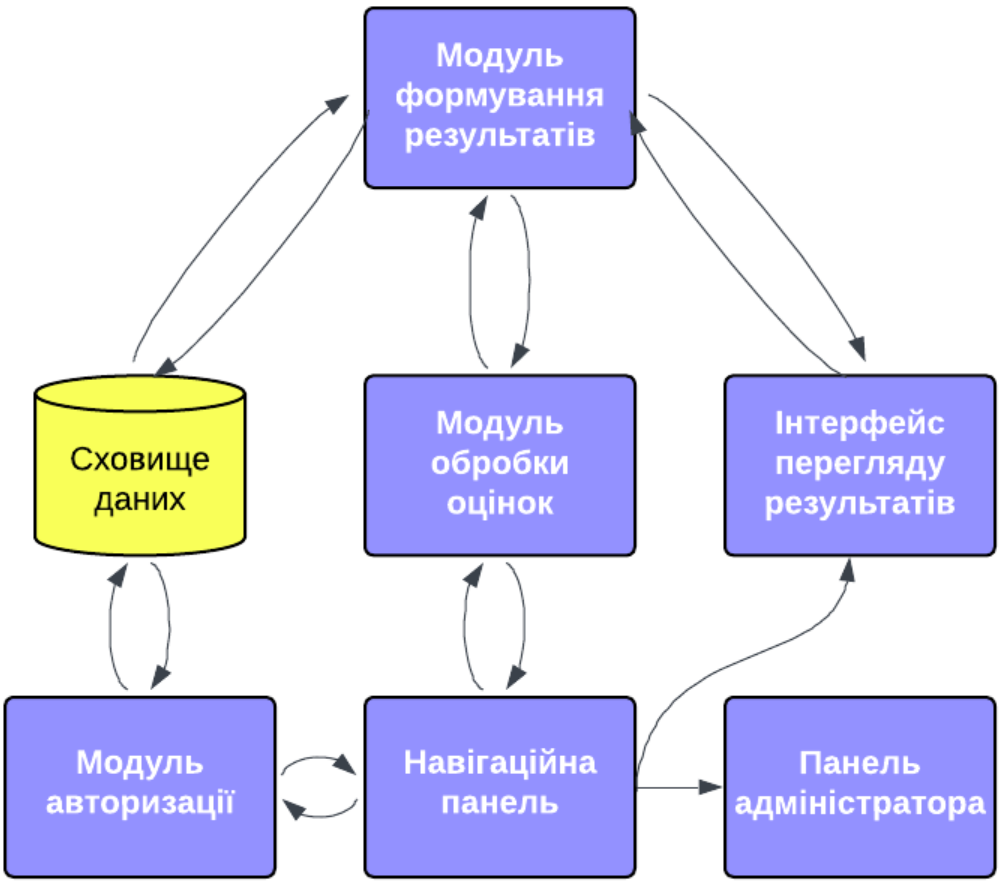


Рисунок 2.1 – Узагальнена концептуальна модель системи

З рисунку видно, що система складається з таких основних компонентів:

1. **Інтерфейс введення даних**
2. **Модуль авторизації**
3. **Модуль обробки оцінок**
4. **Модуль формування результатів**
5. **Інтерфейс перегляду результатів**
6. **Сховище даних**
7. **Панель адміністратора**

Інтерфейс введення даних

Інтерфейс введення даних є першим етапом взаємодії користувача із системою. Він забезпечує заповнення форм, прикріплення фотографій, введення опису дефектів. Інтерфейс має механізми перевірки коректності введених даних і попереджає про помилки.

Таблиця 2.7

Функції інтерфейсу введення даних

Функція	Опис
Текстові поля	Введення адреси, опису дефектів
Валідація даних	Автоматична перевірка формату введених даних
Підказки	Відображення допоміжної інформації під час заповнення форм

Для кращої наочності і розуміння ієрархії функціональних можливостей системи доцільно представити її структуру у вигляді дерева функцій (рис. 2.2). Такий підхід дозволяє відобразити основні дії користувача та логіку роботи модулів у розрізі цілей системи.

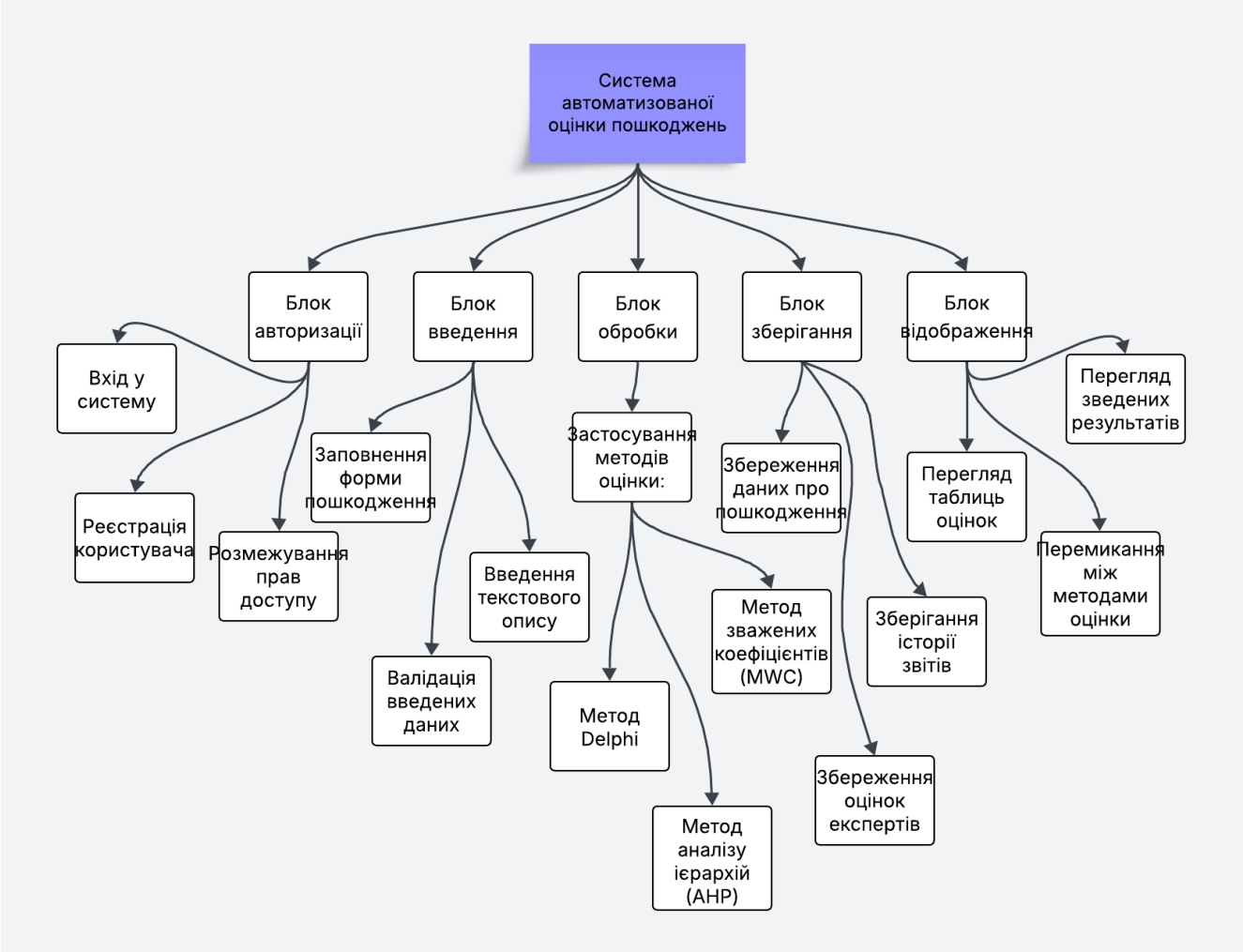


Рисунок 2.2 – Дерево функцій веб-системи

Такий поділ дозволяє уніфікувати логіку роботи системи та зробити її більш зручною для подальшої реалізації. Наприклад, блок введення може бути реалізований у вигляді веб-форми, тоді як блок відображення — у вигляді інтерактивної таблиці з можливістю фільтрації.

Модуль авторизації

Цей модуль відповідає за перевірку облікових даних користувача при вході в систему. Він здійснює валідацію логіна та пароля, визначає роль користувача (адміністратор або експерт) і забезпечує доступ лише до відповідного функціоналу. У разі помилки в авторизації система повідомляє користувача про некоректні дані. Такий підхід дозволяє розмежувати права доступу та захищає систему від несанкціонованого використання.

Модуль обробки оцінок

Цей компонент реалізує обробку даних за допомогою експертних методів (метод Дельфі, метод парних порівнянь, аналіз ієрархій). Він обчислює підсумкові оцінки, визначає пріоритетність об’єктів.

На **рисунку 2.3** представлено, як інформація проходить через модулі системи, включаючи обробку оцінок (рис. 2.3).

Панель адміністратора

Цей модуль надає розширений функціонал користувачам з роллю адміністратора. Він дозволяє створювати, редагувати та видаляти облікові записи користувачів, а також керувати переліком пошкоджень, доступних для оцінювання.

Через інтерфейс панелі адміністратор має змогу очищати звіти, що пов’язані з окремими користувачами або пошкодженнями, та переглядати загальну статистику введених оцінок. Модуль побудовано з урахуванням безпеки доступу та зручності навігації, що забезпечує ефективне управління даними в системі.

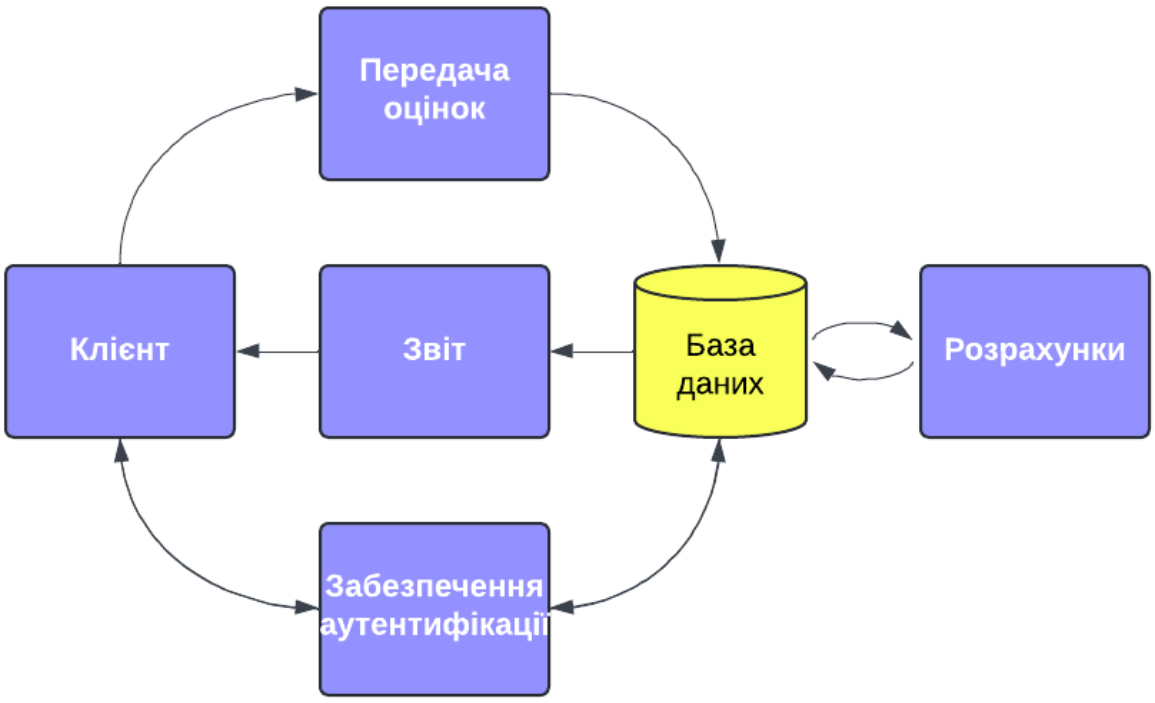


Рисунок 2.3 – Діаграма потоків даних

Модуль обробки оцінок реалізує такі функції (табл 2.8):

Таблиця 2.8

Методи обробки експертних оцінок

Метод	**Опис**
Метод Дельфі	Анонімне багатоетапне опитування експертів з метою досягнення узгодженості
Метод аналізу ієрархій (AHP)	Побудова ієрархічної структури критеріїв та обчислення ваг об’єктів
Метод зважених коефіцієнтів (MWC)	Обчислення загальних оцінок на основі експертних балів та вагових коефіцієнтів

Модуль обробки оцінок забезпечує гнучкість системи: користувач або адміністратор можуть вибрати метод, найбільш придатний для конкретної задачі.

Модуль формування результатів

Цей модуль відповідає за обробку зібраних експертних оцінок та виведення підсумкових результатів на основі обраного методу — Дельфі, аналізу ієрархій або зважених коефіцієнтів. Після обчислення вагових значень результати формуються у вигляді зручних для перегляду таблиць. (табл 2.9).

Таблиця 2.9

Функції модуля формування результатів

Функція	**Опис**
Формування таблиць	Побудова результатів обробки у вигляді табличних структур
Підрахунок вагових коефіцієнтів	Обчислення пріоритетів об’єктів за відповідною методикою
Підключення до API	Отримання оцінок із бази для подальшої обробки

Модуль формування результатів підвищує ефективність підготовки документів, полегшує роботу користувачів.

Інтерфейс перегляду результатів

Інтерфейс реалізовано у вигляді окремих сторінок для кожного методу. Після запуску обробки користувач бачить підсумкову таблицю з ваговими коефіцієнтами. (табл 2.10).

Таблиця 2.10

Функції інтерфейсу перегляду результатів

Функція	Опис
Перегляд таблиць	Відображення підсумкових результатів за методом
Вивід пріоритетів	Показ ранжування пошкоджень за результатами
Навігація між методами	Перехід між модулями Delphi, AHP, MWC через меню

Сховище даних

Сховище даних у проєктованій веб-системі виконує роль централізованого інформаційного середовища, де зберігаються всі ключові об'єкти: дані користувачів, опис пошкоджень конструкцій і результати експертного оцінювання. З метою забезпечення надійності, масштабованості та цілісності даних передбачається побудова бази даних із трирівневою структурою: **інфологічна модель**, **даталогічна модель** та **фізична модель**.

Інфологічна модель (концептуальний рівень) (рис. 2.4)

На концептуальному рівні система оперує трьома ключовими сутностями:

- **Користувач (User)** – особа, яка взаємодіє із системою (експерт або адміністратор);
- **Пошкодження (Damage)** – об'єкт оцінки, що містить опис дефекту конструкції;

- **Оцінка (Report)** – результат, що відображає експертну оцінку конкретного пошкодження.

Зв’язки між сутностями:

- Один користувач може надати багато оцінок;
- Кожне пошкодження може мати багато оцінок;

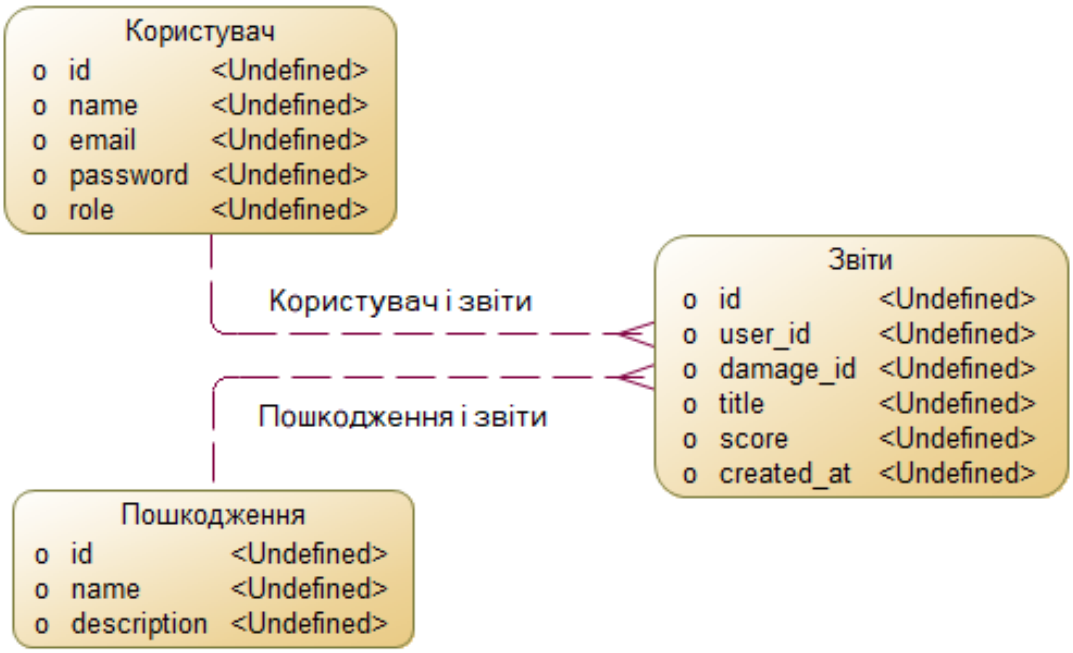


Рисунок 2.4 – Інфологічна модель даних системи

Даталогічна модель (логічна структура бази даних) (рис. 2.5)

На цьому рівні здійснюється перехід від абстрактних сутностей до структурованих таблиць.

- **users** – id, name, email, password, role
- **damages** – id, name, description
- **reports** – id, user_id, damage_id, title, score, created_at

Встановлюються зовнішні ключі:

- reports.user_id → users.id
- reports.damage_id → damages.id

Це дозволяє реалізувати зв’язки між таблицями та забезпечити логічну цілісність даних.

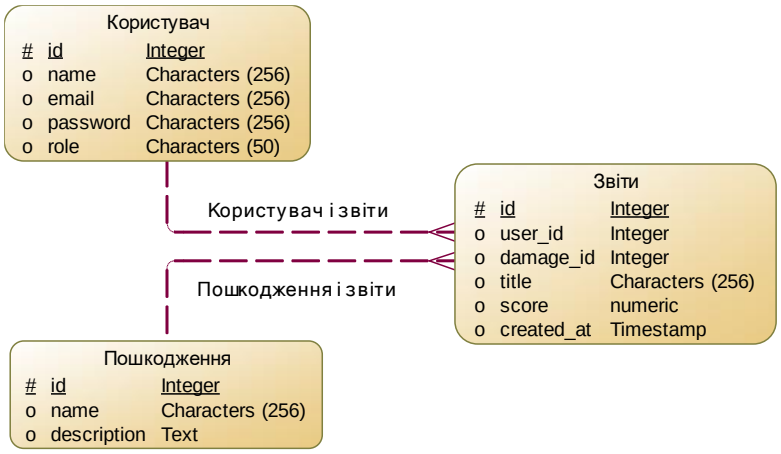


Рисунок 2.5 – Даталогічна модель бази даних

Фізична модель (рівень реалізації в СУБД) (рис. 2.6)

На фізичному рівні проєктування реалізація бази даних передбачається в межах сучасної реляційної системи керування базами даних (наприклад, PostgreSQL). Фізична модель відображає безпосередню реалізацію структури таблиць із урахуванням особливостей обраного середовища зберігання та обробки даних.

Передбачено реалізацію таких функціональних можливостей:

- підтримка зовнішніх ключів із директивою ON DELETE CASCADE, що забезпечує збереження цілісності зв’язків при видаленні пов’язаних записів;
- перевірка допустимих діапазонів значень
- автоматичне збереження дати й часу створення записів за допомогою поля created_at із DEFAULT CURRENT_TIMESTAMP;

Нижче наведено SQL-код, що ілюструє приклад фізичної реалізації основних таблиць системи у середовищі PostgreSQL:

```
-- Таблиця: Користувачі
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  name VARCHAR(256) NOT NULL,
  email VARCHAR(256) NOT NULL UNIQUE,
  password VARCHAR(256) NOT NULL,
  role VARCHAR(50) NOT NULL
);
```

```
-- Таблиця: Пошкодження
CREATE TABLE damages (
  id SERIAL PRIMARY KEY,
  name VARCHAR(256) NOT NULL,
  description TEXT
);

-- Таблиця: Звіти
CREATE TABLE reports (
  id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
  damage_id INTEGER REFERENCES damages(id) ON DELETE CASCADE,
  title VARCHAR(256),
  score NUMERIC CHECK (score >= 0 AND score <= 10),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

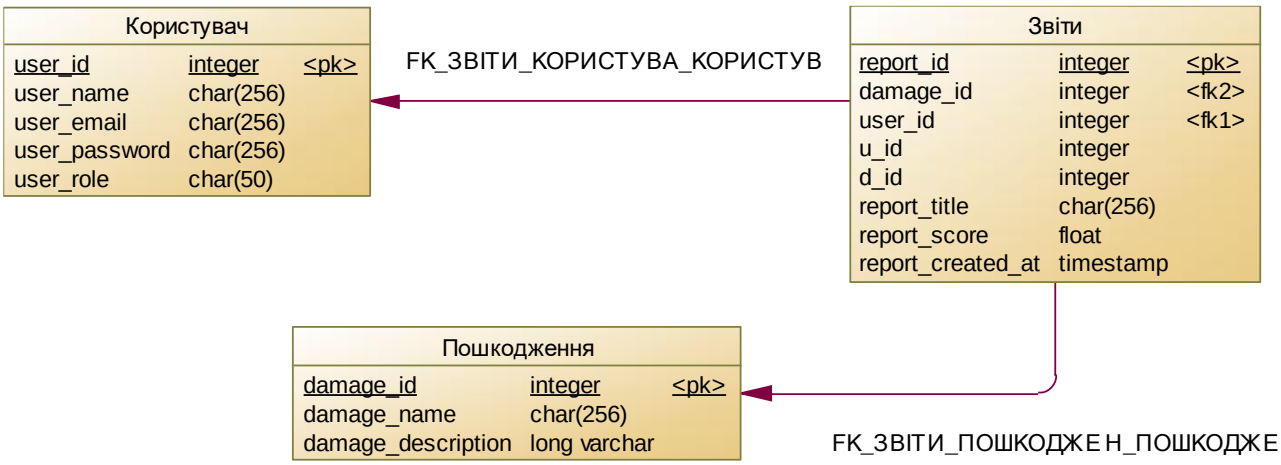


Рисунок 2.6 – Фізична модель таблиці звітів (reports)

Розробка фізичної моделі ґрунтується на принципах модульності, нормалізації та масштабованості. Вона є завершальним етапом проєктування сховища даних, що забезпечує повну відповідність логічній структурі, узгодженість з архітектурою системи та гнучкість для адаптації під будь-яке обране середовище реалізації.

2.4 Розробка взаємодії між компонентами системи

Взаємодія між компонентами веб-системи автоматизованої оцінки дефектів будівельних конструкцій є критичним фактором для забезпечення її ефективного функціонування. Правильно спроєктовані зв’язки дозволяють координувати роботу модулів, гарантують цілісність обробки даних і підтримують логіку бізнес-процесів.

У межах цієї системи кожен компонент виконує чітко визначену функцію, однак їхня робота можлива лише за умови **узгодженої взаємодії через обмін даними і командами**. Організація такої взаємодії базується на принципах сучасного проєктування програмних систем [23].

Принципи взаємодії компонентів

Взаємодія між компонентами системи ґрунтується на таких ключових принципах:

- **Інкапсуляція** – кожен модуль приховує власну реалізацію, взаємодіючи із зовнішнім світом через чітко визначені інтерфейси.
- **Чіткість інтерфейсів** – модулі обмінюються даними у заздалегідь визначених форматах.
- **Модульність** – зміни всередині одного модуля не впливають на інші.
- **Гнучкість** – можливість підключення нових модулів без зміни структури всієї системи.
- **Скалярність** – взаємодія адаптована до зростання обсягів даних і кількості користувачів.

Ці принципи дозволяють створити **масштабовану і зручну для підтримки систему**, де зміни чи оновлення одного модуля не порушують функціональність інших.

Етапи взаємодії між компонентами

Процес взаємодії між модулями можна розбити на послідовні етапи:

1. **Введення даних** – користувач заповнює форму, додає фото.
2. **Перевірка даних** – модуль перевірки контролює заповнення, коректність форматів.
3. **Обробка даних** – модуль обробки оцінок застосовує обраний метод аналізу.
4. **Формування результатів** – створюється звіт, генерується підсумкова таблиця.
5. **Збереження результатів** – дані записуються у сховище.
6. **Перегляд результатів** – користувач отримує доступ до результатів через інтерфейс.

Кожен етап передбачає **чіткий обмін даними між компонентами**, що підтримується стандартними протоколами та інтерфейсами.

Основні напрямки взаємодії (табл 2.11)

Таблиця 2.11

Основні напрямки взаємодії між компонентами системи

Джерело	Приймач	Дані
Інтерфейс введення	Модуль перевірки	Введені текстові, числові, дані
Модуль перевірки	Модуль обробки	Перевірені дані, сигнали про успіх
Модуль обробки	Модуль формування результатів	Оброблені оцінки, проміжні результати
Модуль формування результатів	Сховище даних	Звіти, таблиці
Сховище даних	Інтерфейс перегляду	Архівні та актуальні дані

Кожен напрямок взаємодії супроводжується **підтвердженням успішності обробки** або повідомленням про помилки, що важливо для контролю цілісності даних.

Методи обміну даними

В умовах веб-застосунку взаємодія компонентів організована через **HTTP(S)-запити**, **REST API**, а також через **SQL-запити** для взаємодії із базою даних (табл 2. 12).

Таблиця 2.12

Протоколи і методи обміну даними між компонентами

Компоненти взаємодії	Протокол
Інтерфейс введення – Модуль перевірки	REST API
Модуль перевірки – Модуль обробки	Внутрішній API
Модуль обробки – Модуль формування результатів	Локальні виклики методів
Модуль формування – Сховище даних	SQL-запити
Сховище даних – Інтерфейс перегляду	SQL, REST API

Такий підхід дозволяє **розділити логіку взаємодії від реалізації функціоналу**, що полегшує підтримку системи.

Особливості обробки помилок

Важливою складовою взаємодії є **обробка помилок і виняткових ситуацій**. Кожен модуль має механізми повідомлення про проблеми на попередньому етапі. Наприклад:

- Якщо **модуль перевірки** виявляє помилки, дані повертаються в інтерфейс введення для виправлення.
- Якщо **модуль обробки** стикається з помилками розрахунку (наприклад, відсутність даних), формується повідомлення адміністратору.

- Якщо **сховище даних** недоступне, результати зберігаються локально з наступною синхронізацією.

Ці механізми забезпечують **стійкість системи до збоїв**.

Переваги запропонованої взаємодії (табл 2.13)

Таблиця 2.13

Переваги взаємодії компонентів системи

Перевага	Опис
Надійність	Кожен модуль контролює стан даних і сигналізує про помилки
Масштабованість	Додаткові модулі можна підключати без зміни основної структури
Гнучкість	Можливість адаптації під різні платформи і технології
Незалежність	Компоненти можуть оновлюватися окремо
Прозорість	Відслідковування потоку даних на кожному етапі

Сценарії використання системи

Запропонована взаємодія дозволяє системі ефективно працювати у різних ситуаціях. Наведемо приклади:

1. **Робота на місці пошкодження будівлі**: експерт вводить дані зі смартфона; система перевіряє їх; оцінки обробляються сервером; результати доступні через веб-інтерфейс одразу на об’єкті.
2. **Масова обробка даних в офісі**: аналітик завантажує великий обсяг даних через файл; модуль обробки виконує пакетну обробку; підсумковий звіт формується автоматично.
3. **Аварійна ситуація**: при виході з ладу сервера дані зберігаються локально; після відновлення синхронізуються із центральним сховищем.

Таким чином, взаємодія між компонентами системи є **ключовим фактором її надійності, масштабованості та гнучкості**. Запропонована структура взаємодії дозволяє адаптувати систему під різні умови роботи та забезпечує її готовність до майбутніх розширень.

2.5 Формування вимог до даних та схеми їх обробки

Однією з основних задач при проєктуванні веб-системи автоматизованої оцінки дефектів будівельних конструкцій є **визначення вимог до даних** [21, 24]. Це дозволяє забезпечити коректність обробки інформації, мінімізувати ризики помилок, а також гарантувати ефективне функціонування системи.

Формування вимог до даних включає кілька аспектів:

1. Визначення **структури даних**;
2. Встановлення **форматів і обсягів даних**;
3. Визначення **обов'язкових полів і правил валідації**;
4. Вибір **методів обробки даних**;
5. Побудова **логіки взаємозв'язків між даними**.

Вимоги до структури даних

Система повинна оперувати інформацією, яка описує (табл 2.14):

- дефекти конструкцій (пошкодження);
- результати експертної оцінки;
- користувачів та їх ролі.

Таблиця 2.14

Основні інформаційні об'єкти системи

Об'єкт	Основні атрибути
Пошкодження	ID, назва, опис, дата фіксації
Експертна оцінка	ID оцінки, ID пошкодження, ID експерта, значення оцінки, метод, дата
Користувач	ID користувача, ім’я, логін, пароль, роль

Завдяки чітко визначеним атрибутам кожного об'єкта забезпечується **логічна цілісність даних**.

Вимоги до форматів і значень даних

Для забезпечення коректності введення даних необхідно встановити формати для кожного атрибута (табл 2.15).

Таблиця 2.15

Вимоги до форматів даних

Атрибут	**Формат**	**Обмеження**
ID	Ціле число	Автоматична генерація, унікальне
Назва пошкодження	Рядок (до 100 символів)	Обов’язкове поле
Оцінка	Дробове число (0.0–10.0)	Обов’язкове поле
Дата	Формат "РРРР-ММ-ДД"	Не пізніше поточного дня
Роль користувача	Рядок	admin або expert

Такі вимоги запобігають введенню некоректних або неповних даних.

Обов'язкові поля та правила валідації

Для мінімізації помилок система автоматично перевіряє:

- чи заповнені всі обов’язкові поля;
- чи дотримано формату дати;
- чи знаходяться оцінки в допустимому діапазоні;
- чи немає дублікатів ID користувачів та звітів.

У разі виявлення помилки система блокує збереження та виводить повідомлення для користувача.

Схема обробки даних

Обробка даних у системі передбачає кілька рівнів:

1. **Прийом даних** через інтерфейс введення.
2. **Перевірка даних** на коректність і повноту.
3. **Збереження даних** у базі даних.
4. **Обробка даних** через застосування методів оцінки.
5. **Формування звітів**.
6. **Відображення результатів користувачу**.

Кожен етап передбачає обробку даних відповідно до своїх функцій (табл 1.16).

Таблиця 2.16

Етапи обробки даних

Етап	Опис
Прийом	Отримання даних користувача через інтерфейс
Перевірка	Автоматична перевірка правильності даних
Збереження	Запис даних у сховище (БД)
Обробка	Використання методів Дельфі, парних порівнянь, ієрархій
Формування	Побудова звітів
Відображення	Надання результатів користувачу через інтерфейс

Логіка взаємозв’язків між даними

Усі дані структуровані таким чином, щоб забезпечити простоту збереження і ефективний пошук:

- кожен користувач може оцінювати кілька пошкоджень;
- кожне пошкодження може мати багато оцінок;
- обробка здійснюється окремо для кожного методу.

Така структура дозволяє зберігати дані **нормалізовано** та спрощує побудову запитів до бази даних.

Забезпечення якості даних

Система містить вбудовані механізми контролю якості:

1. Автоматична перевірка формату та заповненості полів.
2. Повідомлення про помилки при спробі зберегти некоректні дані.
3. Чіткий розподіл ролей і прав доступу.
4. Логічна структура БД з обов’язковими зовнішніми ключами.
5. Можливість очищення звітів та перезапуску розрахунків.

Ці механізми забезпечують **захищеність, цілісність і актуальність** інформації.

Сценарії обробки даних

Щоб підкреслити практичність підходу, розглянемо приклади:

- Експерт входить у систему, обирає пошкодження, вводить оцінку → система перевіряє й зберігає звіт.
- Адміністратор запускає обробку → система аналізує всі оцінки й виводить таблицю з пріоритетами.
- Користувач з роллю admin видаляє звіти та повторно запускає аналіз із новими даними.

Ці приклади демонструють, як вимоги до даних і логіка обробки застосовуються в реальних умовах.

Переваги чітко сформованих вимог

Таблиця 2.17

Переваги формування вимог до даних

Перевага	**Опис**
Якість	Зменшення помилок користувачів
Надійність	Цілісність даних збережена на всіх етапах
Легкість супроводу	Простіше знаходити і виправляти проблеми
Уніфікованість	Єдині правила для всіх користувачів
Масштабованість	Можливість розширення бази без конфліктів

Таким чином, формування вимог до даних і побудова схеми їх обробки є важливою складовою проєктування системи. Це дозволяє забезпечити **надійність, ефективність і зручність роботи веб-системи**, незалежно від обраної платформи для її реалізації.

2.6 Висновок до другого розділу

У ході другого розділу було виконано комплексне проєктування веб-системи автоматизованої оцінки дефектів будівельних конструкцій, спрямоване на створення архітектури, логічної структури, функціональних компонентів та вимог до даних. Розроблено модель взаємодії між основними модулями системи, що забезпечує ефективну обробку інформації та підтримку користувацьких процесів.

На основі проведеного проєктування сформульовано ключові завдання, які реалізовані у межах другого розділу:

1. **Обґрунтування архітектурних рішень** – вибір трирівневої клієнт-серверної архітектури, яка забезпечує масштабованість, розподіл відповідальності та надійність.
2. **Побудова логічної структури системи** – визначення основних компонентів, їхніх функцій та взаємозв’язків для досягнення узгодженості обробки даних.
3. **Опис функціональних компонентів** – деталізація ролей таких елементів як інтерфейс введення даних, модуль перевірки, обробки оцінок, формування результатів, перегляду інформації та сховища даних.
4. **Розробка взаємодії між компонентами** – опис напрямків обміну даними, застосовуваних протоколів, принципів організації взаємодії та механізмів обробки помилок.
5. **Формування вимог до даних** – визначення структур даних, форматів, правил валідації, логіки обробки та взаємозв’язків між інформаційними об’єктами.

Очікувані результати проєктування

1. **Створення цілісної моделі системи** – отримання повного опису архітектури, логічної структури та функціональних компонентів для подальшої реалізації.
2. **Гнучкість реалізації** – проєкт дозволяє реалізувати систему на будь-яких платформах без зміни логіки роботи.
3. **Масштабованість рішення** – забезпечена можливість розширення системи додатковими модулями чи сервісами без потреби перебудови всієї архітектури.
4. **Забезпечення якості даних** – впроваджені вимоги до даних гарантують їхню цілісність, повноту і правильність.

Переваги розробленої моделі

- **Чіткий поділ обов'язків між компонентами** – кожен модуль відповідає за свою частину процесу.
- **Узгодженість і прозорість обробки даних** – забезпечується контроль на кожному етапі взаємодії.
- **Гнучкість адаптації під нові потреби** – можливість підключення додаткових методів оцінювання чи функціональних блоків.
- **Підвищення ефективності управління даними** – завдяки уніфікованій структурі та продуманій логіці роботи.

Таким чином, розроблені в другому розділі архітектурні, структурні та функціональні рішення закладають **міцну основу для реалізації веб-системи**, яка здатна автоматизувати процес оцінки стану будівельних конструкцій, забезпечуючи високу якість даних, ефективність обробки та гнучкість подальшого розвитку.

РОЗДІЛ 3. РОЗРОБКА ВЕБ-СИСТЕМИ АВТОМАТИЗОВАНОЇ ОЦІНКИ ДЕФЕКТІВ БУДІВЕЛЬНИХ КОНСТРУКЦІЙ

3.1 Обґрунтування вибору інструментів та технологій розробки

Розробка сучасної веб-системи для автоматизованої оцінки дефектів будівельних конструкцій вимагає ретельно підібраного набору інструментів, який забезпечує гнучкість, надійність, масштабованість та зручність у підтримці. У рамках цієї дипломної роботи було реалізовано повноцінну клієнт-серверну систему, побудовану за принципами трирівневої архітектури, де чітко розділено обов’язки між фронтендом (інтерфейс користувача), бекендом (обробка логіки та даних) і базою даних (зберігання інформації).

Обрані інструменти дозволили реалізувати систему, яка не лише задовольняє функціональні вимоги, але й відповідає сучасним вимогам до безпеки, ефективності та зручності взаємодії користувача з інтерфейсом.

Вибір засобів реалізації клієнтської частини (frontend)

Для реалізації інтерфейсу користувача було використано **React** [26] — одну з найпопулярніших JavaScript-бібліотек, яка підтримує компонентний підхід до побудови UI. React дозволяє створювати гнучкі, легко масштабовані інтерфейси, що особливо важливо в контексті розширення системи (наприклад, при додаванні нових модулів оцінки або типів дефектів).

Для підвищення надійності та контролю типів було обрано **TypeScript** [25] — надмножину JavaScript із підтримкою статичної типізації. Завдяки TypeScript помилки в логіці обробки даних виявляються ще на етапі компіляції, що значно знижує ризики збоїв під час роботи системи. У поєднанні з React це дозволило створити стабільну клієнтську частину з чітко визначеною структурою.

Система стилізації реалізована за допомогою **SCSS** [27] у поєднанні з CSS-фреймворком **Bulma**. SCSS дозволяє структуровано організовувати стилі, використовуючи змінні, міксини, вкладеність, що спрощує підтримку та модифікацію. Bulma, у свою чергу, надає базову сітку, адаптивні компоненти та

уніфіковану типографіку, що сприяє швидкому створенню інтуїтивно зрозумілого інтерфейсу.

Вибір засобів реалізації серверної частини (backend)

На рівні серверної логіки було використано середовище виконання **Node.js** [29], у якому реалізовано RESTful API за допомогою фреймворку **Express**. Такий підхід дозволив забезпечити обробку HTTP-запитів, реалізувати авторизацію, маршрути для введення даних, формування звітів і запуск алгоритмів обробки експертних оцінок (наприклад, метод Дельфі, парні порівняння, AHP).

Для централізованого зберігання конфігураційних параметрів, таких як облікові дані бази даних або порт сервера, було використано бібліотеку **dotenv**, яка дозволяє зберігати змінні середовища у файлі .env. Це спрощує перенесення проєкту між середовищами (наприклад, розробка–продакшн).

Завдяки архітектурі з чітким розподілом маршрутів (/api/users, /api/damages, /api/reports, /api/calculation), серверна частина є масштабованою та зручною для модульного розширення. Кожен маршрут реалізує конкретну функціональність і може бути доопрацьований окремо.

Вибір системи управління базами даних

Для реалізації рівня зберігання даних було обрано **PostgreSQL** [30] — об’єктно-реляційну СУБД, яка підтримує складні типи даних, транзакції, індексацію та аналітичні функції. Така база дозволяє ефективно зберігати структури даних: будівлі, дефекти, оцінки, звіти, користувачів, а також масштабувати їх без втрати продуктивності.

Зв’язок між серверною частиною та базою реалізовано за допомогою бібліотеки pg, що дозволяє створювати пул підключень, забезпечуючи ефективну обробку запитів при одночасному доступі кількох користувачів.

Вибір допоміжних інструментів

У якості основного середовища розробки використовувалось **Visual Studio Code**, яке забезпечує підсвічування синтаксису, інтеграцію з Git, автодоповнення, рефакторинг коду та зручний дебагінг (рис 3.1).

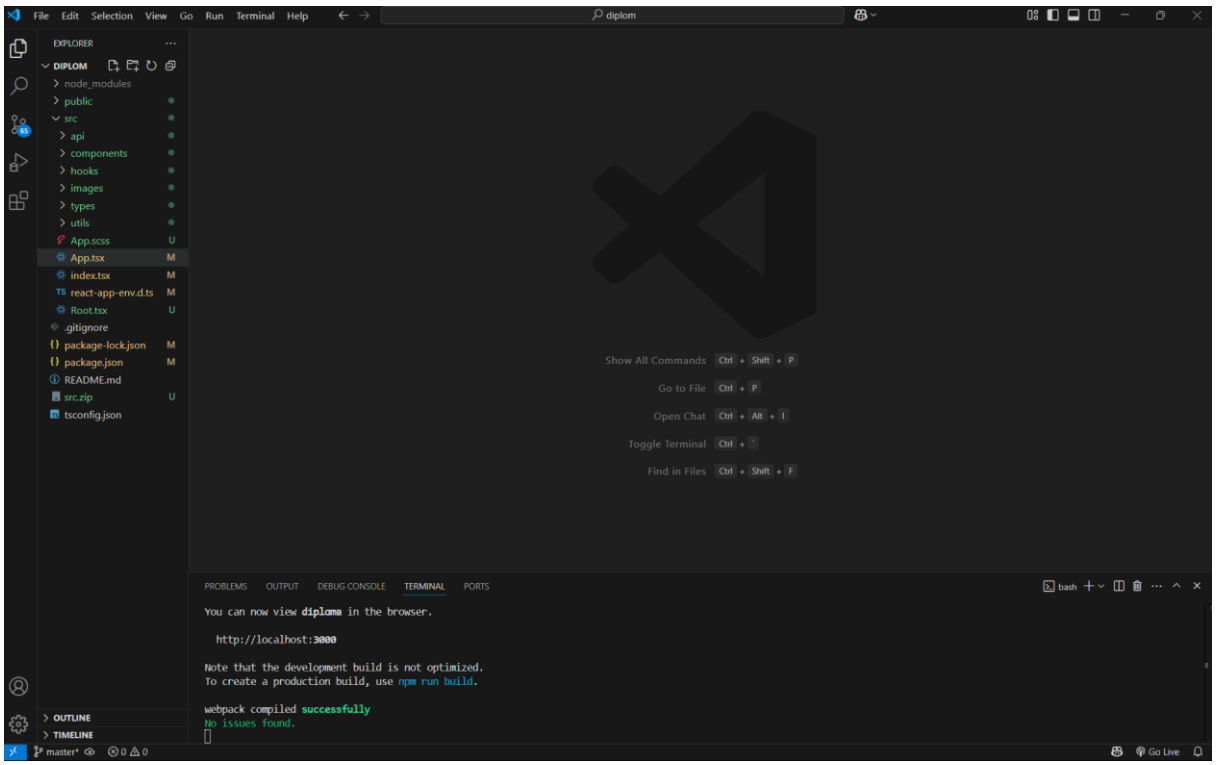


Рисунок 3.1 – Середовище розробки VS Code

Для керування залежностями застосовано **npm**, який дозволяє встановлювати необхідні бібліотеки, оновлювати пакети та зручно організовувати скрипти запуску (наприклад, npm start для сервера або фронтенду).

Узагальнення вибору

Таким чином, поєднання React + TypeScript на фронтенді, Node.js + Express на бекенді та PostgreSQL як СУБД забезпечило реалізацію сучасної, масштабованої та ефективної системи. Кожен інструмент був обраний відповідно до конкретних задач — інтерактивність, обробка експертних оцінок, збереження даних, зручність розробки та супроводу.

3.2 Вибір моделі життєвого циклу розробки програмного забезпечення

Проєктування та розробка будь-якої інформаційної системи потребує визначення підходу до організації життєвого циклу розробки — тобто послідовності дій, які охоплюють весь шлях від формування вимог до впровадження готового продукту. Вибір відповідної моделі має критичне значення, оскільки вона визначає логіку роботи команди, частоту тестування, можливість реагування на зміни та загальний стиль управління проєктом.

Серед найбільш поширених моделей життєвого циклу програмного забезпечення можна виділити каскадну (Waterfall) та спіральну. Каскадна модель передбачає послідовне проходження етапів — від аналізу вимог до тестування і впровадження — без повернення до попередніх фаз. Вона підходить для проєктів зі стабільними, чітко визначеними вимогами, коли всі функціональні та технічні аспекти відомі ще до початку реалізації.

Однак у випадку розробки веб-системи оцінки дефектів будівельних конструкцій на ранньому етапі не всі нюанси були однозначно визначені: алгоритми обробки експертних оцінок потребували уточнення, структура звітності змінювалась у процесі, а також з’являлись нові вимоги до інтерфейсу. У таких умовах каскадна модель була би надто жорсткою — внесення змін на пізніх етапах могло призвести до затримок або повторного перепроєктування.

У зв’язку з цим було прийнято рішення використовувати **спіральну модель**, яка забезпечує ітеративну розробку з можливістю поступового уточнення функціональності. На кожному витку спіралі виконуються типові фази: планування, аналіз ризиків, проєктування, реалізація та тестування. Це дозволяє на ранньому етапі створити прототип, оцінити його працездатність і за потреби скоригувати вимоги (рис. 3.2).

Рисунок 3.2 – Модель життєвого циклу розробки (спіральна модель)

Практична реалізація спіральної моделі у даному проєкті

Розробка веб-системи проходила через чотири основні ітерації, кожна з яких відповідала одному витку спіралі та фокусувалася на конкретній підсистемі:

- **Ітерація 1 – Базова структура системи:** реалізація серверної частини (Node.js + Express), встановлення зв’язку з базою даних PostgreSQL, створення шаблону інтерфейсу користувача (React + TypeScript).
- **Ітерація 2 – Введення та редагування дефектів:** розробка модуля введення пошкоджень, сторінки адміністрування експертів, зв’язок з базою.
- **Ітерація 3 – Модулі експертного оцінювання:** реалізація алгоритмів методу Дельфі, парних порівнянь, AHP; обробка результатів, тестування точності обчислень.
- **Ітерація 4 – Візуалізація звітів і підсумкова оцінка:** побудова інтерфейсу для перегляду результатів, генерація звітів, оптимізація інтерфейсу, виправлення помилок після тестування.

Кожна ітерація включала:

- постановку цілей і формування переліку задач;
- оцінку потенційних ризиків (наприклад, нестача даних, складність реалізації методів);
- реалізацію функціоналу у вигляді модулів;
- створення прототипу і тестування на практичних сценаріях;
- отримання відгуку та підготовку до наступного витка.

Переваги використання спіральної моделі

Вибір спіральної моделі забезпечив такі переваги для даного проєкту:

- **Можливість гнучкого реагування на зміну вимог** — наприклад, адаптація структури звітів під час третьої ітерації.
- **Поступове формування цілісної системи** — кожен виток доповнював та вдосконалював попередній.
- **Раннє виявлення помилок** — тестування реалізовувалося після кожного циклу, що дало змогу вчасно виявити логічні або технічні недоліки.
- **Прозоре планування** — завдяки розбиттю на окремі ітерації вдалося ефективно розподілити час і ресурси.

Цей підхід також дозволив протестувати систему з частковою функціональністю (наприклад, лише модуль парних порівнянь) ще до повного завершення проєкту, що сприяло зменшенню ризиків і пришвидшило інтеграцію.

Таким чином, спіральна модель життєвого циклу виявилась найбільш придатною для даного проєкту, оскільки поєднує послідовність з можливістю адаптації, дозволяє уникати критичних помилок на пізніх етапах розробки, а також сприяє поступовому нарощенню функціональності системи. Її використання забезпечило баланс між структурованістю процесу та гнучкістю реалізації, що відповідає реаліям розробки веб-додатків середнього рівня складності з нестабільними вимогами.

3.3 Архітектура програмної системи

У рамках розробки веб-системи було реалізовано архітектурну модель, що забезпечує чіткий поділ функціональності та взаємодію між компонентами. В цьому підрозділі представлено загальний підхід до організації структури системи без надмірної деталізації її внутрішніх маршрутів або рівнів.

Загальна архітектура включає три основні компоненти:

- клієнтську частину (інтерфейс користувача);
- серверну частину (обробка запитів і логіка);
- базу даних (збереження інформації).

Важливою особливістю цієї архітектури є її відповідність моделі розділення обов'язків (Separation of Concerns), що дозволяє ізолювати функціональність між шарами. Такий підхід забезпечує:

- полегшене тестування окремих частин системи;
- незалежну модифікацію підсистем без порушення загальної логіки;
- можливість паралельної роботи над проєктом кількох команд розробників.

Для збереження стабільності взаємодії між шарами застосовується REST API. Клієнт надсилає запити до серверної частини, яка, своєю чергою, звертається до бази даних. Усі рівні функціонують автономно і взаємодіють між собою лише через визначені інтерфейси, що сприяє масштабованості та надійності системи (рис. 3.3).

Рисунок 3.3 – Логічна схема взаємодії

Завдяки обраній архітектурі розробник може легко впроваджувати нові функції (наприклад, нові методи оцінювання), не змінюючи структуру інтерфейсу або схеми бази даних. Це забезпечує не лише стабільність функціонування, а й можливість безперервного вдосконалення системи.

Таким чином, реалізована архітектура дає змогу дотримуватись ключових принципів надійної розробки ПЗ: модульність, слабке зв’язування компонентів, незалежність рівнів, розширюваність і стабільність.

3.4 Логічна структура та модульна побудова системи

Ефективність програмного забезпечення значною мірою залежить від правильно організованої логічної структури — тобто того, як система розбивається на функціональні частини (модулі) та яким чином ці частини взаємодіють між собою. Такий підхід дозволяє не лише підвищити надійність і керованість проєкту, але й забезпечити легкість у супроводі, модифікації та розширенні функціоналу.

У реалізованій системі модульна структура була сформована відповідно до архітектурної концепції трьох рівнів: інтерфейс користувача, логіка обробки, сховище даних. Кожен рівень реалізовано як сукупність підсистем, які відповідають за конкретні функції та реалізовані як окремі логічні блоки.

Функціональні блоки системи (рис. 3.4):

- **Блок авторизації** — відповідає за створення облікових записів, перевірку даних входу, збереження ролей та контроль доступу до модулів.
- **Блок введення оцінок** — реалізує введення експертами числових оцінок пошкоджень із подальшим збереженням у БД.
- **Блок обробки оцінок** — включає алгоритми обчислення: парні порівняння, метод Дельфі, метод зважених коефіцієнтів. Обробка виконується сервером із доступом до всіх звітів.
- **Блок перегляду результатів** — надає візуальне представлення результатів аналізу: таблиці, графіки, списки пріоритетів.
- **Блок управління об'єктами** — дозволяє адміністратору редагувати записи дефектів, будівель, облікові записи користувачів.

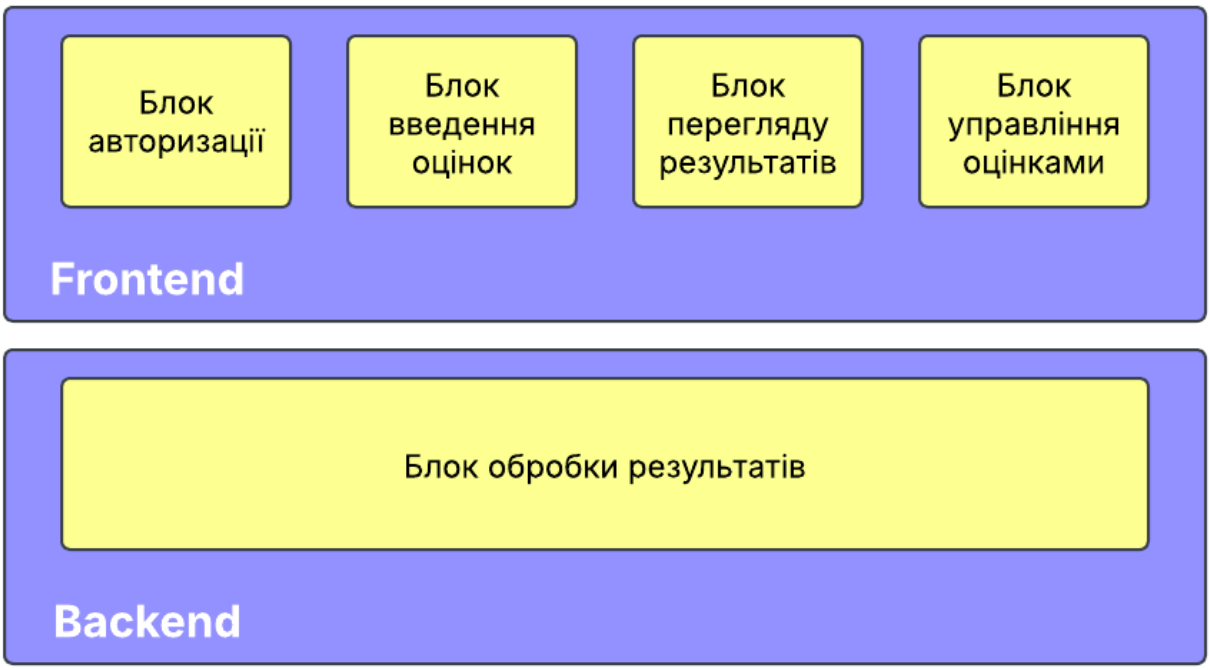


Рисунок 3.4 – Компонентна діаграма модулів за рівнями системи

Кожен з логічних блоків виконує окрему роль, однак всі вони взаємодіють через уніфіковану API-структуру. Такий підхід дає змогу масштабувати систему — наприклад, розширити набір методів обробки або додати нову роль користувача — без потреби переписувати існуючі модулі.

Додатковою перевагою такої побудови є можливість автономного тестування та налагодження кожного модуля. Це особливо важливо при розгортанні системи в умовах багатокористувацького середовища, де надійність роботи кожного блоку має вирішальне значення.

Загалом, реалізація логічної структури системи у вигляді набору незалежних модулів дозволила забезпечити гнучкість розробки, підтримку зрозумілої архітектури та простоту розширення в майбутньому.

3.5 Функціональні переходи та алгоритми обробки оцінок

Розробка інтерфейсної та логічної частини веб-системи вимагає чіткого визначення послідовності дій користувачів, а також механізмів обробки введених ними даних. У цьому підрозділі представлено:

- схему переходів між основними функціями системи;
- алгоритми, які забезпечують обчислення результатів оцінювання пошкоджень.

Сценарії використання

У системі передбачено дві основні ролі: **експерт** та **адміністратор**. Кожна з них має власну послідовність взаємодії з інтерфейсом (рис. 3.5).

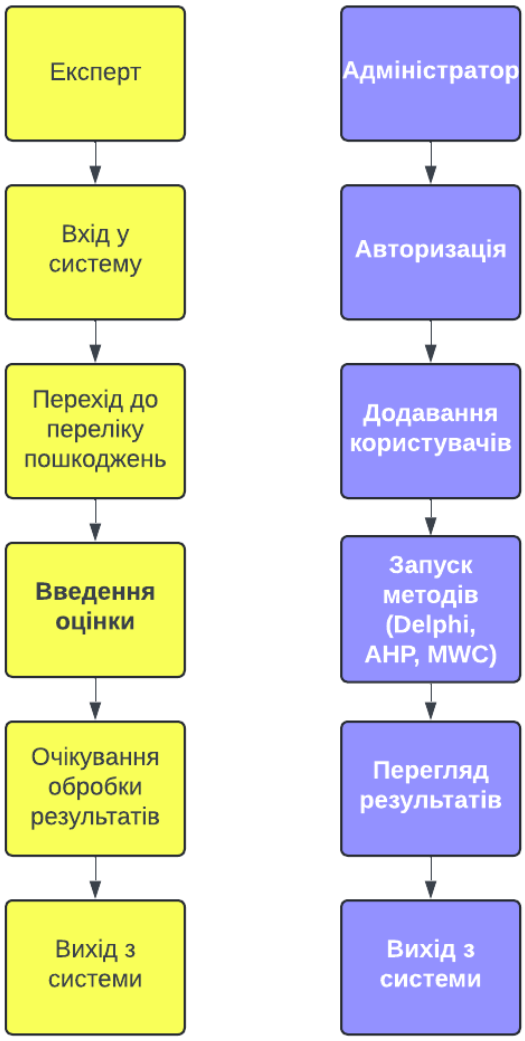


Рисунок 3.5 – Схема переходів між функціями в системі

Алгоритми обробки експертних оцінок

Після збору експертних оцінок система проводить обробку результатів із використанням **трьох основних методів** багатокритеріального прийняття рішень:

- **Метод Delphi**
- **Метод аналізу ієрархій (AHP)**
- **Метод зважених коефіцієнтів (MWC)**

Ці методи реалізуються у бекенд-частині системи, і кожен з них має власну логіку обчислень, адаптовану до структури введених даних.

Метод Delphi (рис. 3.6)

Метод Delphi реалізовано як процес колективного оцінювання, де кожен експерт незалежно вводить свої оцінки, після чого система:

1. Збирає всі оцінки з таблиці reports.
2. Групує їх за damage_id.
3. Розраховує:
    - **середнє значення** оцінок (average);
    - **стандартне відхилення** оцінок (std_deviation) для оцінки узгодженості думок.
4. Встановлює **пріоритетність** пошкодження на основі середнього значення:
    - високий, якщо ≥ 7;
    - середній, якщо від 4 до 7;
    - низький, якщо < 4.

Алгоритм простий, швидкий, та підходить для первинної аналітики. Додатково std_deviation може використовуватись для виявлення контроверсійних випадків.

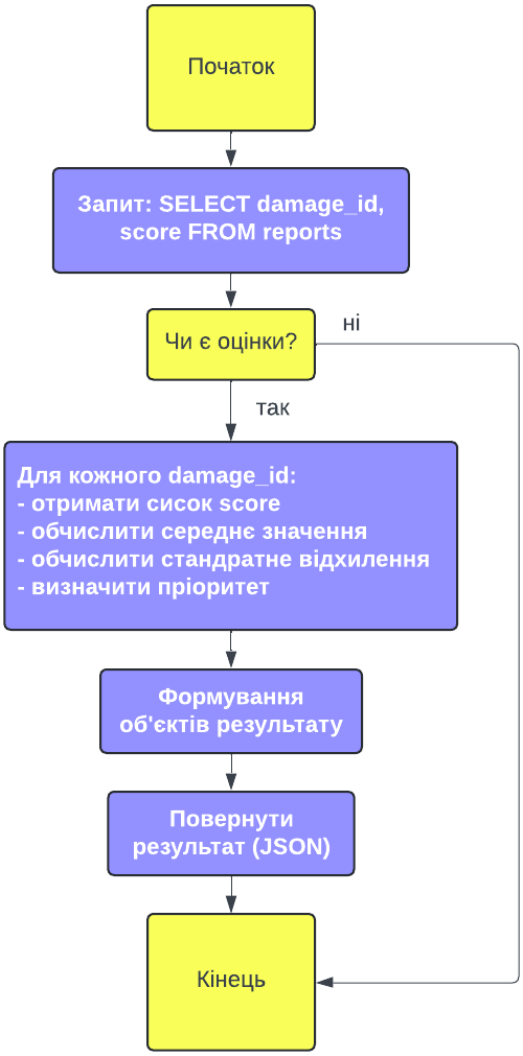


Рисунок 3.6 – Схема реалізації методу Delphi

Метод аналізу ієрархій (AHP, парні порівняння) (рис. 3.7)

Алгоритм реалізовано через автоматичне формування **матриці парних відношень** на основі середніх оцінок по кожному пошкодженню.

Етапи:

1. Розраховується середнє значення оцінок для кожного damage_id.
2. Створюється **матриця AHP**.
3. Кожен стовпець нормалізується: ділиться на суму по стовпцю.
4. Для кожного рядка обчислюється середнє значення — це і є **вагова оцінка** (пріоритет).
5. Результат доповнюється іменами пошкоджень з таблиці damages.

Перевагою є те, що метод автоматично формує відношення навіть без введення парних оцінок вручну, що пришвидшує обчислення.

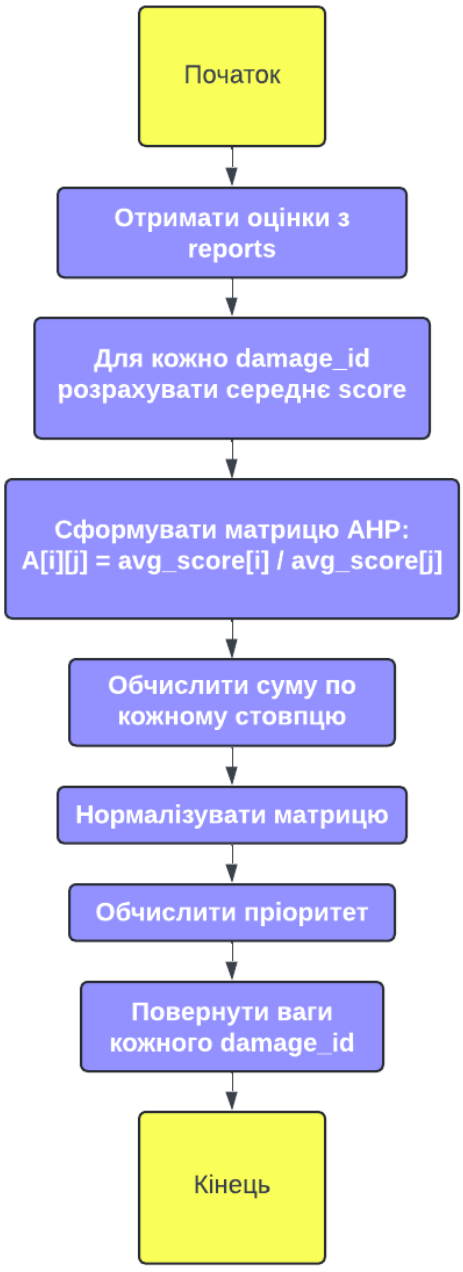


Рисунок 3.7 – Схема реалізації методу AHP

Метод зважених коефіцієнтів (MWC) (рис. 3.8)

Реалізація методу MWC враховує **індивідуальну компетентність кожного експерта**, яка розраховується автоматично:

1. Обчислюється середнє відхилення кожного експерта від загального середнього по кожному пошкодженню.
2. Експерти сортуються за точністю, і найменше відхилення = найбільша компетентність.
3. Компетентність використовується як **вага** в обчисленнях.
4. На основі отриманої зваженої оцінки визначається пріоритет: високий / середній / низький.

Це найгнучкіший метод у системі, який дозволяє динамічно адаптувати вплив кожного експерта на кінцевий результат.

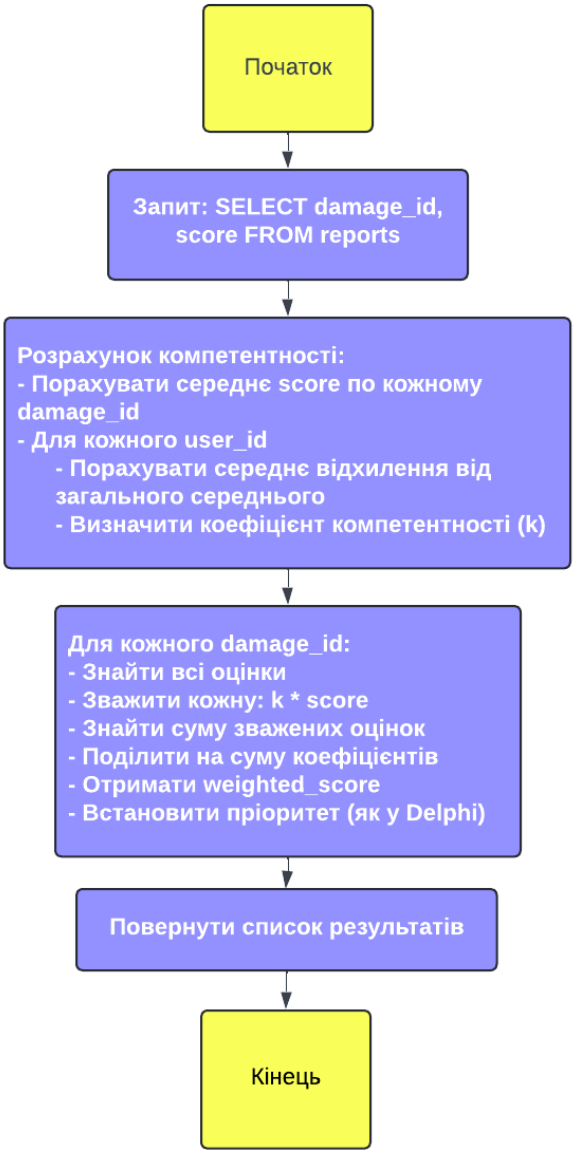


Рисунок 3.8 – Схема реалізації методу зважених коефіцієнтів (MWC)

3.6 Реалізація клієнтської частини (frontend)

Клієнтська частина системи реалізована на основі сучасного стеку веб-технологій — **React**, **TypeScript**, **SCSS**, що забезпечує не лише високу продуктивність інтерфейсу, а й зручність у розширенні функціоналу. Компонентний підхід дозволив створити гнучку та масштабовану структуру, де кожен елемент інтерфейсу функціонує як незалежний модуль із власною логікою, стилями та інтерфейсом обміну даними.

Використані технології

- **React** — бібліотека для побудови користувацьких інтерфейсів із віртуальним DOM, яка забезпечує швидке оновлення окремих частин сторінки без перезавантаження.
- **TypeScript** — мова з статичною типізацією, яка мінімізує помилки, покращує автодоповнення та контроль типів під час розробки.
- **SCSS + Bulma** — для стилізації інтерфейсу використано SCSS (розширення CSS із підтримкою вкладеності, змінних, міксинів) та Bulma (готові UI-компоненти: кнопки, сітка, таблиці тощо).
- **React Router v6** — для організації маршрутизації в межах SPA (Single Page Application).

Структура клієнтського застосунку

Фронтенд реалізовано з урахуванням принципів розділення відповідальності та повторного використання коду. Структура директорій є такою (рис. 3.9):

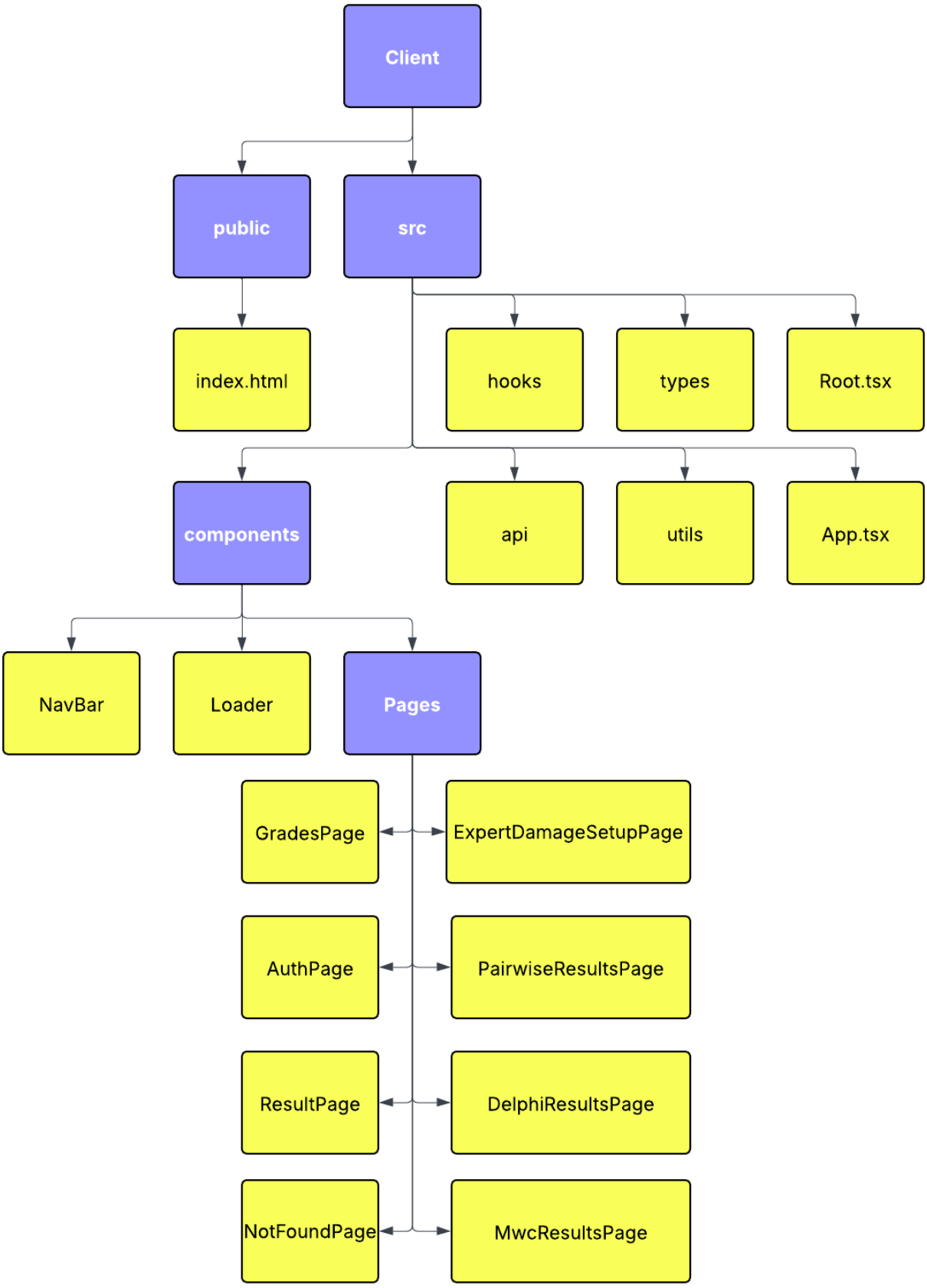


Рисунок 3.9 – Структура файлів клієнтської частини

Компоненти інтерфейсу

Фронтенд реалізовано як набір **перевикористовуваних компонентів (табл. 3.1)**, які дотримуються принципів розділення обов'язків (Separation of Concerns):

Таблиця 3.1

Призначення компонентів інтерфейсу

Компонент	**Призначення**
Navbar	Відображає верхнє меню, адаптується до ролі користувача
AuthPage	Форма авторизації з перевіркою логіна/пароля
ExpertDamageSetupPage	Сторінка адміністрування дефектів та експертів
GradesPage	Форма введення оцінок експертом за заданими критеріями
ResultPage	Виведення узагальнених результатів
DelphiResultsPage, PairwiseResultsPage, MWCResultsPage	Сторінки для виводу результатів за методами

Усі сторінки вкладені в головний макет App.tsx, який відповідає за структуру сторінки (меню + контент) (рис. 3.10).

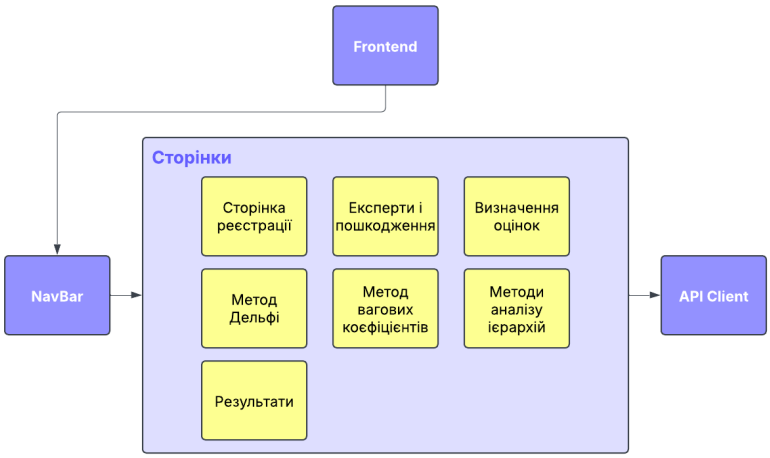


Рисунок 3.10 – Схема Frontend

Маршрутизація

У файлі Root.tsx реалізовано повну маршрутизацію системи (рис. 3.12):

```tsx
export const Root = () => (
  <Router>
    <Routes>
      <Route path="/" element={<App />}>
        {/* Головна сторінка → авторизація */}
        <Route index element={<AuthPage />} />

        {/* Перенаправлення /auth → / */}
        <Route path="auth" element={<Navigate replace to="/" />} />

        {/* Сторінки */}
        <Route path="expertsAndDamages" element={<ExpertDamageSetupPage />} />
        <Route path="grades" element={<GradesPage />} />
        <Route path="pairwise" element={<PairwiseResultsPage />} />
        <Route path="delphi" element={<DelphiResultsPage />} />
        <Route path="mwc" element={<MWCResultsPage />} />
        <Route path="results" element={<ResultPage />} />

        {/* 404 */}
        <Route path="*" element={<NotFoundPage />} />
      </Route>
    </Routes>
  </Router>
);
```

Рисунок 3.12 – Реалізація маршрутизації сторінок у Root.tsx

Це забезпечує зручну навігацію без перезавантаження сторінок (SPA-підхід), що підвищує швидкодію й зменшує навантаження на сервер.

Робота з API

Для комунікації з бекендом використовуються **Axios-запити**, винесені в окремі сервіси. Наприклад (рис. 3.13):

```tsx
// Отримати всіх користувачів
export const getExperts = async () => {
  const response = await apiClient.get('/users');
  return response.data;
};

// Створити користувача (експерта або адміністратора)
export const createExpert = async (expert: CreateExpertDto) => {
  const response = await apiClient.post('/users/register', expert);
  return response.data;
};
```

Рисунок 3.13 – Приклад запитів

Особливості реалізації

- **Локальне збереження ролі користувача** реалізовано через localStorage, що дозволяє рендерити відповідний інтерфейс залежно від статусу.
- **Інтерактивність форм** (введення оцінок, фільтри, кнопки експорту) реалізовано з використанням useState, useEffect, useForm.
- **Модульність стилів** — кожна сторінка має окремий SCSS-файл із просторовою ізоляцією стилів.
- **Інтуїтивний інтерфейс** — завдяки використанню Bulma інтерфейс є легким, адаптивним, доступним для сприйняття (рис. 3.14).

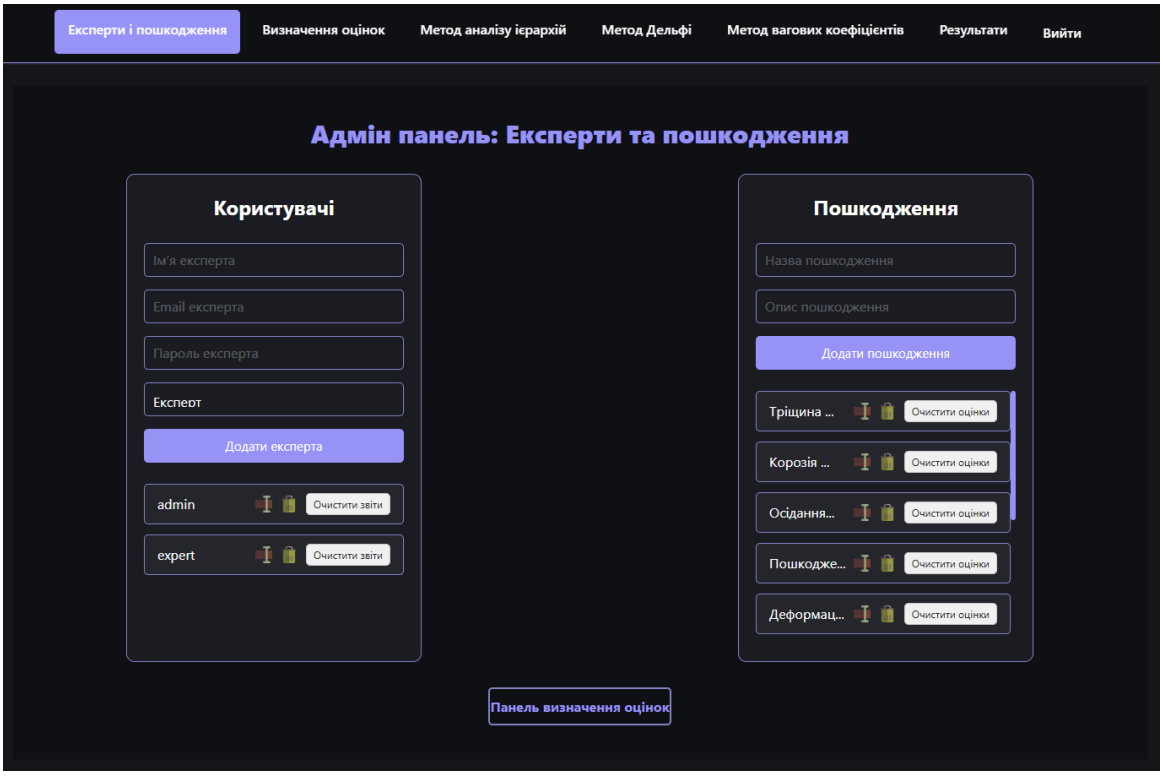


Рисунок 3.14 – Інтерфейс адміністратора

Клієнтська частина системи реалізована відповідно до сучасних підходів у веб-розробці. Вона є структурованою, типізованою, масштабованою, підтримує розширення функціоналу та дозволяє користувачам ефективно взаємодіяти із системою. Компонентний підхід, маршрутизація та відокремлення логіки API забезпечують простоту тестування, адаптивність до нових вимог і зручність розробки.

3.7 Реалізація серверної частини (backend)

Серверна частина веб-системи реалізована на платформі **Node.js** із використанням фреймворку **Express**, що дозволяє організувати обробку HTTP-запитів у вигляді RESTful API. Такий підхід забезпечує масштабованість, легкість інтеграції з клієнтською частиною та модульність розробки.

Загальна структура backend-додатку

Проєкт серверної частини має наступну структуру (рис. 3.15):

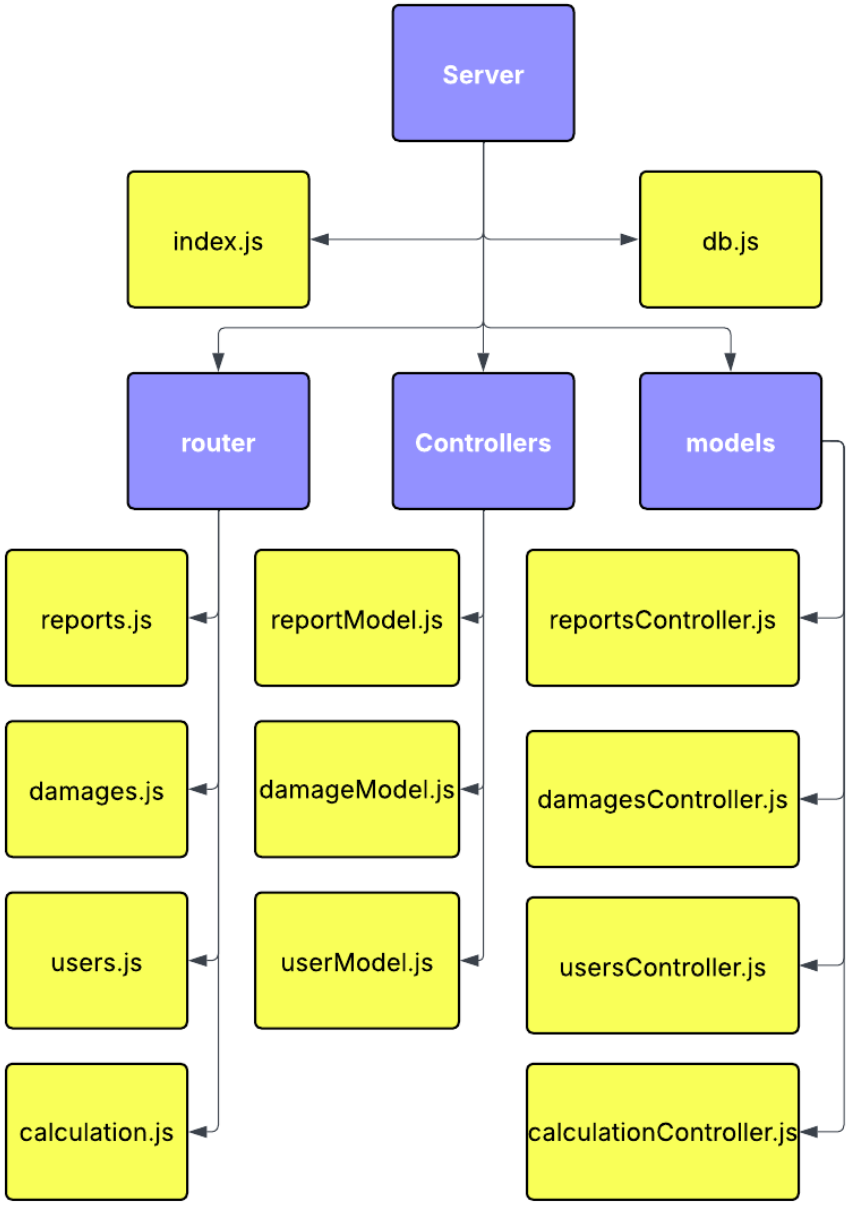


Рисунок 3.15 – Структура серверної частини веб-системи

index.js — запуск сервера

На рисунку нижче зображено головний файл серверної частини — index.js, який виконує ініціалізацію Express-додатку, підключає необхідні модулі та маршрути, а також налаштовує обробку запитів від клієнтської частини.

Цей файл виконує ініціалізацію серверу, підключає всі маршрути та дозволяє клієнтам надсилати запити з інших доменів (через CORS). Він також формує базову структуру API з чітким поділом відповідальностей за кожну з груп запитів: користувачі, пошкодження, звіти та обчислення (рис. 3.16).

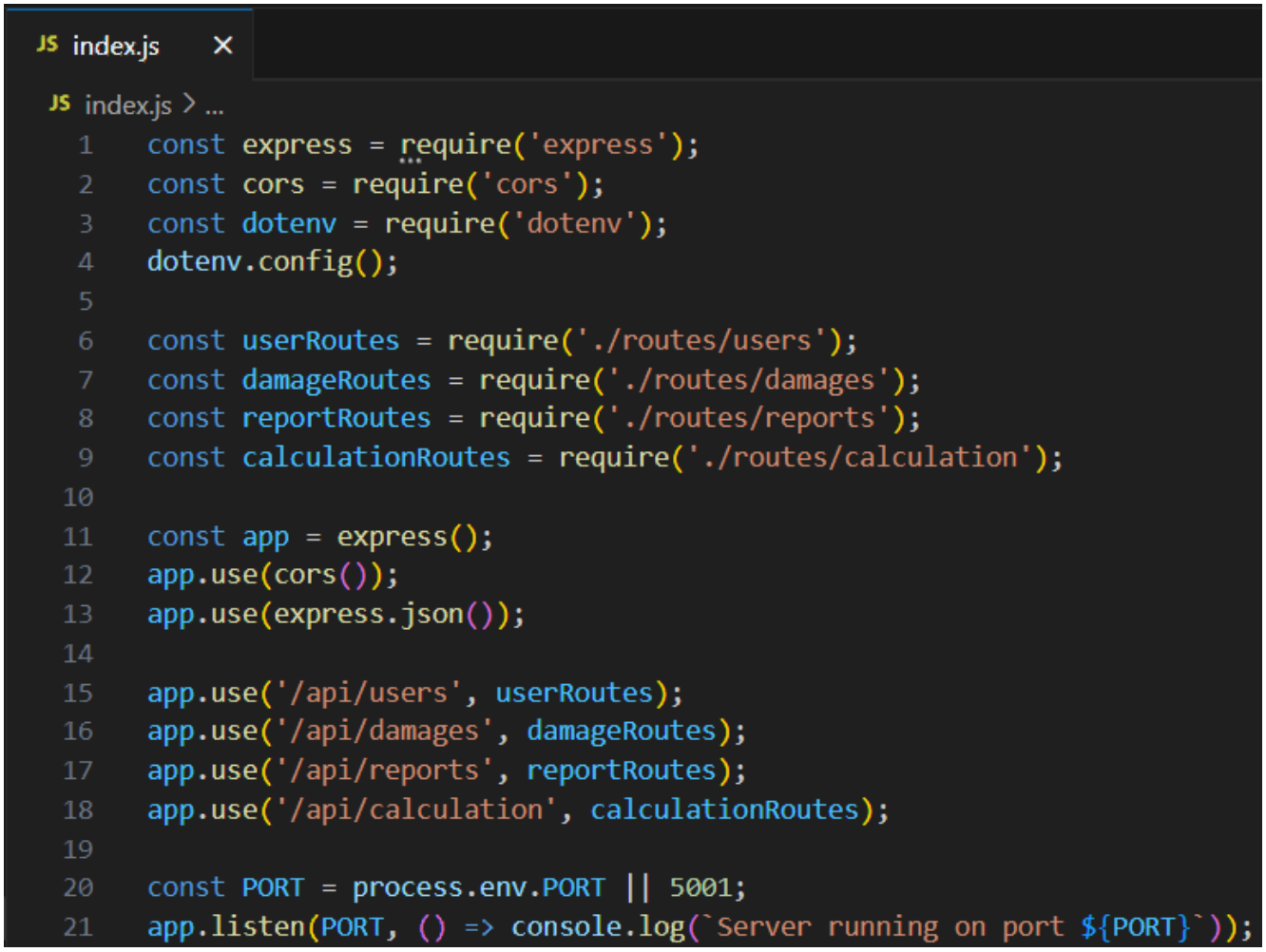


Рисунок 3.16 – Ініціалізація серверної частини у файлі index.js

З’єднання з базою даних

Ще одним ключовим елементом є файл db.js, що відповідає за підключення до СУБД PostgreSQL. У ньому створено **пул з’єднань**, який дозволяє обробляти кілька запитів одночасно без перевантаження сервера.

Параметри з’єднання передаються з .env файлу, що дозволяє змінювати конфігурацію без потреби редагувати код програми (рис. 3.17).

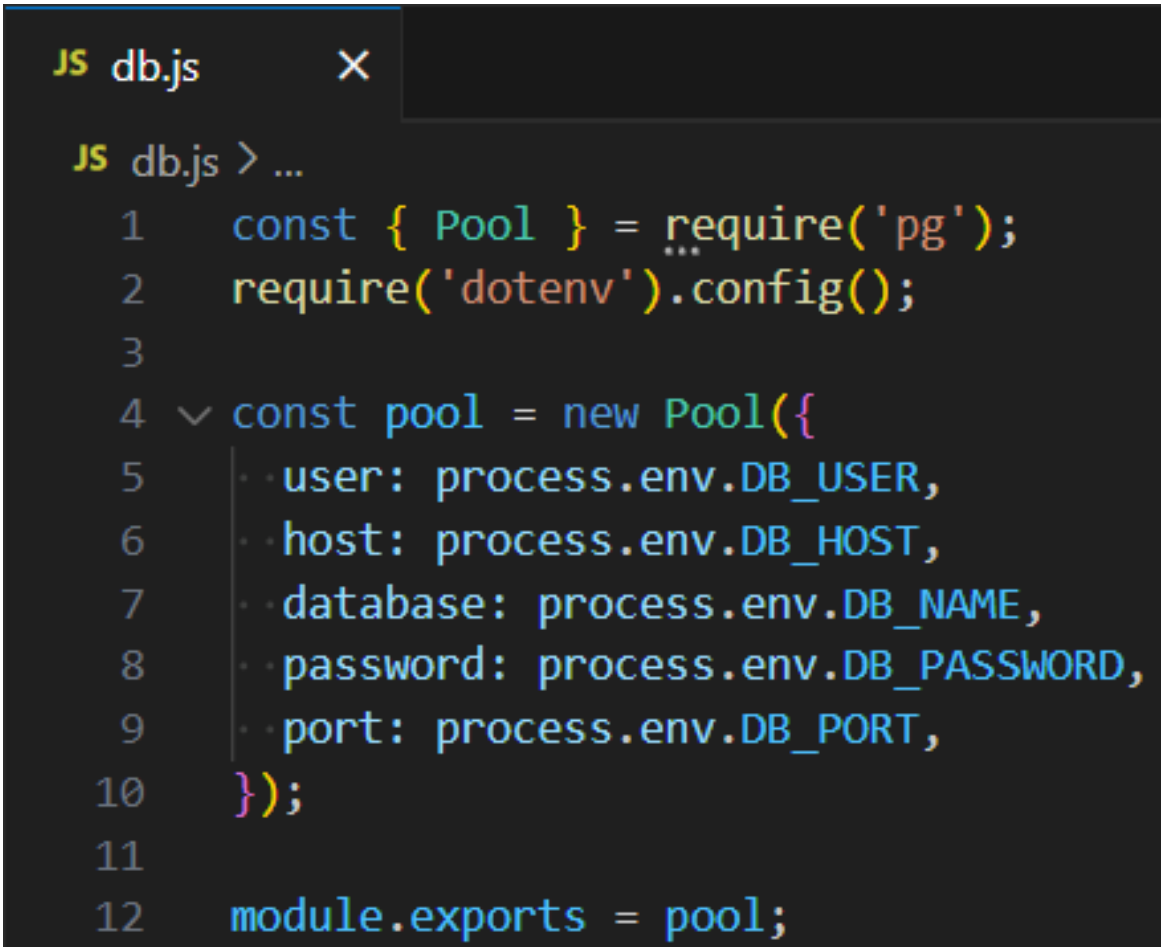


Рисунок 3.17 – Конфігурація підключення до PostgreSQL у файлі db.js

Підключення до бази інкапсульовано, що дозволяє використовувати pool.query(...) у будь-якому маршруті.

Маршрути API системи

На рисунку (рис. 3.18) представлено структуру REST API, яка побудована на основі принципів розділення відповідальності. Усі запити згруповані за HTTP-методами логічними модулями: users, damages, reports, calculation (рис. 1.13).

/api/users

- GET /api/users – отримання списку користувачів
- POST /api/users/register – створення нового облікового запису
- POST /api/users/login – авторизація
- PUT /api/users/:id – редагування користувача
- DELETE /api/users/:id – видалення користувача
- DELETE /api/users/:id/clear-reports – очищення оцінок експерта

/api/damages

- GET /api/damages – перелік пошкоджень
- POST /api/damages – додавання дефекту
- PUT /api/damages/:id – редагування
- DELETE /api/damages/:id – видалення
- DELETE /api/damages/:id/clear – очищення оцінок

/api/reports

- GET /api/reports – отримання всіх звітів
- POST /api/reports – збереження оцінки
- DELETE /api/reports/:id – видалення окремого звіту

/api/calculation

- GET /api/calculation/delphi – метод Дельфі
- GET /api/calculation/pairwise – парні порівняння
- GET /api/calculation/calculateMWC – метод зважених коефіцієнтів

Рисунок 3.18 – Повна структура REST API системи

Організація логіки та безпека

- Дані перед збереженням проходять валідацію: перевіряється формат, тип, обов’язковість полів.
- Для збереження безпеки підключення облікові дані до бази не зберігаються у коді — вони винесені в .env.
- Ролі користувачів (експерт, адміністратор) зберігаються на клієнті, але можуть перевірятися на сервері при потребі.

Обробка алгоритмів експертного оцінювання

Окрему увагу займає модуль calculation.js, який виконує:

- **Агрегацію оцінок** з таблиці reports;
- Обчислення вагових коефіцієнтів (метод парних порівнянь);
- Генерацію кінцевого результату (Delphi тощо);
- Формування ранжованого списку пріоритетів.

Алгоритми реалізовані у вигляді окремих функцій, що приймають масиви з оцінками та повертають результати у форматі JSON.

Особливості реалізації

- **Модульність** — маршрути розбиті за тематиками, що дозволяє легко орієнтуватися в коді.
- **Масштабованість** — у разі необхідності можна винести окремі модулі (наприклад, auth) у мікросервіси.
- **Незалежність логіки від UI** — зміна бекенду не потребує модифікації інтерфейсу, доки API незмінне.
- **Продуктивність** — завдяки пулу з’єднань обробка запитів відбувається швидко навіть при навантаженні.

Серверна частина системи реалізована відповідно до принципів REST-архітектури та забезпечує ефективне обслуговування запитів клієнтів, обробку логіки експертного оцінювання та безпечну взаємодію з базою даних.

3.8 База даних і сховище інформації

Для забезпечення збереження, обробки та доступу до даних, що використовуються у процесі експертного оцінювання дефектів будівельних конструкцій, у системі реалізовано сховище на основі **реляційної бази даних PostgreSQL**. Ця СУБД була обрана завдяки її стабільності, підтримці складних типів даних, транзакцій, надійній роботі з великим обсягом інформації та сумісності з інструментами Node.js.

Основні таблиці бази даних

1. Містить містить облікові записи користувачів системи. (табл. 3.2)

Таблиця 3.2

Таблиця users

Поле	**Тип**	**Опис**
id	SERIAL	Унікальний ідентифікатор
name	TEXT	Ім’я користувача
email	TEXT	Логін (адреса пошти)
password	TEXT	Хеш пароля
role	TEXT	Роль користувача (admin, expert)

2. Описує дефекти, виявлені у будівельних конструкціях (табл. 3.3).

Таблиця 3.3

Таблиця damages

Поле	**Тип**	**Опис**
id	SERIAL	Ідентифікатор дефекту
title	TEXT	Назва або опис пошкодження
severity	INTEGER	Попередній рівень важкості (не обов’язково)

3. Зберігає оцінки, які надають експерти щодо пошкоджень (табл. 3.4).

Таблиця 3.4

Таблиця reports

Поле	**Тип**	**Опис**
id	SERIAL	Ідентифікатор звіту
user_id	INTEGER	Посилання на експерта (users.id)
damage_id	INTEGER	Посилання на пошкодження (damages.id)
grade	FLOAT	Оцінка, виставлена експертом
method	TEXT	Метод оцінювання (delphi, pairwise, тощо)
created_at	TIMESTAMP	Час створення звіту

Крім текстового опису структури таблиць, для наочності на рисунку нижче представлено **ER-діаграму бази даних**, яка ілюструє взаємозв’язки між основними сутностями системи (рис. 3.19):

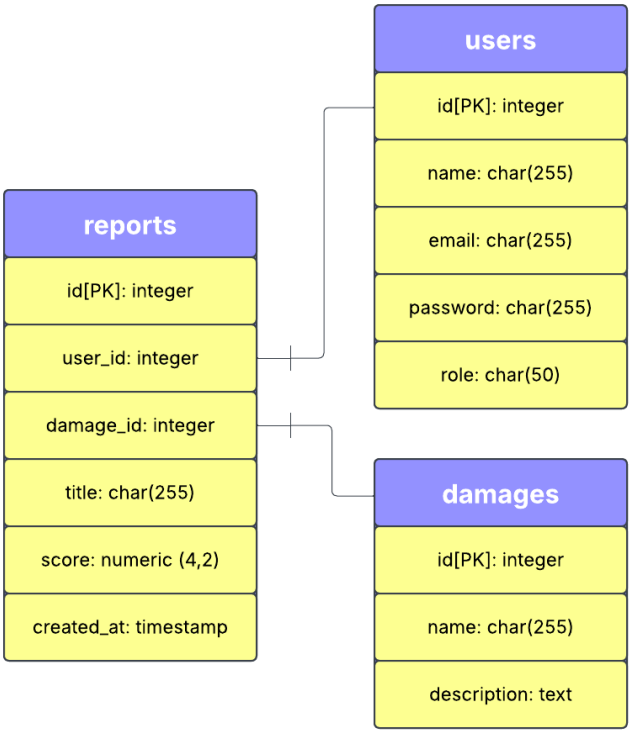


Рисунок 3.19 – ER-діаграма бази даних системи

Взаємозв'язки між таблицями

Всі таблиці пов'язані між собою за допомогою зовнішніх ключів:

- reports.user_id → users.id
- reports.damage_id → damages.id

Це дозволяє:

- Витягувати всі оцінки, залишені певним експертом;
- Аналізувати середні значення оцінок по об'єкту;
- Створювати агреговані звіти за критеріями та методами.

Підключення та робота з базою

Підключення до бази відбувається через pg.Pool (пул з'єднань), що дозволяє обробляти кілька запитів одночасно без перевантаження сервера.

Запити реалізуються у вигляді SQL-виразів у кожному маршруті (рис. 3.20):

```
1   const pool = require('../db');
2
3   // Отримати всіх користувачів
4   const getAllUsers = () => {
5     return pool.query('SELECT id, name, email, role FROM users');
6   };
7
8   // Знайти користувача за ім'ям (для логіну)
9   const findUserByName = (name) => {
10    return pool.query('SELECT * FROM users WHERE name = $1', [name]);
11  };
```

Рисунок 3.20 – Приклад запитів до сервера

Формат запитів забезпечує:

- Захист від SQL-ін'єкцій (через параметризовані запити);
- Високу продуктивність при роботі з великим обсягом записів;
- Можливість розширення (JOIN, GROUP BY, агрегатні функції).

Особливості структури бази

- **Нормалізація:** База приведена до 2NF, що дозволяє уникнути дублювання даних.
- **Гнучкість:** структура підтримує додавання нових методів оцінювання без зміни схеми.
- **Масштабованість:** у майбутньому можна створити окремі таблиці для категорій пошкоджень, типів будівель, результатів обробки, логів тощо.

Реалізована структура бази даних забезпечує ефективне зберігання і обробку інформації, пов’язаної з експертним оцінюванням будівельних дефектів. Завдяки нормалізованим таблицям, чітким зв’язкам між сутностями та можливості гнучкої фільтрації даних система може масштабуватись, адаптуватись до нових методів обробки та зберігати цілісність інформації навіть у багатокористувацькому середовищі.

3.9 Тестування системи

У цьому підрозділі виконано покрокове тестування роботи веб-системи з боку двох основних ролей: **експерта** та **адміністратора**. Кожен сценарій підтверджено скріншотами виконаних дій, результатами обробки даних та виводами інтерфейсу.

Сценарій 1. Дії користувача з роллю “Експерт”

1.1 Реєстрація як експерт (рис. 3.21)

- Заповнення форми реєстрації;
- Відправка даних на сервер та підтвердження створення облікового запису.

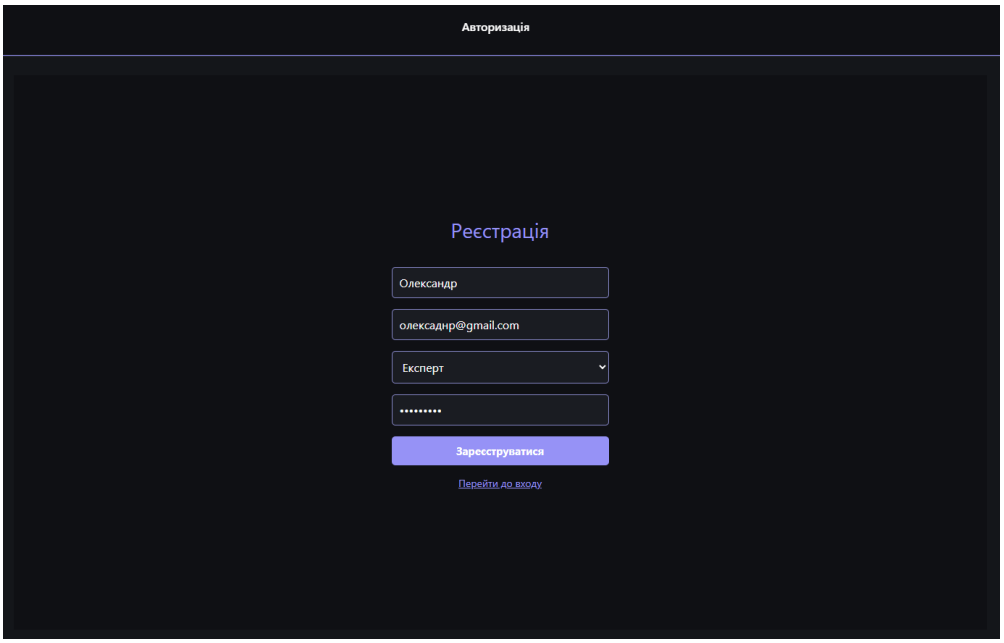


Рисунок 3.21 – Форма реєстрації користувача з роллю “Експерт”

1.2 Вхід до системи як експерт (рис. 3.22)

- Авторизація з раніше зареєстрованими даними;
- Перехід до інтерфейсу експерта.

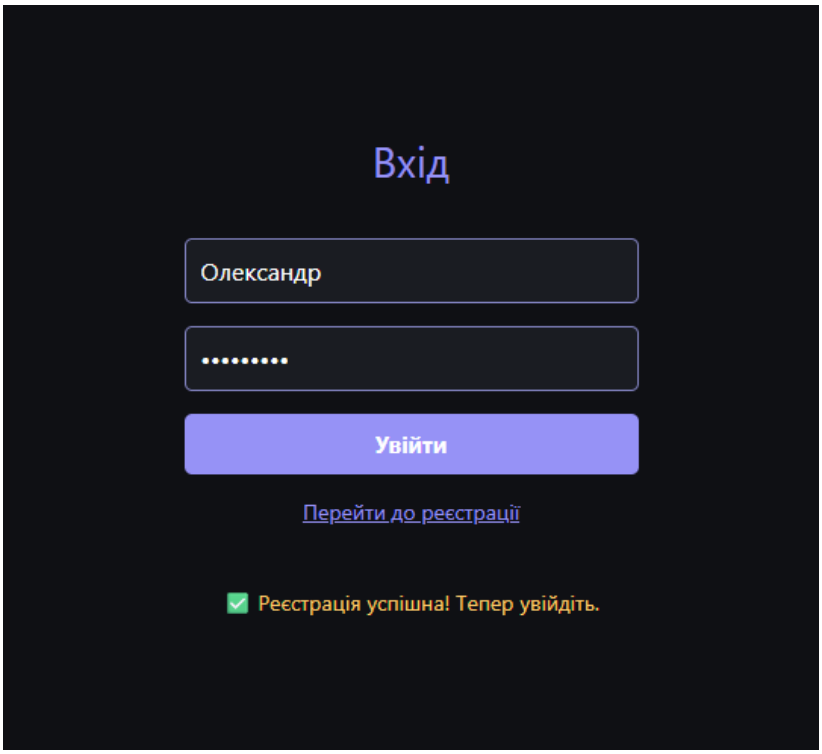


Рисунок 3.22 – Успішний вхід експерта в систему

1.3 Додавання пошкодження (рис. 3.23)

- Вибір функції додавання дефекту;
- Заповнення назви, опису.

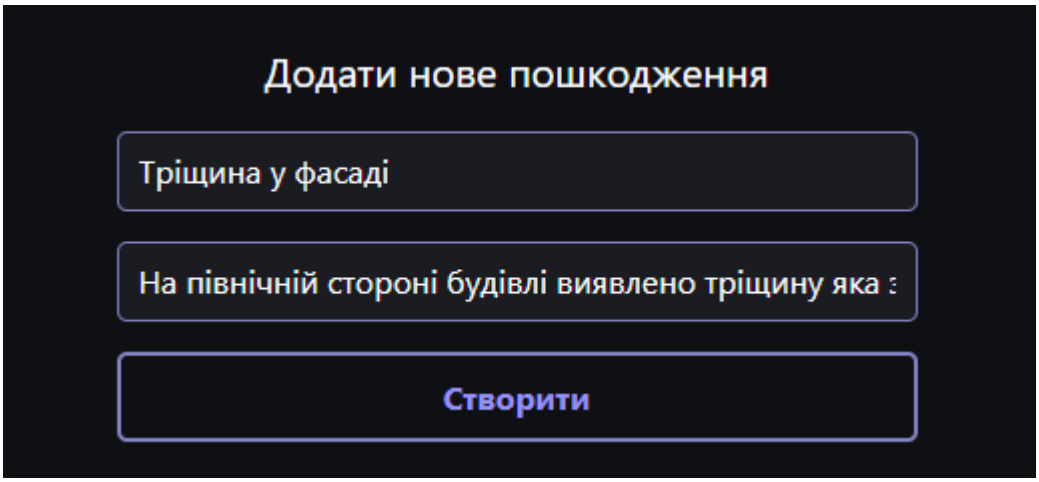


Рисунок 3.23 – Інтерфейс додавання нового пошкодження

1.4 Виставлення оцінок (створення звітів) (рис. 3.24)

- Перегляд переліку пошкоджень;
- Введення оцінки за заданою шкалою;
- Надсилання форми.

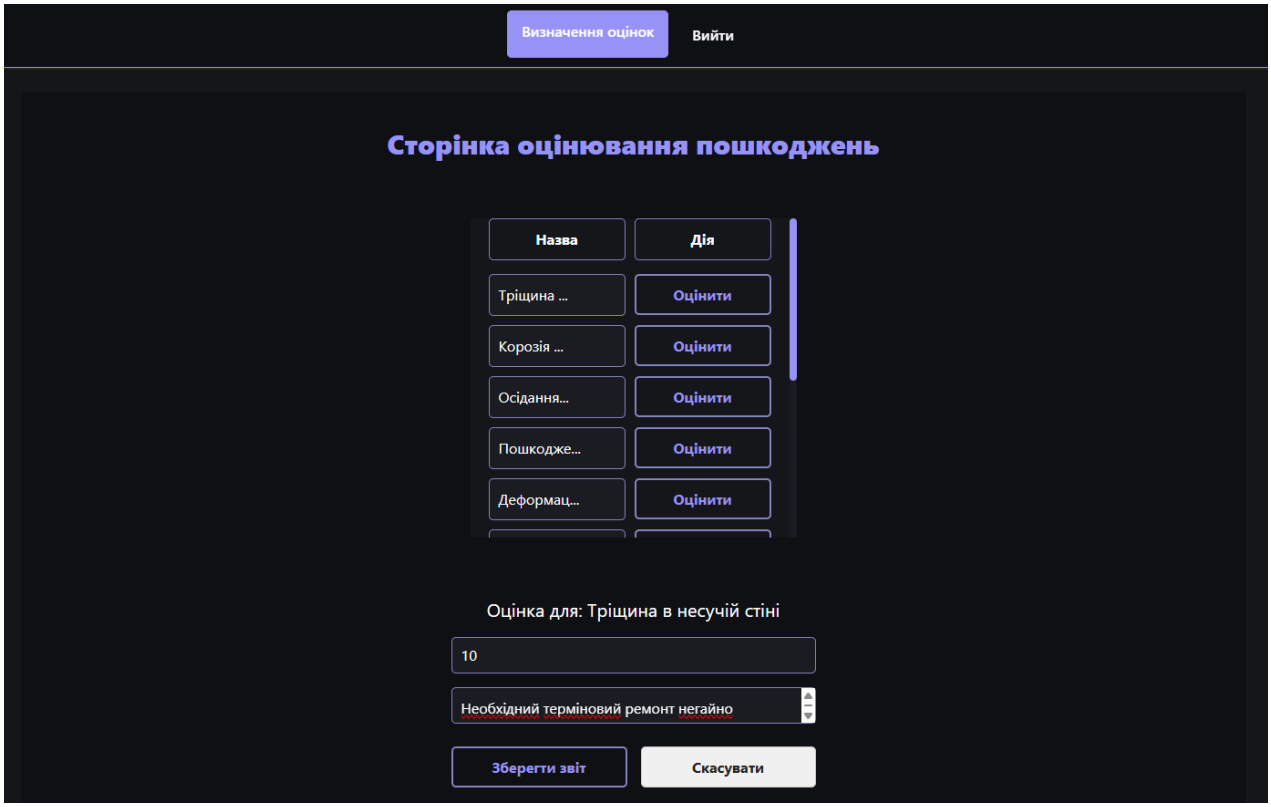


Рисунок 3.24 – Форма виставлення оцінки експертом

Сценарій 2. Дії користувача з роллю “Адміністратор”

2.1 Реєстрація з використанням секретного ключа (рис. 3.25)

- Перехід до реєстрації адміністратора;
- Введення спеціального ключа доступу.

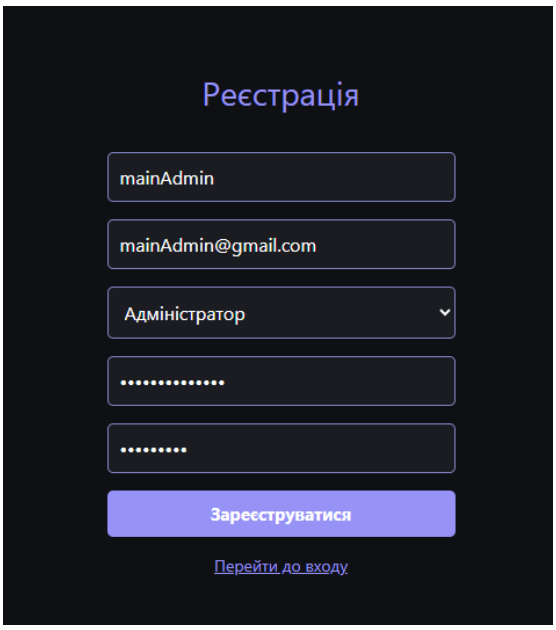


Рисунок 3.25 – Форма реєстрації адміністратора із введенням секретного ключа

2.2 Вхід до системи як адміністратор (рис. 3.26)

- Вхід за допомогою облікового запису адміністратора;
- Відображення інтерфейсу з розширеними правами.

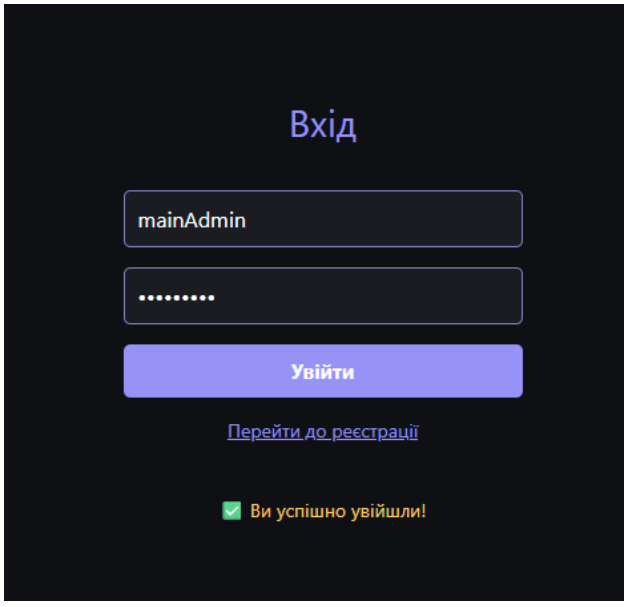


Рисунок 3.26 – Головне меню адміністратора після входу

2.3 Додавання та редагування експерта (рис. 2.27)

- Створення нового експерта;
- Редагування існуючих облікових записів.

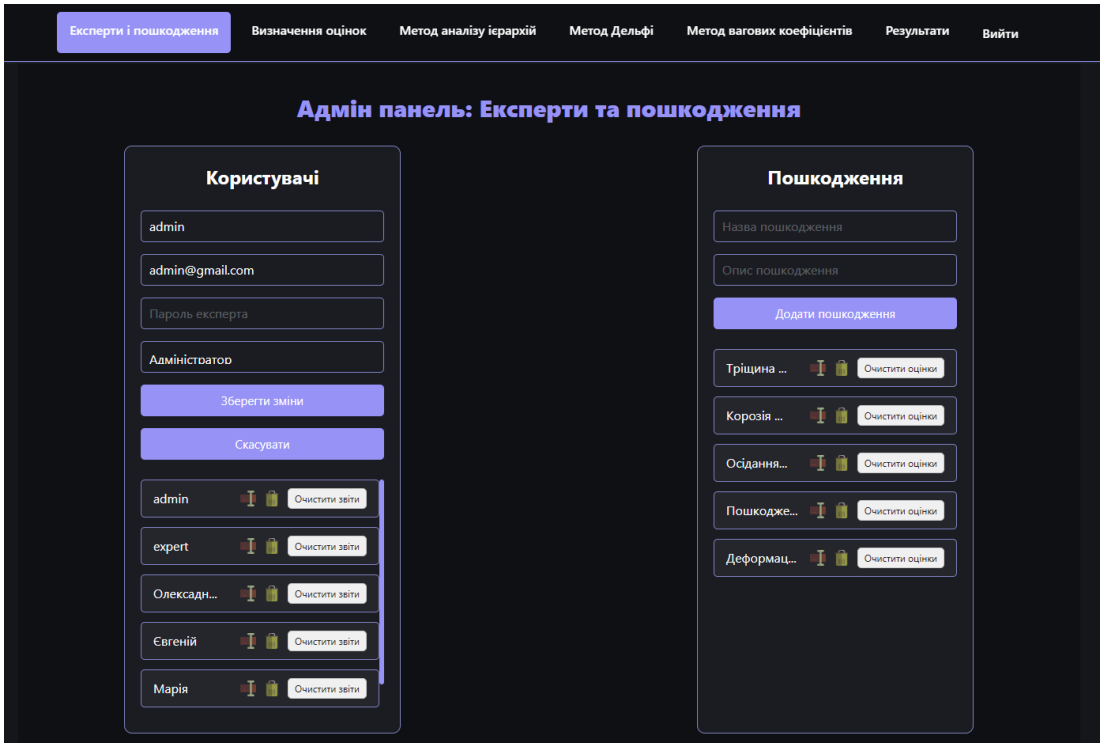


Рисунок 3.27 – Інтерфейс додавання експерта

2.4 Додавання та редагування пошкодження (рис. 3.28)

- Внесення нового дефекту;
- Зміна існуючих даних у базі.

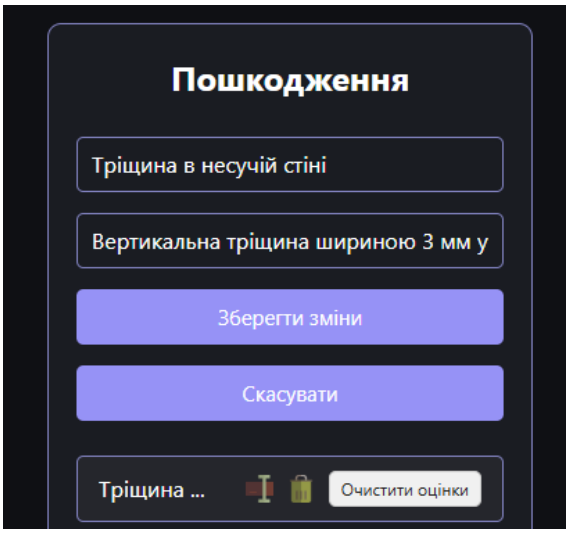


Рисунок 3.28 – Додавання нового пошкодження як адміністратор

2.5 Перевірка результатів оцінювання

Метод Дельфі (рис. 3.29):

Рисунок 3.29 – Результати оцінювання методом Дельфі

Метод аналізу ієрархій (рис. 3.30):

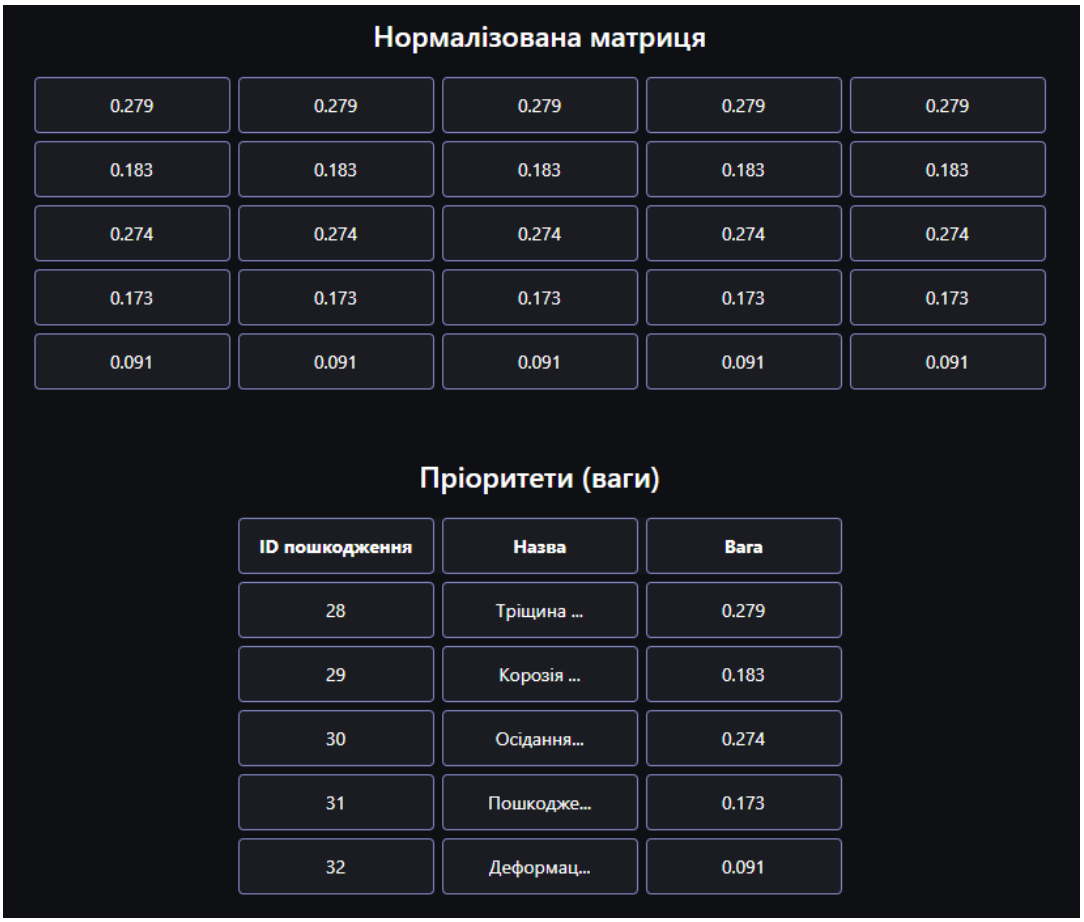


Рисунок 3.30 – Виведення підсумкових оцінок парних порівнянь

Метод зважених коефіцієнтів (MWC) (рис. 3.31):

Рисунок 3.31 – Пріоритети на основі MWC-методу

Загальні результати (рис. 3.32):

Рисунок 3.32 – Зведена таблиця загальних результатів обробки

Система успішно пройшла перевірку всіх ключових функціональних сценаріїв. Ролі "експерт" та "адміністратор" мають доступ до відповідного функціоналу згідно з логікою авторизації. Всі перевірені модулі — від реєстрації до обробки результатів — працюють згідно з очікуваннями. Тестування підтвердило коректність взаємодії між компонентами системи, стабільність роботи алгоритмів оцінки та цілісність збереження інформації в базі даних.

3.10 Висновок до третього розділу

У третьому розділі було реалізовано повноцінну програмну частину веб-системи автоматизованої оцінки дефектів будівельних конструкцій. Вибір сучасного стеку технологій (React, TypeScript, Node.js, PostgreSQL) дозволив створити масштабовану, модульну та зручну в користуванні систему.

У межах цього етапу були виконані такі ключові задачі:

- **Реалізація клієнтської частини** – розроблено інтуїтивно зрозумілий інтерфейс з розмежуванням прав доступу для експерта та адміністратора, використано компонентний підхід та сучасні бібліотеки стилізації.
- **Реалізація серверної частини** – створено REST API з чіткою маршрутизацією, обробкою запитів та інтеграцією з базою даних; окремо реалізовано модуль обробки оцінок із підтримкою трьох методів
- **Розробка бази даних** – створено інфологічну, логічну та фізичну моделі БД, забезпечено цілісність і надійність збереження інформації.
- **Розробка алгоритмів** – імплементовано три математичних методи для обробки експертних оцінок; кожен із них реалізує окрему логіку
- **Тестування** – перевірено всі ключові сценарії взаємодії користувачів, підтверджено правильність обробки даних, стабільність роботи інтерфейсу та відповідність функціоналу технічним вимогам.

У результаті розробки система отримала такі **ключові властивості**:

- **Автоматизація експертного оцінювання** – оцінки обробляються на сервері, а результати миттєво виводяться у зручному форматі.
- **Узгодженість даних** – чітка структура БД забезпечує уніфіковане зберігання та швидкий доступ до інформації.
- **Гнучкість і масштабованість** – компонентна структура коду дозволяє легко розширювати систему новими модулями.

Таким чином, реалізована система повністю відповідає поставленим функціональним вимогам, демонструє стабільну роботу в різних сценаріях і є готовою до подальшого розгортання та використання у сфері управління станом будівельних конструкцій.

РОЗДІЛ 4. ЕРГОНОМІКА ТА ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ВЕБ-СИСТЕМИ

4.1 Ергономіка системи

Принципи зручності взаємодії користувача з інтерфейсом

Під час розробки веб-системи оцінки дефектів будівельних конструкцій особливу увагу було приділено ергономіці — зручності користування інтерфейсом з боку кінцевого користувача. Основними принципами, які були враховані, є:

- **Мінімізація навантаження на пам’ять користувача** — усі необхідні функції згруповані логічно та доступні за кілька кліків.
- **Інтуїтивна навігація** — реалізована через зрозуміле меню та вкладки.
- **Консистентність** — стилі оформлення, кольорові схеми, кнопки та таблиці витримані в єдиному стилі на всіх сторінках.
- **Зворотний зв’язок** — усі дії користувача супроводжуються сповіщеннями (успіх, помилка, валідація форм).
- **Простота введення даних** — система не перевантажена зайвими полями

Застосування цих принципів дозволило забезпечити низький поріг входу для користувачів без спеціальної технічної підготовки.

Аналіз елементів інтерфейсу: форми, таблиці, повідомлення

Інтерфейс системи складається з таких основних компонентів:

- **Форма авторизації** — реалізована з двома обов’язковими полями: логін (email) та пароль. При введенні некоректних даних відображається повідомлення про помилку.
- **Сторінка експерта** (рис.4.2) — містить блок введення оцінок до пошкоджень. Використано числові поля із валідацією значень у діапазоні.
- **Сторінка адміністратора** — надає доступ до управління користувачами, пошкодженнями, звітами.
- **Інтерфейс результатів** (рис. 4.1) — таблиці з результатами методів Delphi, AHP та MWC виводяться у зручному вигляді, ранжовані по зменшенню пріоритету.

Рисунок 4.1 – Інтерфейс перегляду результатів оцінки методом Delphi

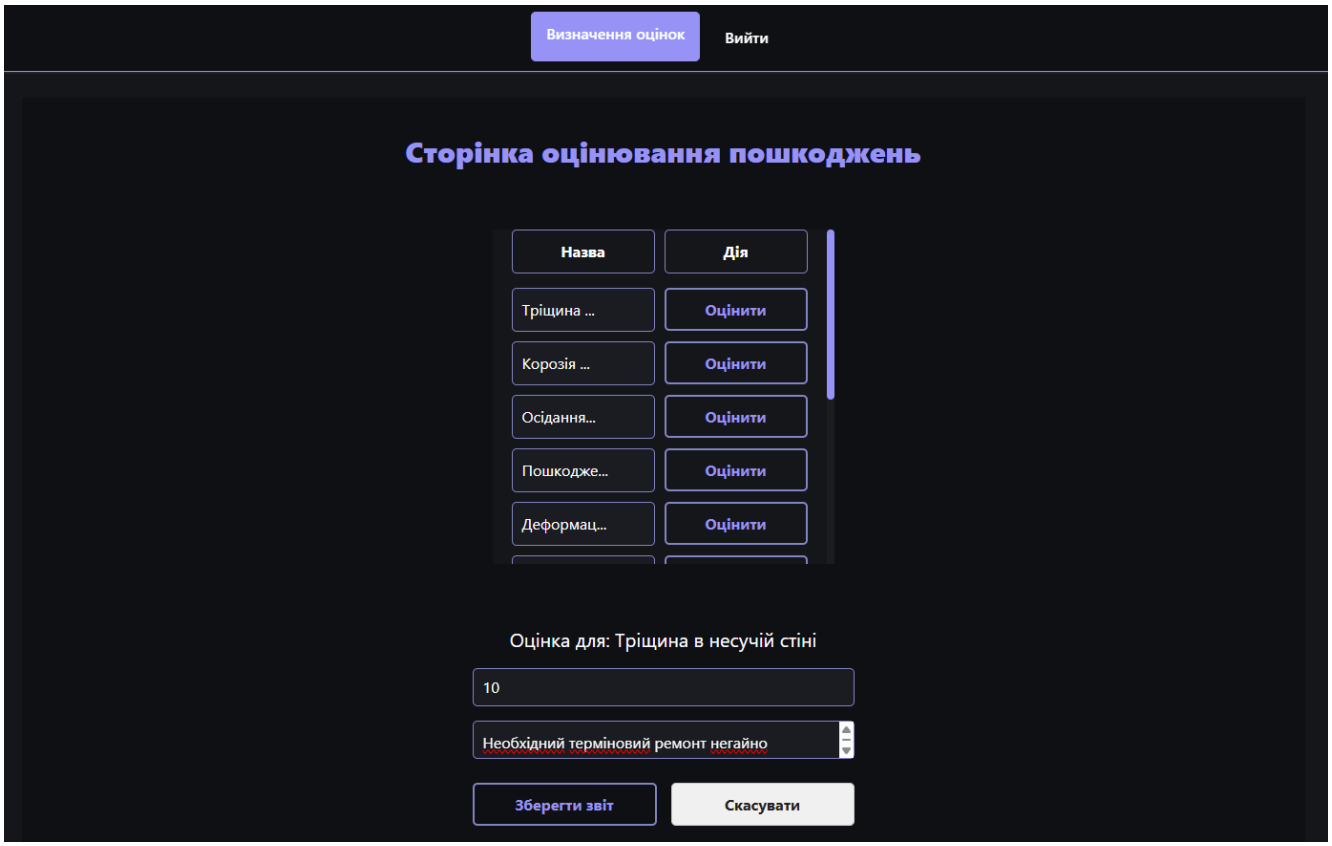


Рисунок 4.2 – Сторінка введення оцінок експертом

Усі таблиці мають фіксовані заголовки та адаптивну ширину колонок, що дозволяє ефективно працювати з великим обсягом даних.

Адаптивність і кросбраузерність веб-додатку

Система реалізована з урахуванням адаптивної верстки. Це дозволяє автоматично підлаштовувати відображення інтерфейсу під:

- **Роздільну здатність екрану** — для моніторів, ноутбуків, планшетів.
- **Мобільні пристрої** — базовий функціонал доступний на смартфонах.
- **Популярні браузери** — Chrome, Firefox, Microsoft Edge, Safari.

Завдяки використанню SCSS і CSS-фреймворку Bulma забезпечено стабільне та коректне відображення незалежно від пристрою або середовища користувача.

Відповідність вимогам ергономіки

Згідно з основними вимогами ергономіки інтерфейсів, розроблена система задовольняє такі критерії (табл. 4.1):

Таблиця 4.1

Параметри та їх оцінки

Параметр	**Реалізація в системі**
Час на виконання типових операцій	не перевищує 5–10 секунд
Кількість кліків до цільової дії	2–3 кліки
Візуальна ієрархія	заголовки, відступи, розділення контенту
Валідація	помилки підсвічуються, супроводжуються текстовими підказками
Уніфікація стилів	однакові кнопки, поля, кольори на всіх сторінках

Інтерфейс є простим у використанні, візуально привабливим і відповідає сучасним стандартам UI/UX.

Оцінка зручності інтерфейсу на прикладі сценаріїв

Сценарій 1: Експерт вводить оцінки

1. Авторизується в системі.
2. Обирає потрібне пошкодження зі списку.
3. Вводить оцінку відповідно до критеріїв.
4. Натискає кнопку “Зберегти” — з’являється повідомлення про успішне збереження.

Інтерфейс дозволяє швидко та інтуїтивно вводити інформацію, без додаткових інструкцій або довгого навчання.

Сценарій 2: Адміністратор керує користувачами, об’єктами та результатами

1. Після входу в систему адміністратор отримує доступ до **панелі керування**, де:
    - Керує **користувачами**: створює нових, редагує або видаляє існуючих, задає ролі;
    - Керує **пошкодженнями**: додає опис нових дефектів, редагує або видаляє наявні.
2. У розділі аналітики адміністратор має доступ до **трьох окремих сторінок результатів**:
    - Метод Delphi;
    - Метод AHP (аналіз ієрархій);
    - Метод зважених коефіцієнтів (MWC).

Кожна сторінка представляє результати у вигляді таблиці з ранжуванням об’єктів за пріоритетом ремонту.

3. Також передбачена **сторінка зведеної інформації**, де зібрані всі оцінки експертів. Це дозволяє проводити порівняльний аналіз та виявляти розбіжності між оцінками.

Усі функції згруповані логічно, структура інтерфейсу зрозуміла, що забезпечує високу швидкість виконання адміністративних завдань.

4.2 Економічне обґрунтування

Визначення мети економічного аналізу

Метою економічного обґрунтування є аналіз доцільності розробки та впровадження веб-системи для автоматизованої оцінки дефектів будівельних конструкцій з точки зору витрат, окупності та ефективності використання ресурсів. Важливо не лише розрахувати витрати на створення системи, а й оцінити потенційні вигоди — зменшення витрат часу, оптимізація роботи персоналу, зниження навантаження на управлінський апарат.

В умовах післявоєнного відновлення інфраструктури в Україні така система може відіграти ключову роль, дозволяючи швидко виявляти критичні пошкодження, ранжувати об’єкти за пріоритетністю ремонту та забезпечити прозорий процес прийняття рішень.

Опис проєктованого продукту

Проєктована веб-система призначена для збору, обробки та аналізу оцінок експертів щодо стану будівель. Вона реалізує функціонал введення пошкоджень, виставлення числових оцінок, розрахунків за методами Delphi, AHP, MWC, та формування результатів у вигляді таблиць пріоритетності.

Інтерфейс адаптований під дві ролі: експерт і адміністратор. Система реалізована з використанням React, Node.js і PostgreSQL, що забезпечує масштабованість, відкритість та можливість подальшого розширення.

Ринок збуту

Цільова аудиторія розробленої системи включає:

- державні служби технічного нагляду;
- органи місцевого самоврядування;
- комунальні підприємства;
- приватні компанії, що займаються реконструкцією будівель;
- освітні заклади (для дослідницької або навчальної діяльності).

Актуальність системи підсилюється загальнонаціональними зусиллями щодо відбудови України та потребою у прозорих механізмах оцінювання пошкоджених об'єктів.

Розрахунок витрат на розробку веб-системи

Умовно розглянемо повний цикл створення веб-системи як проєкт з обмеженим терміном реалізації. Основні статті витрат включають (табл. 4.2):

Таблиця 4.2

Витрати на розробку

Стаття витрат	**Орієнтовна сума, грн**
Аналіз предметної області	4 000
Проєктування архітектури системи	5 000
Розробка інтерфейсу (frontend)	10 000
Програмування серверної частини	12 000
Проєктування БД та зв'язків	3 000
Тестування та відлагодження	5 000
Документація, підготовка до впровадження	1 500
Разом:	**40 500**

Витрати розраховано на основі середньоринкових ставок для молодших розробників або фріланс-команд. У випадку самостійного виконання (наприклад, дипломним проєктом), грошові витрати на оплату праці відсутні, однак вони враховуються для оцінки економічної ваги проєкту.

Порівняння із альтернативними рішеннями

На ринку існують комерційні ПЗ-продукти для технічної діагностики будівель, однак більшість з них мають вузьку спеціалізацію, високу вартість ліцензування або не передбачають використання методів колективної експертної оцінки (табл. 4.3).

Таблиця 4.3

Порівняння з альтернативами

Параметр	**Власна система**	**Комерційне ПЗ**
Початкова вартість	0 – 40 000 грн	80 000 – 150 000 грн
Гнучкість адаптації	Висока	Обмежена
Вартість підтримки	Низька / нульова	Висока
Відкритість алгоритмів оцінки	Повна	Обмежена / закрита
Можливість локалізації	Так	Частково

Таким чином, створення власної веб-системи є економічно доцільним при потребі налаштування під специфіку українських реалій, впровадження української мови інтерфейсу, гнучких методів експертного аналізу.

Оцінка економії від впровадження системи

Використання веб-системи дозволяє скоротити витрати в кількох напрямках:

- **Зменшення тривалості прийняття рішень**: у ручному режимі обробка одного об’єкта може займати до 2–3 днів, тоді як автоматизована система дозволяє обробити десятки об’єктів за декілька годин.
- **Оптимізація витрат на фахівців**: при використанні системи необхідність у залученні великої кількості аналітиків для сортування об'єктів за пріоритетами зменшується.
- **Скорочення витрат на паперовий документообіг**: результати одразу зберігаються в електронній формі з можливістю експорту.

Умовно при роботі з 200 об'єктами на місяць економія часу та ресурсів може становити (табл. 4.4):

Таблиця 4.4

Порівняння використання ресурсів

Показник	**Без системи**	**Із системою**
Людино-дні на аналіз об'єктів	~20	~4
Вартість людських ресурсів (грн)	25 000	5 000
Економія коштів на місяць	—	~20 000 грн

За 6 місяців роботи система може заощадити понад 120 000 грн при регулярному використанні.

Показники ефективності: строк окупності, рентабельність, ROI

Для завершення економічного обґрунтування проведемо оцінку основних показників:

- **Витрати на розробку**: 40 500 грн
- **Щомісячна економія**: ~20 000 грн
- **Окупність (Payback period)**:

$$\mathrm{T}_{\text{окупн}} = \frac{40500}{20000} = 2.03\text{міс.}$$

- **Коефіцієнт рентабельності інвестицій (ROI)**:

$$ROI = \frac{\text{прибуток}}{\text{витрати}} = \frac{120000 - 40500}{40500} \approx 196\%$$

Таким чином, система окупається протягом перших 2 місяців використання і надалі приносить економічну вигоду. Це свідчить про високу ефективність розробки власного веб-продукту для вирішення спеціалізованої задачі.

Організаційний і юридичний аспект

Як дипломний проєкт або open-source рішення, система не потребує ліцензування або реєстрації. Якщо система буде виводитися на ринок — можливе створення ФОП або ТОВ, а також реєстрація авторського права. Особливу увагу слід приділити захисту персональних даних користувачів, якщо вони зберігатимуться.

Оцінка ризиків (табл. 4.5)

Таблиця 4.5

Потенційні ризики

Потенційний ризик	**Засіб пом’якшення**
Втрата даних при збоях сервера	Регулярне резервне копіювання
Атаки на систему або несанкціонований доступ	Авторизація, HTTPS, обмеження доступу
Помилки при обробці даних	Валідація введених даних, логування
Неможливість масштабування	Модульна архітектура, трирівнева модель

Розроблена веб-система є **економічно доцільною та виправданою альтернативою** дорогим комерційним програмним продуктам, особливо в умовах обмеженого фінансування державного або муніципального рівня. Її впровадження забезпечує **значне скорочення витрат на аналітичну діяльність**, **підвищення оперативності** в прийнятті рішень та **зменшення навантаження на персонал**.
Система дозволяє:

- автоматизувати процес експертного оцінювання будівельних дефектів;
- централізувати збір даних і формування пріоритетів для ремонту;
- забезпечити прозорість, об'єктивність і гнучке налаштування під конкретні потреби.

Згідно з проведеними розрахунками, **строк окупності становить близько 2 місяців**, а **коефіцієнт рентабельності (ROI)** сягає **майже 200%**. Загальна економія ресурсів протягом піврічного періоду використання може перевищувати **120 000 грн**, що свідчить про **високу ефективність, швидку віддачу інвестицій та раціональність впровадження** такого IT-рішення в процеси відновлення будівельної інфраструктури України.

4.3 Висновки до розділу 4

У розділі було проаналізовано ергономіку інтерфейсу та економічну доцільність створення веб-системи. Інтерфейс відповідає вимогам зручності: простий у використанні, логічно структурований, адаптований до різних пристроїв і не потребує додаткового навчання користувача.

Реалізовані сценарії роботи експерта й адміністратора забезпечують швидке виконання ключових дій — введення, перегляд, редагування даних, а також контроль за результатами. Завдяки зручному дизайну та функціональності, система підходить для щоденного застосування у сфері оцінки пошкоджень.

Економічний аналіз показав, що розробка власної системи є вигідною альтернативою комерційним рішенням. Система окупається за 2 місяці використання та дозволяє заощаджувати ресурси вже з перших тижнів.

У результаті впровадження забезпечується швидке прийняття рішень, зменшення витрат і прозорість експертних оцінок. Це робить систему ефективним інструментом у сфері відновлення будівельної інфраструктури.

ЗАГАЛЬНІ ВИСНОВКИ

У процесі виконання дипломної роботи на тему **«Розробка веб-системи автоматизованої оцінки дефектів будівельних конструкцій»** було всебічно проаналізовано проблему діагностики технічного стану будівель та створено інструмент, що дозволяє її ефективно вирішувати за допомогою сучасних веб-технологій.

У вступі обґрунтовано **актуальність проєкту**, зумовлену нагальною потребою в ефективних засобах оцінювання пошкоджень у післявоєнний період. У першому розділі досліджено методи оцінювання: традиційні, інструментальні, аналітичні та інноваційні. Особливу увагу приділено **методам експертного аналізу**, на яких ґрунтується побудована система.

У другому розділі спроєктовано **трирівневу архітектуру** веб-системи, визначено логіку обробки даних, побудовано структуру функціональних блоків і модель взаємодії між компонентами. Сформовано повноцінний проєкт для реалізації.

У третьому розділі реалізовано **фронтенд на React/TypeScript** та **бекенд на Node.js/Express**, з підключенням до PostgreSQL. Реалізовано алгоритми обробки оцінок за методами **Delphi, AHP та зважених коефіцієнтів (MWC)**. Проведено тестування, підтверджено стабільність роботи, коректність обчислень і зручність інтерфейсу.

У четвертому розділі проаналізовано **ергономіку інтерфейсу**, що відповідає сучасним UI/UX-вимогам. Здійснено техніко-економічне обґрунтування. За підсумками аналізу:

- **строк окупності системи** становить близько **2 місяців**;
- **економія ресурсів** при її використанні перевищує **120 000 грн за пів року**;
- **рентабельність (ROI)** — понад **190%**.

Система виявилася **ефективнішою** за наявні комерційні аналоги, оскільки пропонує відкритість алгоритмів, гнучкість адаптації та можливість локалізації під українські реалії.

Авторський внесок

Здобувачем **самостійно реалізовано повноцінний веб-додаток** з нуля, включно:

- проектуванням архітектури та її розробка;
- реалізацією алгоритмів експертної оцінки;

Ключовою **новизною** є комплексне поєднання математичних методів експертного оцінювання у єдиному веб-інтерфейсі, що автоматично формує пріоритети ремонту на основі узгоджених оцінок кількох фахівців.

Відповідність завданню

Усі поставлені в завданні цілі досягнуті:

- проведено огляд методів;
- спроєктовано архітектуру;
- реалізовано програмне забезпечення;
- виконано тестування;
- проведено економічне обґрунтування.

Рекомендації

На основі результатів роботи рекомендується:

1. Використовувати розроблену систему в органах, що займаються технічним оглядом будівель.
2. Розширити систему на інші сфери, де потрібне експертне оцінювання (екологія, інфраструктура, освіта).
3. Поширити продукт як open-source або подати його на розгляд до державних ініціатив цифрової трансформації.

Можливість впровадження

Система **може бути впроваджена на практиці** в комунальних структурах, проєктних бюро чи навіть університетських лабораторіях. Прикладне значення підтверджується технічною реалізацією, відсутністю потреби в ліцензуванні та можливістю адаптації до різних сценаріїв використання.

СПИСОК ДЖЕРЕЛ

1. Смирнов В. Аналіз сейсмостійкості будівельних конструкцій. — Київ: Наукова література, 2021.
2. Klimenko A. Structural Flood Damage: Case Study in Europe. — Berlin: Springer, 2020.
3. Утворення тріщин і механіка структурного руйнування кранових зварних металоконструкцій. *Головна*. URL: https://kzpto.com.ua/uk/kranovi-zvarni-metalokonstrukcij/ (дата звернення: 15.01.2025).
4. Самочинне (самовільне) будівництво: ризики та наслідки його зведення. *Legal Partner*. URL: https://legalpartner.com.ua/ru/samochinne-samovilne-budivnicztvo-riziki-ta-naslidki-jogo-zvedennya/ (дата звернення: 15.01.2025).
5. Вплив вібрації на будівельні конструкції. *inul.hodynnyk.cx.ua*. URL: https://inul.hodynnyk.cx.ua/articles/vpliv-vibracii-na-budivelni-konstrukcii.html (дата звернення: 16.01.2025).
6. Старіння металу: що це таке і коли застосовується?. *ООО «КАРБАЗ»* URL: https://karbaz.com.ua/uk/statti-ta-publikaci/starinnja-metalu-shho-ce-take/ (дата звернення: 16.01.2025).
7. Технічне обстеження будівельних конструкцій, будівель та споруд. *NOVOTEST*. URL: https://novotest.ua/ua/stati/tehnicheskoe-obsledovanie-stroitelnyh-konstrukcii.html (дата звернення: 16.01.2025).
8. Методичні рекомендації щодо застосування тепловізорів на пожежі [Електронний ресурс] / ДСНС України. – Івано-Франківськ, 2020. – 17 с. – Режим доступу: https://if.dsns.gov.ua/upload/2/1/3/1/5/1/7/metod-rekom-shhodo-zastosuvannia-teplovizoriv-na-pozezi.pdf, вільний. – (дата звернення: *16.01.2025*).
9. Дубенець В. Г. Основи МСЕ [Електронний ресурс] / В. Г. Дубенець. – Чернігів: ЧНТУ, 2019. – 132 с. – URL: https://ir.stu.cn.ua/bitstream/handle/123456789/11250/ДубенецьВГ.Основи%20МСЕ.pdf, вільний. – (Дата звернення: *14.01.2025*).

10. Bulat A.F., Bunko T.V., Shatov S.V., Kokoulin I.Ye., Yashchenko I.O., Papirnyk R.B. Using the UAV for inspecting accident sites and threatened areas in case of emergency situations occurred in coal mines and on build objects. *About the journal*. URL: http://www.geotm.dp.ua/index.php/en/collection/348-geotekhnicheskaya-mekhanika-scientific-papers-published-in-2018/collection-of-scientific-papers-geotekhnicheskaya-mekhanika-issue-141/3665 (дата звернення: 17.01.2025).
11. Іванова К. Супутникові знімки пожеж у Лос-Анджелесі: вогонь знищує цілі квартали. *ГЛАВКОМ*. URL: https://glavcom.ua/world/observe/suputnikovi-znimki-pozhezh-v-los-andzhelesi-vohon-znishchuje-tsili-kvartali-1039651.html (дата звернення: 18.01.2025).
12. Основи експертних оцінок у будівельній сфері. URL: https://press.vntu.edu.ua/index.php/vntu/catalog/download/324/612/651-1.
13. Учасники проєктів Вікімедіа. Дельфійський метод – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Дельфійський_метод (дата звернення: 18.01.2025).
14. Методи оцінки і ліквідації пошкоджень в конструкціях будівель. *Компанія ПОЛИГОНАЛЬ - виробництво будівельних конструкцій, інжинірингові послуги*.
URL: https://polygonal.com.ua/metodi_otsInki_lIkvIdatsIYi_poshkodzhen_konstruktsIyah_budIvel.php (дата звернення: 18.01.2025).
15. Аналітичний центр "Український союз промисловців і підприємців". Відновлення України: фінансування та економічне зростання [Електронний ресурс] / УСПП. – Київ, 2020. – 32 с. – URL: https://uscc.ua/uploads/page/5fc101712e613.pdf, вільний. – (дата звернення: *30.01.2025*).
16. ШІ в будівництві: приклади, роль та переваги | Wezom. *IT WEZOM - Київ, Україна*. URL: https://wezom.com.ua/ua/blog/shi-v-budivnitstvi-vpliv-na-transformatsiyu-budivelnoyi-galuzi (дата звернення: 16.01.2025).

17. Учасники проєктів Вікімедіа. Триярусна архітектура – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Триярусна_архітектура (дата звернення: 10.04.2025).
18. Учасники проєктів Вікімедіа. Клієнт-серверна архітектура – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Клієнт-серверна_архітектура (дата звернення: 12.04.2025).
19. Що таке структура сайту: Повний посібник з розробки та оптимізації від MSystem. *Інтернет агентство MSystem*. URL: https://msystem.ua/ua/shho-take-struktura-sajtu/ (дата звернення: 20.04.2025).
20. Логічна структура сайту: ключ до успіху в SEO. bizZzdev. URL: https://bizzzdev.com/why-do-you-need-the-proper-logic-of-the-site-structure/ (дата звернення: 22.04.2025).
21. Про вимоги до форматів даних електронного документообігу в органах державної влади. Формат електронного повідомлення. *Офіційний вебпортал парламенту України*. URL: https://zakon.rada.gov.ua/go/z1306-11 (дата звернення: 24.04.2025).
22. Веб-додатки. Поняття, компоненти та принципи роботи. Вікторія. URL: https://www.victoria.lviv.ua/library/students/wpr/lecture/5.docx (дата звернення: 05.05.2025).
23. Особливості застосування технології веб-сокетів для асинхронної взаємодії. Науковий вісник Львівської політехніки. URL: https://science.lpnu.ua/sites/default/files/journal-paper/2017/nov/6732/22-139-147.pdf (дата звернення: 06.05.2025).
24. Про затвердження Положення про набори даних, які підлягають оприлюдненню у формі відкритих даних. Кабінет Міністрів України. URL: https://www.kmu.gov.ua/npas/248573101 (дата звернення: 26.04.2025).
25. An Introduction to TypeScript: Static Typing for the Web â SitePoint. *SitePoint â Learn HTML, CSS, JavaScript, PHP, Ruby & Responsive Design*. URL: https://www.sitepoint.com/introduction-to-typescript/ (дата звернення: 10.05.2025).

26. GeeksforGeeks. React Component Based Architecture - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/react-component-based-architecture/ (дата звернення: 12.05.2025).
27. Sass: Sass Basics. *Sass: Syntactically Awesome Style Sheets*. URL: https://sass-lang.com/guide/ (дата звернення: 10.05.2025).
28. How To Structure React Projects From Beginner To Advanced. *Web Dev Simplified Blog*. URL: https://blog.webdevsimplified.com/2022-07/react-folder-structure/ (дата звернення: 08.05.2025).
29. Index | Node.js v24.1.0 Documentation. *Node.js – Run JavaScript Everywhere*. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 11.05.2025).
30. PostgreSQL. *PostgreSQL*. URL: https://www.postgresql.org/ (дата звернення: 13.05.2025).

ДОДАТОК А

```
import 'bulma/css/bulma.css';
import '@fortawesome/fontawesome-free/css/all.css';

import { createRoot } from 'react-dom/client';
import { Root } from './Root';

const container = document.getElementById('root') as HTMLDivElement;

createRoot(container).render(<Root />);
```

```
import './App.scss';
import { Outlet } from 'react-router-dom';
import { Navbar } from './components/NavBar/Navbar';

export const App = () => {
  return (
    <div data-cy="app">
      <Navbar />
      <div className="section">
        <div className="container">
          <Outlet />
        </div>
      </div>
    </div>
  );
};
```

```
import {
  Navigate,
  Route,
  HashRouter as Router,
  Routes,
} from 'react-router-dom';
import { App } from './App';
import { ExpertDamageSetupPage } from './components/Pages/ExpertDamageSetupPage';
import { GradesPage } from './components/Pages/GradesPage';
import { NotFoundPage } from './components/Pages/NotFoundPage';
import { ResultPage } from './components/Pages/ResultPage';
```

```
import { AuthPage } from './components/Pages/AuthPage';
import { PairwiseResultsPage } from './components/Pages/PairwiseResultsPage';
import { DelphiResultsPage } from './components/Pages/DelphiResultsPage';
import { MWCResultsPage } from './components/Pages/MwcResultsPage';

export const Root = () => (
  <Router>
    <Routes>
      <Route path="/" element={<App />}>
        {/* Головна сторінка → авторизація */}
        <Route index element={<AuthPage />} />

        {/* Перенаправлення /auth → / */}
        <Route path="auth" element={<Navigate replace to="/" />} />

        {/* Сторінки */}
        <Route path="expertsAndDamages" element={<ExpertDamageSetupPage />} />
        <Route path="grades" element={<GradesPage />} />
        <Route path="pairwise" element={<PairwiseResultsPage />} />
        <Route path="delphi" element={<DelphiResultsPage />} />
        <Route path="mwc" element={<MWCResultsPage />} />
        <Route path="results" element={<ResultPage />} />

        {/* 404 */}
        <Route path="*" element={<NotFoundPage />} />
      </Route>
    </Routes>
  </Router>
);
```

```
import { NavLink, useSearchParams, useNavigate } from 'react-router-dom';
import { getLinkClass } from '../../utils/getLinkClass';
import './Navbar.scss';
import { useLocalStorage } from '../../hooks/useLocalStorage';

export const Navbar = () => {
  const [search] = useSearchParams();
  const navigate = useNavigate();
  const [role, setRole] = useLocalStorage<string>('role', '');

  const handleLogout = () => {
```

```
    setRole('');
    navigate('/auth');
  };

  return (
      <nav  className="navbar  is-fixed-top  has-shadow"  role="navigation"  aria-
label="main navigation">
      <div className="container navbar__container">
        <div className="navbar-brand" id="nav">
          {role === '' && (
            <NavLink className={getLinkClass} to="/auth">
              Авторизація
            </NavLink>
          )}

          {role === 'admin' && (
            <>
              <NavLink className={getLinkClass} to="/expertsAndDamages">
                Експерти і пошкодження
              </NavLink>
              <NavLink
                className={getLinkClass}
                to={{
                  pathname: '/grades',
                  search: search.toString(),
                }}
              >
                Визначення оцінок
              </NavLink>
              <NavLink className={getLinkClass} to="/pairwise">
                Метод аналізу ієрархій
              </NavLink>
              <NavLink className={getLinkClass} to="/delphi">
                Метод Дельфі
              </NavLink>
              <NavLink className={getLinkClass} to="/mwc">
                Метод вагових коефіцієнтів
              </NavLink>
              <NavLink className={getLinkClass} to="/results">
                Результати
              </NavLink>
            </>
```

```
          )}

          {role === 'expert' && (
            <>
              <NavLink
                className={getLinkClass}
                to={{
                  pathname: '/grades',
                  search: search.toString(),
                }}
              >
                Визначення оцінок
              </NavLink>
            </>
          )}

          {role && (
            <button
              className="nItem"
              onClick={handleLogout}>
              Вийти
            </button>
          )}
        </div>
      </div>
    </nav>
  );
};
```

```
import React, { useEffect, useState } from 'react';
import { getDelphi, getMWC, getPairwiseResults } from '../../../api/calculationApi';
import './ResultPage.scss';
import { DelphiRow, AHPRow, DamageRow } from '../../../types';
import { truncateText } from '../../../utils/truncateText';

export const ResultPage: React.FC = () => {
  const [delphi, setDelphi] = useState<DelphiRow[]>([]);
  const [mwc, setMWC] = useState<DamageRow[]>([]);
  const [ahp, setAHP] = useState<AHPRow[]>([]);

  useEffect(() => {
```

```
    const fetchData = async () => {
      const delphiData = await getDelphi();
      const mwcData = await getMWC();
      const pairwiseData = await getPairwiseResults();
      setDelphi(delphiData);
      setMWC(mwcData);
      setAHP(pairwiseData.priorities);
    };
    fetchData();
  }, []);

  const getRowColor = (priority: string) => {
    switch (priority) {
      case 'високий': return 'red';
      case 'середній': return 'orange';
      case 'низький': return 'green';
      default: return 'gray';
    }
  };

  return (
    <div className="finalResultsPage">
            <h1   className="finalResultsPage__title">Фінальна   зведена   таблиця
пошкоджень</h1>
      <div className="finalResultsPage__table">
        <div className="finalResultsPage__row finalResultsPage__row--header">
          <div className="finalResultsPage__cell">ID</div>
          <div className="finalResultsPage__cell">Назва</div>
          <div className="finalResultsPage__cell">Delphi</div>
          <div className="finalResultsPage__cell">MWC</div>
          <div className="finalResultsPage__cell">AHP</div>
          <div className="finalResultsPage__cell">Пріоритет (MWC)</div>
        </div>
        {mwc.map((row) => {
          const delphiRow = delphi.find(d => d.damage_id === row.damage_id);
          const ahpRow = ahp.find(a => a.damage_id === row.damage_id);

          return (
            <div key={row.damage_id} className="finalResultsPage__row">
              <div className="finalResultsPage__cell">{row.damage_id}</div>
                   <div  className="finalResultsPage__cell">{truncateText(row.name,
12)}</div>
```

```
                <div className="finalResultsPage__cell">{delphiRow?.average ?? '-
'}</div>
              <div className="finalResultsPage__cell">{row.weighted_score}</div>
              <div className="finalResultsPage__cell">{ahpRow?.weight ?? '-'}</div>
              <div
                className="finalResultsPage__cell"
                style={{ color: getRowColor(row.priority), fontWeight: 'bold' }}
              >
                {row.priority}
              </div>
            </div>
          );
        })}
      </div>
    </div>
  );
};
```

```
const express = require('express');
const cors = require('cors');
const dotenv = require('dotenv');
dotenv.config();

const userRoutes = require('./routes/users');
const damageRoutes = require('./routes/damages');
const reportRoutes = require('./routes/reports');
const calculationRoutes = require('./routes/calculation');

const app = express();
app.use(cors());
app.use(express.json());

app.use('/api/users', userRoutes);
app.use('/api/damages', damageRoutes);
app.use('/api/reports', reportRoutes);
app.use('/api/calculation', calculationRoutes);

const PORT = process.env.PORT || 5001;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

```
const { Pool } = require('pg');
require('dotenv').config();

const pool = new Pool({
  user: process.env.DB_USER,
  host: process.env.DB_HOST,
  database: process.env.DB_NAME,
  password: process.env.DB_PASSWORD,
  port: process.env.DB_PORT,
});

module.exports = pool;
```

```
const express = require('express');
const router = express.Router();
const usersController = require('../controllers/usersController');

router.get('/', usersController.getAllUsers);
router.post('/register', usersController.registerUser);
router.post('/login', usersController.loginUser);
router.put('/:id', usersController.updateUser);
router.delete('/:id/clear-reports', usersController.clearExpertReports);
router.delete('/:id', usersController.deleteUser);

module.exports = router;
```

```
const pool = require('../db');

const getAllUsers = () => {
  return pool.query('SELECT id, name, email, role FROM users');
};

const findUserByName = (name) => {
  return pool.query('SELECT * FROM users WHERE name = $1', [name]);
};

const findUserByEmail = (email) => {
  return pool.query('SELECT * FROM users WHERE email = $1', [email]);
};
```

```
const insertUser = ({ name, email, hashedPassword, role }) => {
  return pool.query(
    'INSERT INTO users (name, email, password, role) VALUES ($1, $2, $3, $4) RETURNING
id, name, email, role',
    [name, email, hashedPassword, role]
  );
};

const updateUser = ({ id, name, email, role }) => {
  return pool.query(
    'UPDATE users SET name = $1, email = $2, role = $3 WHERE id = $4 RETURNING id,
name, email, role',
    [name, email, role, id]
  );
};

const deleteUser = (id) => {
  return pool.query('DELETE FROM users WHERE id = $1', [id]);
};

const clearExpertReports = (expertId) => {
  return pool.query('DELETE FROM reports WHERE user_id = $1', [expertId]);
};

module.exports = {
  getAllUsers,
  findUserByName,
  findUserByEmail,
  insertUser,
  updateUser,
  deleteUser,
  clearExpertReports
};
```

```
const pool = require('../db');

exports.getAllReports = async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM reports');
    res.json(result.rows);
  } catch (err) {
```

```
    res.status(500).json({ error: err.message });
  }
};

exports.createReport = async (req, res) => {
  const { title, score, user_id, damage_id } = req.body;
  try {
    const result = await pool.query(
      'INSERT INTO reports (title, score, user_id, damage_id) VALUES ($1, $2, $3,
$4) RETURNING *',
      [title, score, user_id, damage_id]
    );
    res.json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};


exports.deleteReport = async (req, res) => {
  const { id } = req.params;
  try {
    await pool.query('DELETE FROM reports WHERE id = $1', [id]);
    res.json({ message: 'Звіт видалено' });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};
```

```
const damageModel = require('../models/damageModel');

exports.getAllDamages = async (req, res) => {
  try {
    const result = await damageModel.getDamages();
    res.json(result.rows);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

exports.createDamage = async (req, res) => {
```

```
  const { name, description } = req.body;
  try {
    const result = await damageModel.createDamage({ name, description });
    res.json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

exports.updateDamage = async (req, res) => {
  const { name, description } = req.body;
  const { id } = req.params;
  try {
    const result = await damageModel.updateDamage({ id, name, description });
    res.json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

exports.deleteDamage = async (req, res) => {
  const { id } = req.params;
  try {
    await damageModel.deleteDamage(id);
    res.json({ message: 'Пошкодження видалено' });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

exports.clearDamageData = async (req, res) => {
  const { id } = req.params;
  console.log('⏎ Запит на очищення damage_id:', id);

  try {
    await damageModel.clearDamageReports(id);
    res.json({ message: 'Звіти очищено' });
  } catch (err) {
    console.error('❌ Помилка:', err);
    res.status(500).json({ error: 'Помилка при очищенні' });
  }
};
```

```
const bcrypt = require('bcrypt');
const dotenv = require('dotenv');
dotenv.config();
const userModel = require('../models/userModel');

exports.getAllUsers = async (req, res) => {
  try {
    const result = await userModel.getAllUsers();
    res.json(result.rows);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

exports.registerUser = async (req, res) => {
  const { name, email, password, role, adminSecret } = req.body;
  try {
    if (role === 'admin' && adminSecret !== process.env.ADMIN_SECRET) {
      return res.status(400).json({ error: 'Невірний секрет адміністратора' });
    }

    const existing = await userModel.findUserByEmail(email);
    if (existing.rows.length > 0) {
      return res.status(409).json({ error: 'Користувач з таким email вже існує' });
    }

    const hashedPassword = await bcrypt.hash(password, 10);
    const result = await userModel.insertUser({
      name,
      email,
      hashedPassword,
      role: role === 'admin' ? 'admin' : 'expert',
    });

    res.status(201).json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

exports.loginUser = async (req, res) => {
  const { name, password } = req.body;
```

```
  try {
    const result = await userModel.findUserByName(name);
    if (result.rows.length === 0) {
      return res.status(400).json({ error: 'Користувача не знайдено' });
    }

    const user = result.rows[0];
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(400).json({ error: 'Невірний пароль' });
    }

    res.json({
      message: 'Успішний вхід',
      role: user.role,
      userId: user.id,
      userName: user.name,
    });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

exports.deleteUser = async (req, res) => {
  const { id } = req.params;
  try {
    await userModel.deleteUser(id);
    res.json({ message: 'Користувача видалено' });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

exports.updateUser = async (req, res) => {
  const { name, email, role } = req.body;
  const { id } = req.params;

  try {
    const result = await userModel.updateUser({ id, name, email, role });
    res.json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
```

```
  }
};

exports.clearExpertReports = async (req, res) => {
  const { id } = req.params;

  try {
    await userModel.clearExpertReports(id);
    res.json({ message: 'Звіти експерта видалено' });
  } catch (err) {
    console.error('❌ Помилка при видаленні звітів експерта:', err);
    res.status(500).json({ error: 'Помилка при очищенні звітів експерта' });
  }
};
```

```
const pool = require('../db');

exports.getReports = () => pool.query('SELECT * FROM reports');

exports.createReport = ({ title, score, user_id, damage_id }) =>
  pool.query(
    'INSERT INTO reports (title, score, user_id, damage_id, created_at) VALUES ($1,
$2, $3, $4, NOW()) RETURNING *',
    [title, score, user_id, damage_id]
  );
```

ДОДАТОК Б

ЗАХИСТ ДИПЛОМНОЇ РОБОТИ

Тема: «Розробка веб-системи автоматизованої оцінки дефектів будівельних конструкції

Студент
групи КН-21-1
Рапін Олександр Олександрович

Керівник роботи
Доктор технічних наук, професор
Бородавка Євгеній Володимирович

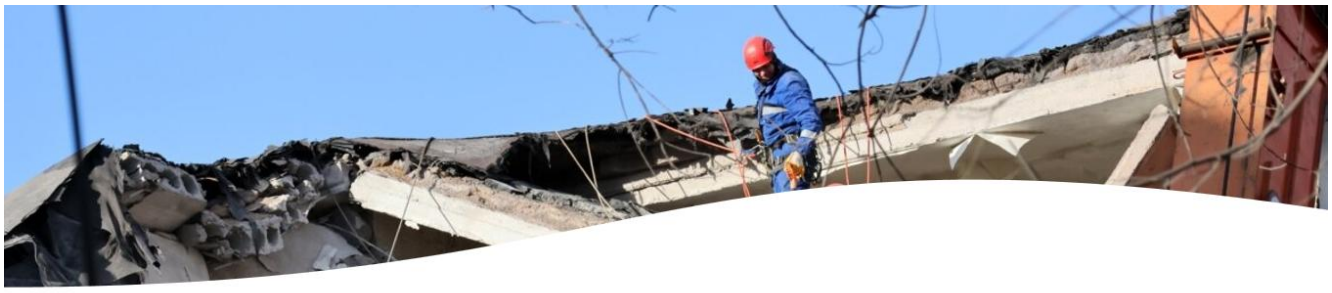
КНУБА 2025 р.

Мета:

Метою роботи є створення веб-системи для автоматизованої оцінки пошкоджень будівельних конструкцій на основі експертних методів.

Завдання:

- проаналізувати сучасні підходи до оцінювання;
- обрати відповідні методи обробки експертних оцінок;
- спроектувати архітектуру системи;
- реалізувати функціонал і протестувати систему;
- оцінити її ергономічність та економічну ефективність.



Методи оцінки пошкоджень

Візуальні методи
Первинний огляд конструкцій

Інструментальні
Ультразвук, тепловізія, рентген

Математичні моделі
Наприклад, метод скінченних елементів

Інноваційні технології
Дрони, 3D-сканери, IoT-датчики, штучний інтелект

Кожен метод має свої переваги, але вимагає часу, ресурсів або фахової підготовки. Саме тому акцент у системі зроблено на експертному підході з автоматизацією.

Ідея розробленої системи

Введення оцінок
Експерти оцінюють пошкодження будівель через зручний веб-інтерфейс

Обробка даних
Система автоматично обробляє ці дані за допомогою математичних методів

Формування рейтингу
Формується рейтинг пріоритетності ремонту

Це дає змогу:

- зменшити вплив суб'єктивного людського фактору,
- пришвидшити аналіз великої кількості об'єктів,
- забезпечити прозоре й обгрунтоване прийняття рішень.

Методи експертного оцінювання

Метод Дельфі

Анонімне багатотурове опитування, яке дозволяє досягти узгодженості між думками експертів.

Метод аналізу ієрархій

Побудова матриці парних порівнянь і визначення ваг кожного об'єкта через обчислення пріоритетів.

Метод зважених коефіцієнтів (MWC)

Врахування надійності оцінок кожного експерта на основі їх відхилення від середнього.

Таблиця порівняння методів експертних оцінок

Критерій / Метод	Точність	Об'єктивність	Простота реалізації	Швидкість виконання	Маштабо-ваність
Метод Дельфі	+	+	-	-	-
Метод парних порівнянь	+	+	+	-	+
Метод зважених коефіцієнтів	+	+	+	+	+
Метод аналізу ієрархій	+	+	-	-	+

Кожен метод дає різну глибину аналізу, а система дозволяє обрати потрібний у залежності від задачі.

Реалізація архітектури системи

Була реалізована трирівнева архітектура системи

Клієнтська частина (React + TypeScript)

Авторизація, введення оцінок, перегляд результатів (Delphi, AHP, MWC)

Серверна частина (Node.js + Express)

REST API, обробка методів експертного оцінювання, агрегація результатів

База даних (PostgreSQL)

Зберігання звітів, оцінок, пошкоджень, ролей користувачів

Такий підхід забезпечує прозору обробку експертних оцінок і автоматичне формування пріоритетності ремонту.

Компоненти та структура

Модуль формування результатів
Агрегує результати розрахунків і готує їх до виведення.

Сховище даних
Зберігає дані для швидкої та зручної їх обробки

Модуль авторизації
Дозволяє додавати нові об'єкти для оцінки

Навігаційна панель
Забезпечує зручний перехід між сторінками застосунку

Модуль обробки оцінок
Виконує обчислення за методами Delphi, AHP та MWC.

Інтерфейс перегляду результатів
Відображає зведені таблиці з пріоритетністю пошкоджень.

Панель адміністратора
Керує даними про користувачів та пошкодження

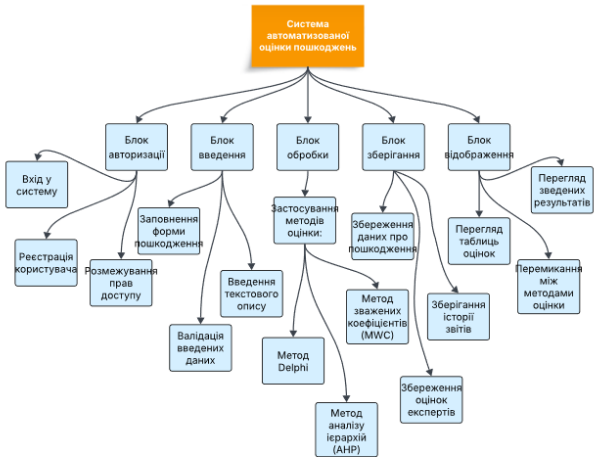
Користувачі мають ролі: експерт або адміністратор, що визначає їх доступ до функцій.

Функції веб-системи та користувачів

Система
Реалізує п’ять основних функціональних блоків: авторизація, введення, обробки, зберігання, відображення даних

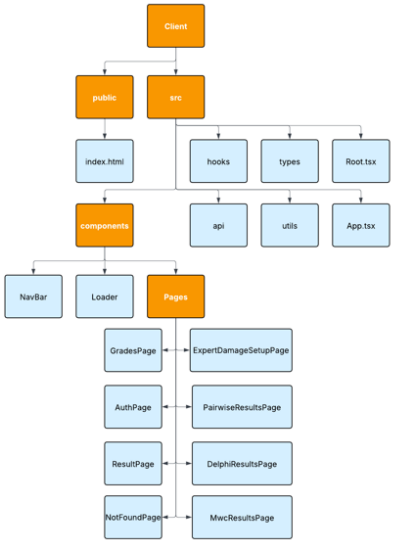
Адміністратор
Проходить реєстрацію та вхід з спеціальним паролем
Редагує пошкодження та користувачів, переглядає результат

Експерти
Проходять реєстрацію та вхід
Додають пошкодження та оцінюють їх

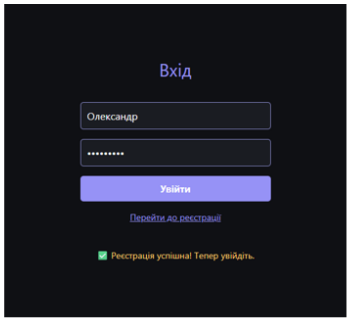


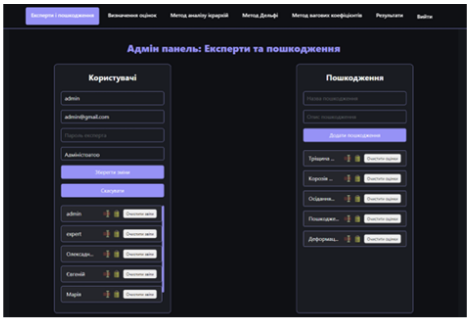
Структура клієнтської частини

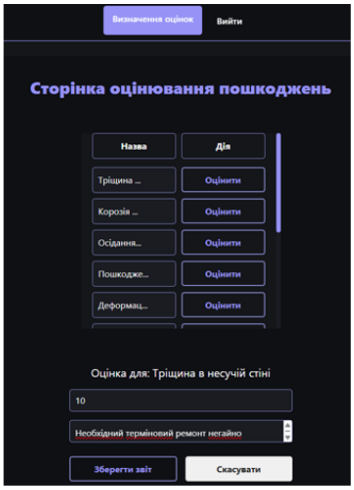
Блок	Призначення
App.tsx, Root.tsx	Основна структура сторінки та маршрутизація
components/	Повторно використовувані компоненти (меню, форми, лоадери)
Pages/	Сторінки застосунку: авторизація, оцінки, результати
api/	Запити до серверної частини (axios)
hooks/, utils/, types/	Користувацькі хуки, допоміжні функції та глобальні типи
public/index.html	Базовий HTML-файл запуску застосунку



Інтерфейс клієнтської частини

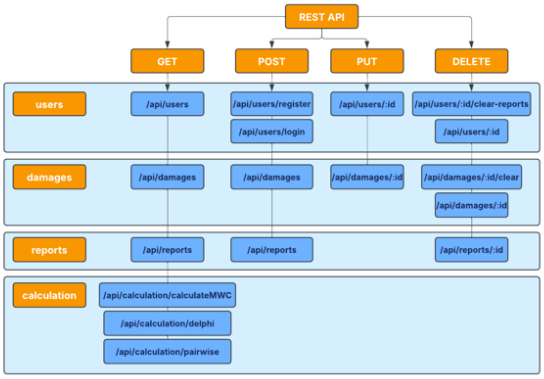




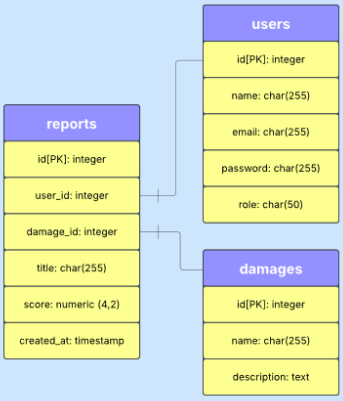


Реалізація серверної частини

Сутність	Опис
Авторизація Реєстрація	Реалізовано POST /api/****/login, перевірка ролі, паролю для адмінів.
Створення об'єктів	POST /api/**** — збереження даних до бази.
Розрахунки	GET-запити до /api/calculation/delphi, /pairwise, /calculateMWC.
Формування результатів	Сервер обробляє дані з reports і повертає JSON-результати.



Серверна частина працює на Node.js з фреймворком Express. Логіка розрахунків методів AHP, Delphi та MWC винесена в окремі модулі.



База даних

Таблиця	Призначення	Зв'язки
Користувачі (users)	Зберігання даних про експертів та адміністраторів	Один-до-багатьох зі звітами (reports)
Пошкодження (damages)	Інформація про об'єкти оцінювання	Один-до-багатьох зі звітами (reports)
Звіти (reports)	Збереження оцінок пошкоджень від конкретних користувачів	Зв'язок "багато-до-одного" з users і "багато-до-одного" з damages

Для зберігання інформації використано PostgreSQL. Дані зв'язані між собою через зовнішні ключі. Це забезпечує надійність і збереження історії змін.

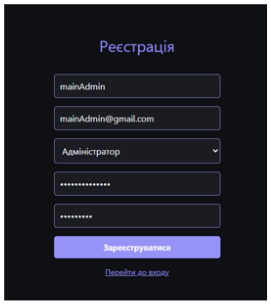
Тестування системи

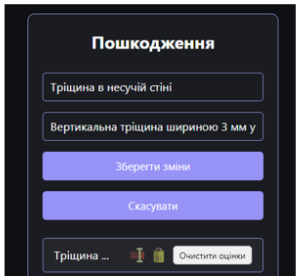
Було протестовано ключові сценарії:

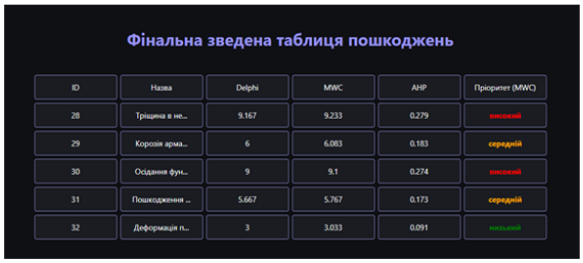
– робота експерта: авторизація, реєстрація, введення оцінок;

– дії адміністратора: авторизація, реєстрація, керування користувачами та пошкодженнями, запуск обчислень.

Усі функції працюють стабільно. Результати трьох методів обробки оцінок збігаються логічно і мають узгодженість.







Ергономіка та ефективність

Інтерфейс системи спроєктовано згідно принципів UI/UX:

всі дії виконуються за 2–3 кліки;

зрозумілі кнопки, адаптивний дизайн.

Завдяки автоматизації зменшено час аналізу кожного об'єкта у 5 разів.

Окупність — 2 місяці, розрахований ROI — близько 196%.

$$ROI = \frac{\text{прибуток}}{\text{витрати}} = \frac{120000 - 40500}{40500} \approx 196\%$$

$$T_{\text{окупн}} = \frac{40500}{20000} = 2.03\text{міс.}$$

Висновки роботи

Розроблено повноцінну веб-систему для експертної оцінки пошкоджень будівельних конструкцій.

Реалізовано три математичні методи обробки експертних оцінок: Delphi, AHP, MWC.

Система протестована і показала стабільну роботу з реальними сценаріями.

Проведено ергономічну та економічну оцінку, яка підтвердила практичну доцільність впровадження.

Отримані результати можуть бути використані як в академічних, так і в прикладних цілях — зокрема для служб технічної експертизи, будівельних компаній та відновлювальних проєктів.