НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

Комп'ютерних систем, мереж та кібербезпеки

_____ Касаткін Д.Ю., к.пед.н., доц.

(підпис) (ПІБ, вчене звання і ступінь)

«___»_____2025 р.

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

На тему: «Розроблення комп'ютерної системи оцінювання рівня ризиків для невиробничих процесів»

Спеціальність 123 «Комп'ютерна інженерія»

Гарант освітньої програми

к.фіз.-мат.н., доц. (підпис) (ПІБ) / Нікітенко Є.В. /

Керівник дипломного проекту: _____ / Шкарупило В.В. /

(підпис) (ПІБ)

Виконав: _____ / Осадчий Д.О/

(підпис) (ПІБ)

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

«ЗАТВЕРДЖУЮ»
завідувач кафедри
комп'ютерних систем, мереж та кібербезпеки
_____/ Касаткін Д.Ю., к.п.н., доц. /
підпис ПІБ, вчене звання і ступінь
«__» _____ 20__ р.

З А В Д А Н Н Я

ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ БАКАЛАВРСЬКОЇ СТУДЕНТУ

Осадчого Дмитра Олеговича
(прізвище, ім'я, по батькові)
Спеціальність (напрям підготовки): комп'ютерна інженерія
Тема кваліфікаційної бакалаврської роботи: «Розроблення комп'ютерної системи оцінювання рівня ризиків для невиробничих процесів»
затверджена наказом ректора НУБіП України від "_16__" _12_2024_р. №2250.С__
Термін подання завершеної роботи на кафедру _____
Вихідні дані до кваліфікаційної бакалаврської роботи __проектування та розробка комп'ютерної системи оцінювання рівня ризиків для невиробничих процесів
Перелік питань, що підлягають розробці:
1. _____
2. _____
3. _____
Перелік графічного матеріалу (за потреби) _____

Дата видачі завдання "_____"_____2024 р.

Керівник кваліфікаційної роботи _____ Шкарупило В.В., д.т.н., професор
(підпис) (прізвище та ініціали)
Завдання прийняв до виконання _____ Осадчий Д.О
(підпис) (прізвище та ініціали студента)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз предметної області	04.02.2025 р.	Виконано
2	Проектування системи	15.03.2025 р.	Виконано
3	Реалізація системи	10.04.2025 р.	Виконано
4	Тестування системи	01.05.2025 р.	Виконано
5	Оформлення пояснювальної записки	13.06.2025 р.	Виконано
6	Оформлення графічного матеріалу	13.06.2025 р.	Виконано

Студент _____ **Д.О. Осадчий**
(підпис) (ініціали та прізвище)

Керівник проекту (роботи) _____ **В.В Шкарупило**
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Пояснювальна записка: 69 сторінки, 13 рисунків, 4 таблиць, 14 лістингів, 1 додатки, 10 джерел.

КОМП’ЮТЕРНА СИСТЕМА, TYPESCRIPT, NODE.JS, NEST.JS, REACT, MONGODB, JWT, OPEN AI.

Предмет дослідження - процес розробки комп'ютерної програми для оцінки ризиків у невиробничих процесах

Метою дослідження є розробка комп'ютерної програми для автоматизованого аналізу та оцінки ризиків у невиробничих процесах з використанням штучного інтелекту.

Проект складається з трьох розділів.

У першому розділі проаналізовано технічні характеристики та досліджено існуючі програми оцінки ризиків для невиробничих процесів.

Другий розділ описує вибір та обґрунтування технологій, що використовуються для досягнення поставлених цілей.

У третьому розділі представлено повну реалізацію додатку на основі технологій MERN (розробка клієнта і сервера), а також проектування бази даних.

Результатом цієї роботи є веб-система, яка дозволяє користувачеві, використовуючи моделі штучного інтелекту, оцінювати рівень ризику в невиробничих процесах на основі вхідних даних.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ

ОС – Операційна система

ПЗ – Програмне забезпечення

БД – База даних

HTML — стандартизована мова розмітки документів для перегляду веб-сторінок (англ. Hyper Text Markup Language)

CSS — мова стилю сторінок (англ. Cascading Style Sheets)

JavaScript — мова програмування.

TypeScript - мова програмування.

MongoDB - Документо-орієнтована система керування базами даних

React — бібліотека JavaScript для розробки користувацького інтерфейсу.

AI, ШІ — штучний інтелект

ВСТУП

У інформаційному суспільстві, де інформаційні технології впливають на всі види діяльності, питання управління ризиками у сфері невиробничих процесів набуває особливого значення. Невиробничі процеси включають всі види операцій, наприклад, в управлінні, освіті, медицині, фінансах і т.д., які іноді здаються недостатньо зрозумілими.

Не минає й дня, щоб організації не стикалися з викликами, які впливають на їхню здатність бути ефективними, захищати свій персонал, забезпечувати безпеку даних і дотримуватися законодавчих вимог. Оскільки це дуже складно виміряти і на це впливає так багато факторів, традиційні підходи є недостатньо гнучкими або вимагають занадто багато годин ручної роботи.

З огляду на це, існує потреба в автоматизованих рішеннях, які можуть оцінювати рівень ризику в будь-якій сфері швидко, зручно і ефективно, без суворого навчання користувачів, яке вимагалося раніше. Використання методів штучного інтелекту, таких як лінгвістичні моделі, представляє нову парадигму для інтерпретації текстових даних, виведення висновків з описів ситуацій і узагальнення на велику кількість областей.

Цінність таких систем особливо висока в умовах обмежених ресурсів - коли організація не може дозволити собі утримувати команду з управління ризиками на повну ставку. Тому доцільно і реально розробити комп'ютерну систему, яка може автоматично оцінювати рівень ризику на основі опису процесу.

Метою цієї статті є реалізація комп'ютерної системи оцінки ризиків у невиробничих процесах.

1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Формулювання проблеми.

У сучасному динамічному цифровому світі організації стикаються зі зростаючою загрозою не лише з боку виробництва, але й з боку так званих непродуктивних або невиробничих процесів. До них відносяться адміністрування, обробка документів, освіта, управління, медичне та фінансове управління, управління персоналом та багато інших видів обміну інформацією та комунікації.

Невиробничі процеси, як правило, характеризуються високим ступенем мінливості, залежністю від людського фактору, відсутністю структури та слабшими межами контролю, що ускладнює ідентифікацію, прогнозування та контроль ризиків. На відміну від ситуації на виробництві, де ризики можуть бути ідентифіковані на основі чітко визначеного набору стандартних питань, критеріїв безпеки, нормативних актів і технічних інструкцій, ризики в невиробничому контексті часто проявляються неочевидним чином: Погані рішення, недотримання процедур, загрози інформації, соціальна напруженість, управління змінами тощо. Ризики, як правило, не помітні на перший погляд.

За таких обставин деякі традиційні підходи до оцінки ризиків досягають своєї межі:

- Вони вимагають глибоких знань і залучення фахівців у кожній галузі

- Вони повільні, якщо виконуються вручну

- Їх нелегко адаптувати до конкретних умов компанії або процесу

- Вони не дозволяють швидко обробляти вільні текстові дані, надані користувачами системі

Більше того, малі та середні підприємства, некомерційні організації, школи, заклади охорони здоров'я тощо, як правило, не мають у штаті спеціаліста з управління ризиками, що тільки погіршує ситуацію. У таких умовах завжди є потреба в

інструменті, який може швидко оцінити ризик на основі введеного тексту або опису ситуації, не вимагаючи від користувача ніяких спеціальних знань.

Останні розробки в галузі лінгвістичних моделей і нейронних мереж, такі як GPT (Generative Pre-trained Transformer), дозволяють автоматично опрацьовувати і аналізувати текстові описи процесів і генерувати оцінки ризиків. Ці моделі можуть враховувати ситуацію, робити логічні висновки і навіть пропонувати рішення, щоб зменшити ризик, чого не може зробити звичайна система.

Отже, проблема полягає у відсутності додатку, який би дозволяв просто і доступно оцінювати ризики, пов'язані з непродуктивними процесами:

- Можна використовувати з неструктурованими даними

- Не потребує навчання та кваліфікованого користувача

- Підходить для багатьох видів діяльності

- Пропонує точність і зручність

Такий веб-інструмент має особливе значення для користувача. Ця система демократизує аналіз ризиків, зменшить навантаження на експертів і підвищить стандарти безпеки та відповідності в усьому світі.

1.2 Мета та завдання проєкту.

Мета проєкту

Метою цієї роботи є створення інтелектуальної комп'ютерної системи оцінки ризиків для невиробничих процесів. Система використовує, як традиційні ручні методи розрахунку готовності, так і сучасні методи, включаючи аналіз текстових описів з використанням моделей штучного інтелекту для обробки текстів. Система повинна мати зручний веб-інтерфейс, дозволяти вибирати режими, бути доступною для широкого спектру користувачів і використовуватися в різних сферах, включаючи

освіту, медичне обслуговування, адміністративний контроль, інформаційні технології та інші організації, що займаються нематеріальною діяльністю.

Метою цієї ініціативи є подолання розриву між застарілими ручними методами управління ризиками, які потребують фахівців, і наявністю комерційних корпоративних систем, які є фінансово або технічно недоступними для невеликих установ, компаній або індивідуальних користувачів. Потрібно створити гібридну систему, яка поєднує ручні та автоматизовані підходи для забезпечення гнучкості та масштабованості в майбутньому.

Завдання проєкту

План роботи для досягнення поставленої мети проекту повинен включати наступні кроки:

1. Аналіз поняття ризику: його види та класифікація по відношенню до невиробничих процесів, а також представлення сучасних методів аналізу та оцінки ризиків.

2. Аналіз програмного забезпечення, доступного для управління ризиками, показ його можливостей, переваг та недоліків, а також визначення того, чи є запропонованне програмне забезпечення виправданим з точки зору вимог

3. Проектування архітектури системи: архітектура системи проектується шляхом розробки двох функціональних модулів:

4. Ручний режим: калькулятор ризиків розраховується за формулою $R = P \times S \times E$, де P = ймовірність, S = серйозність, E = частота впливу.

5. Інтелектуальний режим: автоматичний аналіз текстового опису ситуації, генерація оцінки ризиків та рекомендації для зниження ризиків, з використанням мовної моделі GPT.

6. Вибір технологічного стеку: пояснення щодо вибору технологічного стеку для розробки (мова програмування, фреймворки, база даних та API).

7. Розробка на стороні клієнта: розробка додатків з використанням React та бібліотек компонентів TypeScript і Material UI для забезпечення зручного та адаптивного інтерфейсу.

8. Розробка бекенд системи: написати серверну частину на основі фреймворку NestJS, яка має обробляти запити, зберігати дані в MongoDB, авторизувати користувачів та підключатися до OpenAI

9. Забезпечити два режими оцінки ризиків: можливість вільного перемикання між режимами та зручна підказка з текстовим виводом.

10. Зберігання та відображення історії оцінювання: системи повинна мати можливість зберігати та відображати історію оцінювання користувача. Наприклад: час запиту, введення, обраний режим оцінювання та результат.

11. Тестування системи: на цьому етапі потрібно протестувати точність за допомогою ручних розрахунків, ефективність за допомогою GPT-обчислень та стійкість до помилок і збоїв за допомогою API.

1.3 Функціональні вимоги до системи.

Для того, щоб впровадити систему оцінки ризиків для невиробничих процесів, необхідно визначити чіткий перелік функцій для досягнення поставлених цілей. Розроблене програмне забезпечення має дві основні особливості: реалізація двох режимів оцінки ризиків (ручного та інтелектуального), які задовольняють один одного, та підтримка всього життєвого циклу користувача, від реєстрації до отримання аналітичного висновку.

1.3.1 Авторизація користувача.

Аутентифікація здійснюється за допомогою електронної пошти та пароля. Доступ до опцій аналізу та історії запитів повинена бути обмежена для не зареєстрованих користувачів. Аутентифікація (JWT - JSON Web Token) також повинна бути інтегрована в систему.

1.3.2 Режими оцінки ризиків.

1.3.2.1 Ручний режим.

У ручному режимі користувач має можливість вести три числових значення:

- P (ймовірність виникнення ризику) — шкала 1 до 5;
- S (Серйозність негативного впливу) — шкала від 1 до 5;
- E (Частота впливу) — шкала від 1 до 5.

Ризик оцінюється за формулою:

$$R = P \times S \times E,$$

де R є кінцевой оцінкою. Ступінь рівня оцінки визначається системою на основі значення:

Значення R	Рівень ризику
1–25	Низький
26 – 50	Середній
51-75	Високий

76 і вище	Критичний

Ручний режим вирахування оцінки ризиків є корисним для користувачів, які мають специфічні галузеві знання про процедуру і потребують швидкого результату.

1.3.2.2 Інтелектуальний режим (на базі OPENAi).

У цьому режимі користувач вводить короткий опис процедури в текстовому вигляді. Інтерфейс має поля для:

- Галузь (наприклад, освіта, будівництво, IT)
- Назва процесу
- Опис процесу
- Відомі ризики для користувача

На основі введених даних система надсилає запит до мовної моделі OPENAi, яка виконує наступні завдання:

- Опрацьовує та аналізує ситуацію відповідно до логіки
- Визначає рівень ризику (низький, середній, високий, критичний)
- Складає обґрунтування прийнятого рішення
- Надає рекомендацій щодо зменшення виявлених ризиків

Таким чином, система функціонує як експертна система, що імітує роботу фахівця з управління ризиками.

1.3.4 Історія запитів.

Усі оцінки від зареєстрованого користувача повинні бути збережені у приватній історії. Слід відмітити наступне для кожного запиту потрібно зберегти таку інформацію:

– Дата і час перегляду
– Вибір режиму (ручний/інтелектуальний)
– Вхідні дані
– Результат оцінки
– Пояснення або пояснення AI

Ця функція дозволить користувачу контролювати, зберігати оцінку, і повторно запускати попередні процеси.

1.3.5 Додаткові вимоги.

Елементи інтерфейсу користувача, побудованого за допомогою бібліотеки компонентів Material UI, повинні бути прості та максимально зручні для користувачів з різними цифровими навичками.

Дані зберігаються в базі даних MongoDB, з урахуванням взаємозв'язків між користувачем, рейтингами та вхідними даними. Обробка запитів OPENAI є надійною, оскільки впроваджена система управління помилками, частота запитів обмежена, а введені дані перевіряються.

Введені дані повинні перевірятися як на стороні клієнта (перевірка формату, обов'язкових полів), так і на стороні сервера (тип, валідація тощо).

1.4 Огляд існуючих систем.

Наразі існує близько десятка систем, які претендують на вирішення проблеми управління ризиками в компаніях за допомогою програмного забезпечення. Найпоширеніші з них - RiskWatch, LogicManager і Resolver. Вони широко використовуються у великих компаніях для виявлення та оцінки ризиків, а також для забезпечення належного моніторингу ризиків у часі.

RiskWatch - це середовище для аналізу ризиків, аудиту. Ця система має багато інструментів для створення власних шаблонів оцінки ризиків і зберігає кастомні файли для сотень журналів і дій.

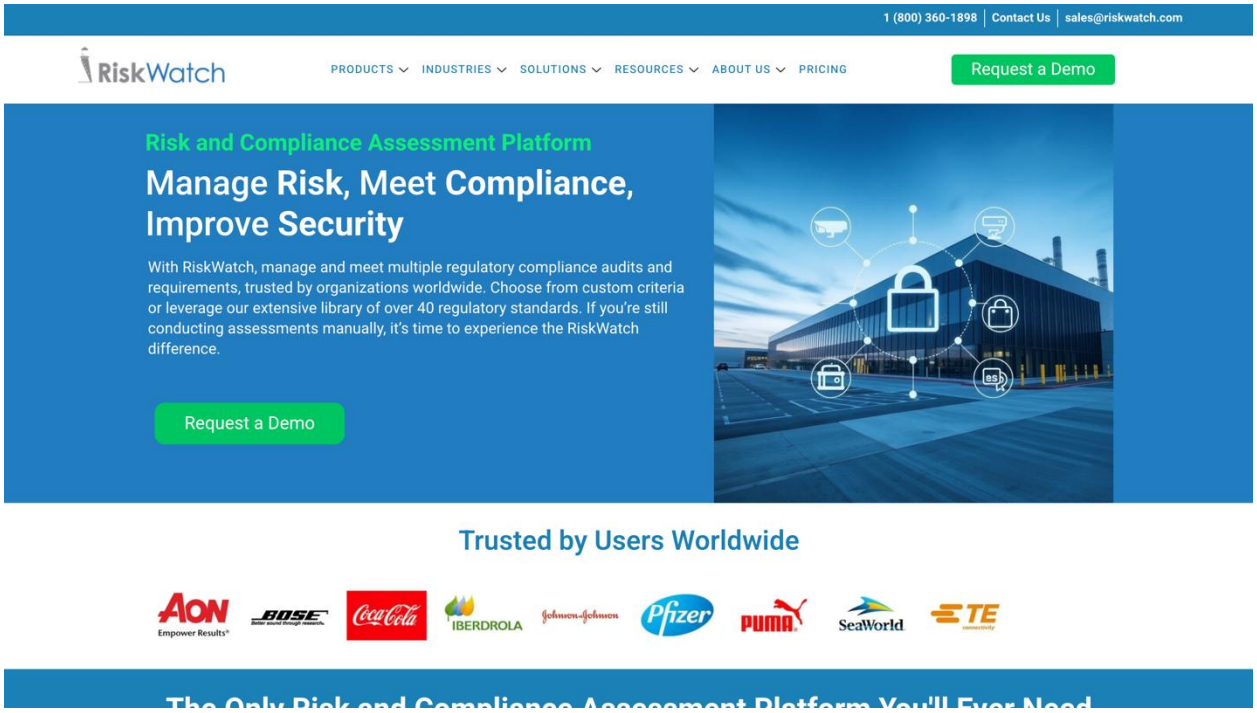

Рисунок 1.1 – Інтерфейс RiskWatch

Переваги RiskWatch:

- Налаштування шаблонів та методів оцінки ризиків за бажанням
- Відповідність стандартам безпеки
- Інтеграція з іншими бізнес-системами
- Підтримка повноцінного звітування та аналізу

LogicManager - це програмне забезпечення для управління корпоративними ризиками, яке може допомогти організаціям приймати рішення щодо ризиків в рамках своїх політик, планувати ризики та ефективно впроваджувати ці політики. Система включає в себе інструменти для внутрішнього аудиту, відстеження інцидентів та аналізу впливу.

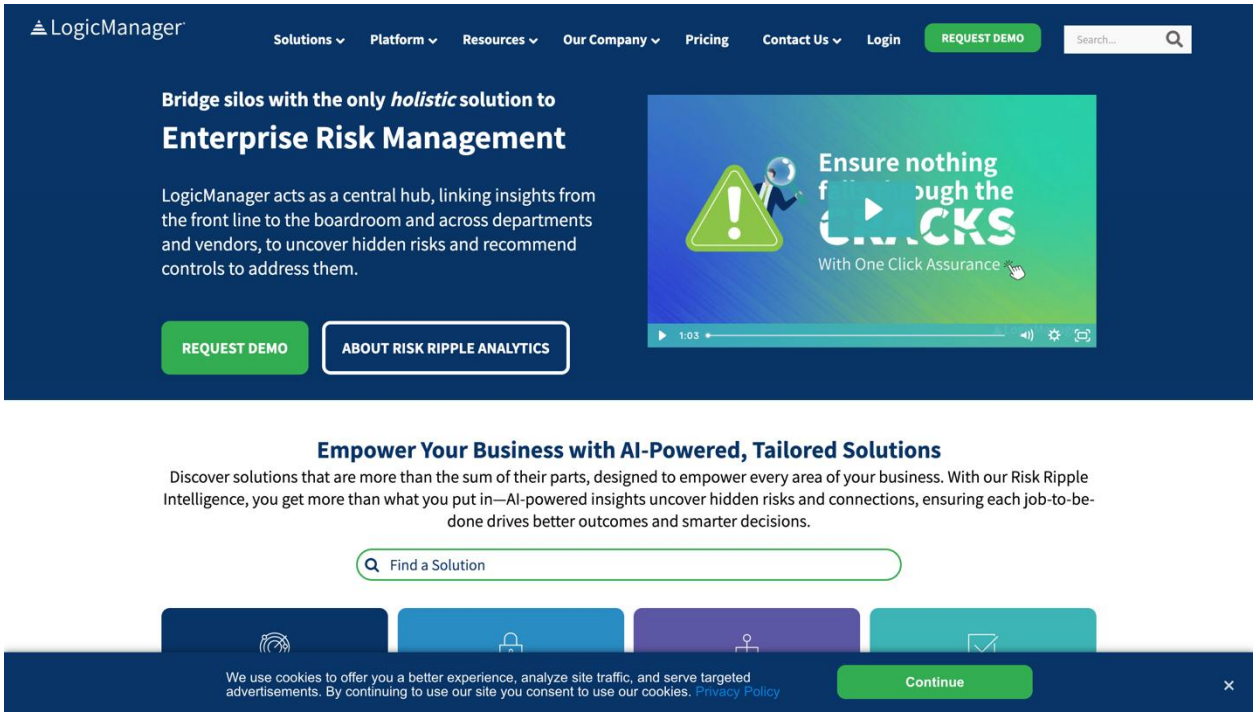

Рисунок 1.2 – Інтерфейс LogicManager

Переваги LogicManager:

- Містить зручний дашборд із готовими шаблонами
- Допомагає галузям дотримуватися стандартних практик
- Інтеграція з різними системами
- Масштабується для великих підприємств

Resolver - це хмарна система управління ризиками, яка зосереджена на централізованому управлінні ризиками, управлінні інцидентами та внутрішньому контролі. За допомогою Resolver ви можете створювати індивідуальні сценарії управління ризиками та мати гнучкі та потужні інструменти для їх аналізу.

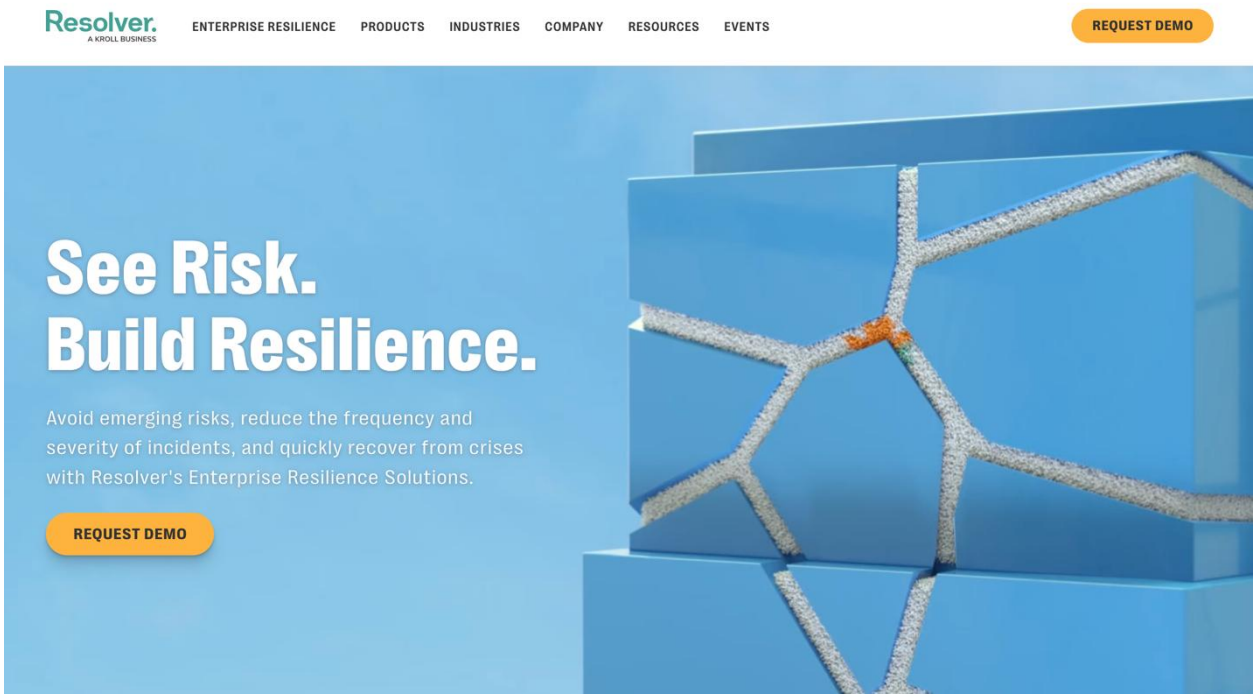

Рисунок 1.3 – Інтерфейс Resolver

Переваги Resolver:

- Потужна платформа для обробки інцидентів і ризиків;
- Прив'язка до ролей персоналу та контроль змін;
- Модулі для аналізу численних сценаріїв;
- Інтеграція з іншими бізнес-системами.

Незважаючи на свої розширені функціональні можливості, всі ці системи мають прогалини в певних загальних функціях. Недоліки таких систем:

- Конфігурація є складною: більшості систем потрібен тривалий час налаштування перед тим, як вони можуть бути використані у виробничій роботі
- Середньостатистичні користувачі знаходять інтерфейс складним

- Висока вартість ліцензій та обслуговування, орієнтована на корпоративний сектор
- Відсутність підтримки аналізу тексту чи інтеграції ШІ у процеси обробки текстових описів
- Для ринку малих та середніх підприємств (або для індивідуальних користувачів) ці системи мають обмеження

Аналіз існуючих рішень показує, що існує нагальна потреба в простих, доступних системах, які дозволяють швидко, як на друкарській машинці, проводити оцінку ризиків у текстовому вигляді, коли завгодно і в будь-який час. Звідси випливає нагальна потреба в розробці нової системи, яка використовує лінгвістичні моделі, що наразі застосовуються для автоматизованого аналізу ризиків у невиробничих процесах.

2 ПРОЕКТУВАННЯ ВЕБЗАСТОСУНКА

2.1 Архітектура системи.

Веб-додаток для оцінки ризиків, пов'язаних з невиробничими процесами, повинен гарантувати стабільність, масштабованість та інтеграцію з зовнішніми сервісами (в тому числі API штучного інтелекту).

Модель клієнт-серверної архітектури складається з трьох основних рівнів:

Frontend — клієнтська частина:

- Користувацький інтерфейс
- Обробка API
- Велика частина роботи виконується на клієнтському пристрої.

Backend — серверна частина:

- Обробка бізнес-логіки
- Маршрутизація

– Авторизація
– Взаємодія з базою даних та OpenAI API

База даних

– Сховище даних користувача
– Історія запитів
– Результати оцінки ризиків

Крім того, додаток може працювати з OpenAI API, який виступає в ролі інтелектуального аналізатора текстових описів.

Структура загальної архітектури:

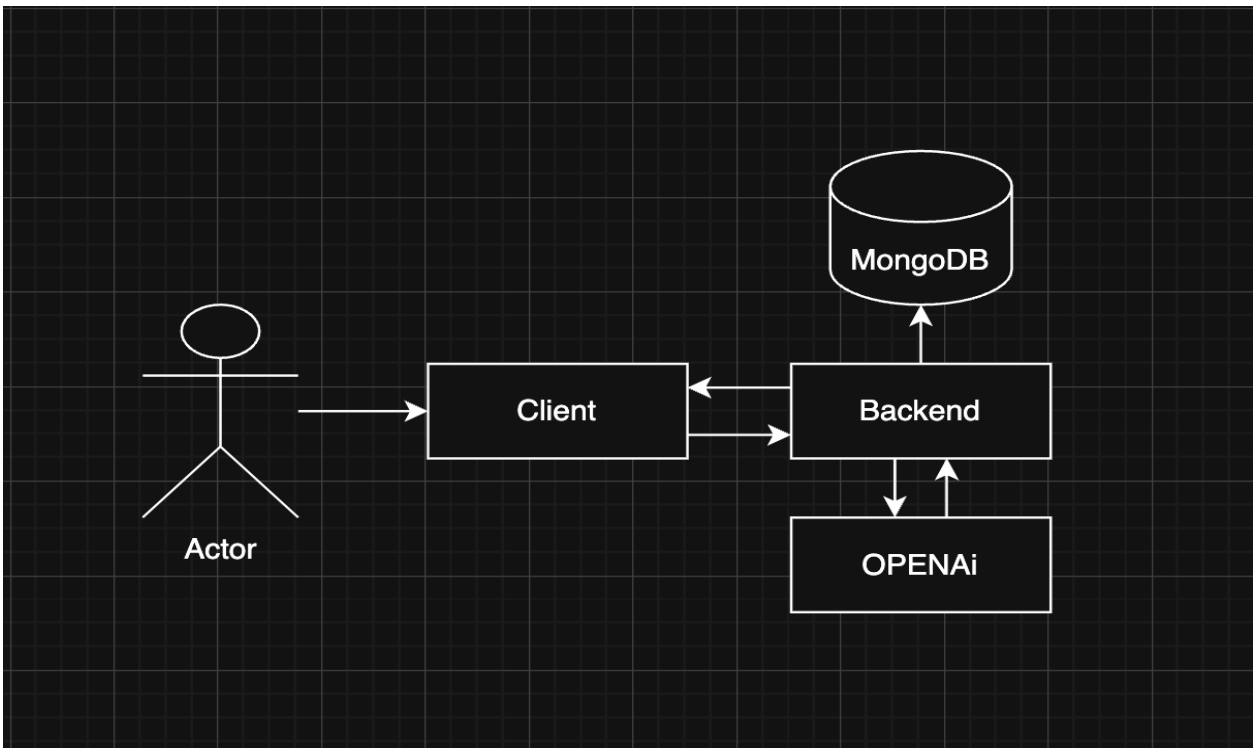

Рисунок 2.1 – Загальна архітектура вебзастосунка для оцінювання ризиків

Програмне забезпечення розроблено на основі концепції поділу завдань. Кожен компонент виконує свою роль, а система легко розробляється, тестується і розширюється.

Взаємодія:

- Інтерфейс управляється користувачем на фронтенді.
- Введені користувачем дані відправляються на сервер через HTTP-запит.

Backend:

- Здійснює обробку даних у ручному режимі або за допомогою AI;
- Відправляє запит на інтелектуальний аналіз до OpenAI API;
- Взаємодіє з MongoDB для збереження історії;
- Відправляє оброблені дані назад на сторону клієнта.

Ключові компоненти системи

2.1.1 Користувач (Браузер).

Додаток доступний для користувача за допомогою веб-браузера. Взаємодія відбувається через інтерфейс, створений за допомогою React та Material UI.

Користувач може:

- Авторизуватися в системі
- Вибрати режим оцінки (ручний або ші)
- Ввести дані (числові або текстові)
- Переглядати результати
- Ознайомлюватися з попередніми оцінками в особистій історії

2.1.2. Клієнтська частина (Frontend).

Фронтенд на JavaScript побудований з використанням React і TypeScript для забезпечення типізації та організації інтерфейсу в компонентний спосіб.

За допомогою Material UI легко створити адаптивний інтерфейс, який відповідає сучасним стандартам UX/UI.

Основні задачі фронтенду:

– Відображення форм введення
– Валідація введених даних
– Маршрутизація (React Router)
– Відправка запитів на бекенд
– Отримання запитів і відображення результатів

2.1.3 Серверна частина (Backend).

Бекенд написаний на NestJS — архітектурному фреймворку Node.js, заснованому на принципі контролер-сервіс-модуль.

Відповідає за:

– Отримання та обробку HTTP-запитів
– Виконання оцінки в ручному режимі
– Побудову запитів до API OpenAI
– Роботу з авторизацією користувачів (JWT)
– Працю з MongoDB за допомогою бібліотеки Mongoose

2.1.4 База даних (MongoDB).

MongoDB використовується для зберігання:

– Облікових записів користувачів
– Інформації про вид оцінки ризику, дату та результати
– Історії пошуку всіх запитів, пов'язаних з користувачем

Документно-орієнтована архітектура бази даних дозволяє гнучко масштабувати систему, додаючи нові типи даних в майбутньому.

2.1.5 API OpenAI.

Модель машинного навчання OpenAI використовується для інтелектуального аналізу тексту, наданого користувачем. Зокрема, сервер передає запит до GPT, яке оцінює сценарій.

Деякі з результатів моделі:

- Категорія ризику (низький, середній, високий, дуже високий)
- Опис оцінки
- Пропозиції щодо зменшення ризику

Таким чином, система може виконувати автоматичну експертну оцінку без участі спеціаліста.

Сценарії взаємодії

1. Ручний режим.

- Користувач заповнює форму (значення P, S, E), дані надсилаються на бекенд.
- Сервер обчислює результат ($R = P \times S \times E$) та визначає рівень ризику на основі обчислення.
- Відповідь зберігається в базі даних і відправляється назад клієнту.

2. Інтелектуальний режим.

- Користувач надає назву, опис процесу, за бажанням поле.
- Потім створює запит і відправляє його в OpenAI.
- Відповідь з поясненням і категорією ризику зберігається в базі даних.
- Результат відображається на екрані (див. результат для кінцевого користувача).

Переваги архітектури:

- Розділення логіки за рівнями (інтерфейс, логіка, дані);
- Висока гнучкість і можливість масштабування;
- Зручна інтеграція із зовнішніми API;
- Можливість повторного використання компонентів;

– Безпечна авторизація та контроль доступу.

2.2 Обґрунтування вибору технологічного стеку.

При розробці сучасного веб-додатку потрібно відповідально ставитися до технологічного стеку, оскільки від цього залежить, наскільки швидко додаток можна буде реалізувати, наскільки легко його підтримувати і наскільки він буде масштабованим і зручним для користувача.

Враховючи вимоги, перераховані вище - система повинна підтримувати клієнт-серверні системи, легко інтегруватися зі сторонніми API та веб-інтерфейсом, зберігати та аутентифікувати дані користувачів та мати зручний інтерфейс - я зупинив свій вибір на дещо модифікованому стеку MERN, замінивши Express на NestJS. Основними елементами стеку є MongoDB, NestJS, React, TypeScript.

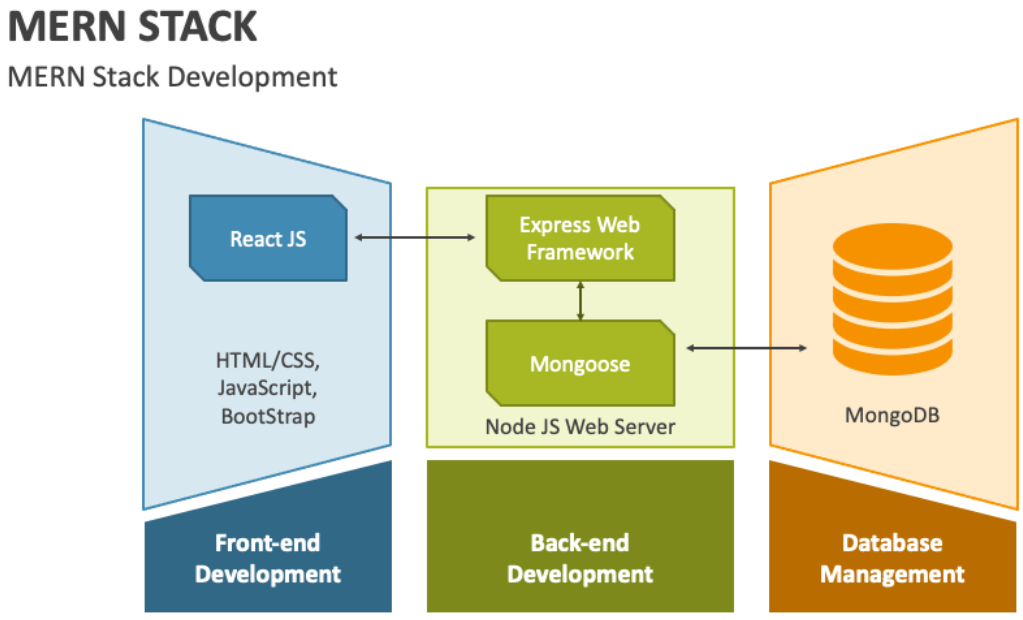

Рисунок 2.2 – Етапи розробки стеку MERN

MongoDB

MongoDB - це база даних документів, розроблена для спрощення розробки та масштабування. Вона підходить, якщо взаємодія між даними не повинна бути дуже тісною і якщо у вас динамічна структура даних. Одиницею зберігання в MongoDB є «колекція», в якій зберігаються дані:

- облікові записи користувачів (електронна пошта, хешований пароль, id);
- результати оцінки ризиків (режим, вхідні параметри, дата, результат)
- повний журнал усіх взаємодій користувача з системою

MongoDB забезпечує:

- можливість масштабування (шляхом горизонтального масштабування)
- швидкий доступ до документів
- легке об'єднання з NestJS, використовуючи `@nestjs/mongoose`

NestJS

NestJS - це серверний фреймворк, який дуже схожий на Angular.js своєю модульною архітектурою, декораторами, інверсією залежностей та сильною типізацією з підтримкою TypeScript. NestJS схожий на Express, але має свої переваги.

Переваги NestJS:

- модульність: дуже легко розбити логіку на самостійні функціональні блоки (автентифікація, вирахування ризиків, користувач)
- підтримка REST API
- інтеграція з JWT, необхідна для безпечної авторизації
- легкий доступ до сторонніх API через сервіси (наприклад, взаємодія з OpenAI)
- валідація, з підтримкою бібліотек class-validator та class-transformer.

У цьому проєкті я використовував NestJS для:

- обробки запитів від клієнта

– роботи з OpenAI API
– збереження/читання даних у MongoDB;
– авторизації користувачів (JWT).

React + TypeScript

React - одна з найпопулярніших бібліотек для користувацьких інтерфейсів. Завдяки її компонентній структурі ми можемо легко створювати багаторазові частини користувацького інтерфейсу. У поєднанні з TypeScript, React можна використовувати для розробки надійних інтерфейсних додатків.

Переваги React:

– швидкий рендеринг завдяки віртуальному DOM
– розробка на основі багаторазових компонентів
– велика кількість готових до використання бібліотек (Formik, Axios, React Router)

Переваги TypeScript:

– статична типізація, що дозволяє зменшити кількість помилок під час компіляції
– краща підтримка IDE (автодоповнення, рефакторинг)
– вдосконалена документація API через описи типів.

У цьому додатку React відповідає за створення:

– форм введення ризиків (вручну та у режимі AI)
– інтерфейсу взаємодії для перегляду результатів
– історії оцінок з можливістю фільтрації
– валідації на рівні клієнта

Material UI (MUI)

Material UI (MUI) є бібліотекою компонентів React. Вона дозволяє створювати сучасні гнучкі, адаптивні та візуально єдині додатки.

Причини для вибору MUI:

- готові елементи (кнопки, поля вводу, таблиці, модальні вікна)
- налаштування вигляду інтерфейсу
- гнучкість у стилізації
- гарна документація та чудова підтримка TypeScript

MUI обрано для розробки користувацького інтерфейсу, де MUI побудовано з акцентом на простоту використання, логіку та доступність.

OpenAI API

OpenAI API лежить в основі модуля штучного інтелекту системи. Інтеграція моделі GPT додає в систему будь-який текстовий опис процесу, а користувачі також можуть отримати класифікацію ризиків, пояснення логіки оцінки та рекомендації щодо зменшення ризиків.

OpenAI API інтегрований на рівні сервера (NestJS):

- створення запиту відповідно до даних користувача
- формування запиту до OpenAI
- опрацювання відповіді

Перевага цього методу полягає в тому, що ця система може робити контекстний аналіз, що робить його на крок ближче до діагностики експерта.

Висновок:
Обраний стек технологій ідеально відповідає потребам системи - надійна, масштабна система з простими в розробці декларативними інтерфейсами, сучасним користувацьким інтерфейсом, API інтеграцією та безпечною обробкою даних.

2.3 Інтерфейсу користувача.

Фронтенд є важливою частиною веб-додатків, адже це те, з чим взаємодіє користувач. Клієнтська частина цього проєкту була реалізована з використанням бібліотеки React і мови програмування TypeScript, щоб забезпечити адаптивність, покращуючи підтримку коду (разом із високою продуктивністю). Крім того, також використовується MUI (набір компонентів Material UI), який спеціально розроблений для інтерфейсів платформи.

2.3.1 Структура фронтенду.

Проєкт використовує компонентно-орієнтований підхід, чіткий інтерфейс розділений на логічні, незалежні частини, які легко тестуються, повторно використовуються та підтримуються. Нижче наведена базова структура:

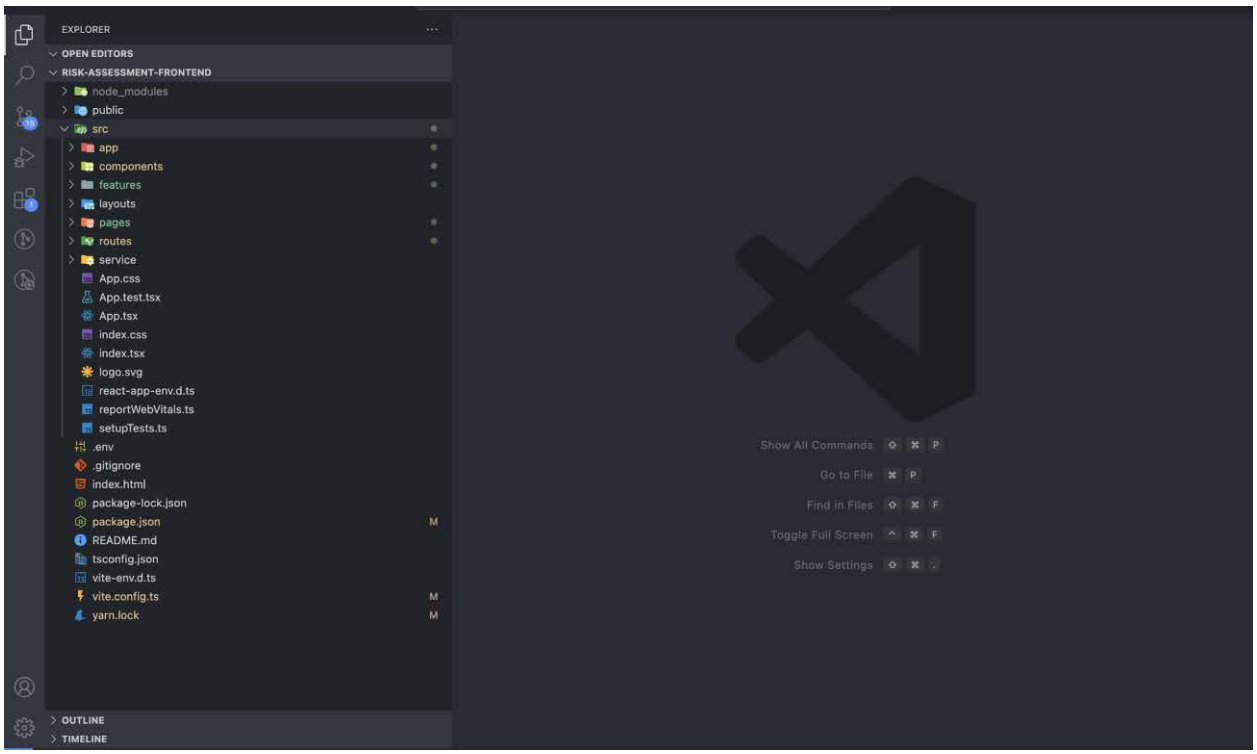

Рисунок 2.3 – структура фронтенду

Такий підхід забезпечує ізоляцію бізнес-логіки, підвищує зрозумілість коду та дозволяє легко масштабувати систему.

2.3.2 Мова програмування: TypeScript.

TypeScript використовується для:

- типізації вхідних даних до компонента
- типізації вихідних даних від бекенда
- виявлинння помилок на раніх фазах розробки

Приклад використання типів:

```
interface RiskFormInputs {

     industry: string;
     processName: string;
     processDescription: string;
     riskKnownByUser: string;

}

type RegisterFormInputs = {

     email: string;
     password: string;

};
```

2.4 Реалізація основних екранів та форм.

2.4.1 Основна форма оцінки ризиків.

- У ручному режимі використовується форма з трьома числовими полями: P, S, E.
- В інтелектуальному режимі використовуються текстові поля: Галузь, Назва процесу, Опис процесу, Відомі ризики (необов'язково.
- Валідація здійснюється за допомогою Yup у поєднанні з Formik.
- Модуль є адаптивним: він коректно відображається на десктопах, планшетах і смартфонах.

2.4.2 Відображення результату.

– Після позитивного результату оцінки користувач бачить
– рівень ризику (у вигляді кольорового індикатора)
– пояснення (абзац з текстом)
– рекомендації щодо мінімізації ризику (тільки в режимі ШІ)
– кнопку для збереження або повернення до оцінки.

2.4.3 Сторінка історії оцінювання.

Користувач може переглянути список всіх попередніх оцінок, які він проводив. Таблиця містить

– дата вирахування ризику
– за яким сценарієм був оцінений ризик
– короткий опис
– рівень ризику

2.4.4 Інтерфейс користувача: Material UI.

Material UI використовується для:

– Створення інтуїтивно зрозумілих форм
– Реалізації адаптивних макетів
– Налаштування тем (світлі/темні)
– Сповіщення (алерти, панелі перекусів)
– Спінерів, вкладок, акордеонів

MUI пропонує візуальну цілісність, підтримку доступності (a11y), кастомізацію стилів за допомогою sx API та швидкий запуск проекту за допомогою попередньо зібраних компонентів.

2.4.5 Навігація: React-router.

Бібліотека react-router-dom використовується для реалізації навігації по SPA:

Маршут	Сторінка	Призначення
/login	LoginPage	Сторінка входу
/register	RegisterPage	Сторінка реєстрації
/ai-assessment	AiAssessmentPage	Сторінка оцінки ризиків за допомогою штучного інтелекту
/calculator	CalculatorPage	Сторінка оцінки ризиків за допомогою ручного режиму
/history	HistoryPage	Сторінка історії оцінки ризиків
/	HomePage	Головна сторінка

2.4.6 робота з API: Axios + useEffect / Redux.

Запити на стороні сервера реалізуються за допомогою бібліотеки Axios з попередньо налаштованим інтерпретатором для додавання JWT-токену до заголовків:

```
axios.interceptors.request.use((config) => {
  const token = localStorage.getItem('token')
```

```
  if (token) {
    config.headers.Authorization = `Bearer ${token}`
  }
  return config
})
```

Хуки React (useEffect, useState) можна використовувати для реалізації асинхронної логіки та оптимізації модуля, зберігання відповідей, обробки станів навантаження та помилок.

2.4.7 UX рішення.

Щоб забезпечити високу якість користувацького досвіду, було реалізовано наступне

- адаптивна валідація модуля
- зрозумілі повідомлення про помилки (змістовний текст, кольорове виділення)
- індикатори успішності/неуспішності оцінювання
- зручну логіку навігації

Висновок:

Фронтенд частина веб-додатку була реалізована відповідно до сучасних принципів розробки інтерфейсів. Поєднання React, TypeScript та Material UI дозволило створити інтуїтивно зрозумілий, кастомізований та легко масштабований інтерфейс, який дозволяє користувачеві комфортно взаємодіяти з інтелектуальною системою оцінки ризиків.

2.5 Внутрішня частина веб-додатку.

Бекенд - це центральна частина системи, яка обробляє запити користувачів, виконує бізнес-логіку, взаємодіє з базою даних, надає авторизацію та інтегрується із зовнішніми сервісами, включаючи OpenAI API. У цьому проекті серверна частина була реалізована за допомогою NestJS - сучасного фреймворку для Node.js,

заснованого на принципах інверсії залежностей, модульності та строгої типізації за допомогою TypeScript.

2.5.1 Архітектура додатку на NestJS.

Додаток використовує стандартну структуру NestJS:

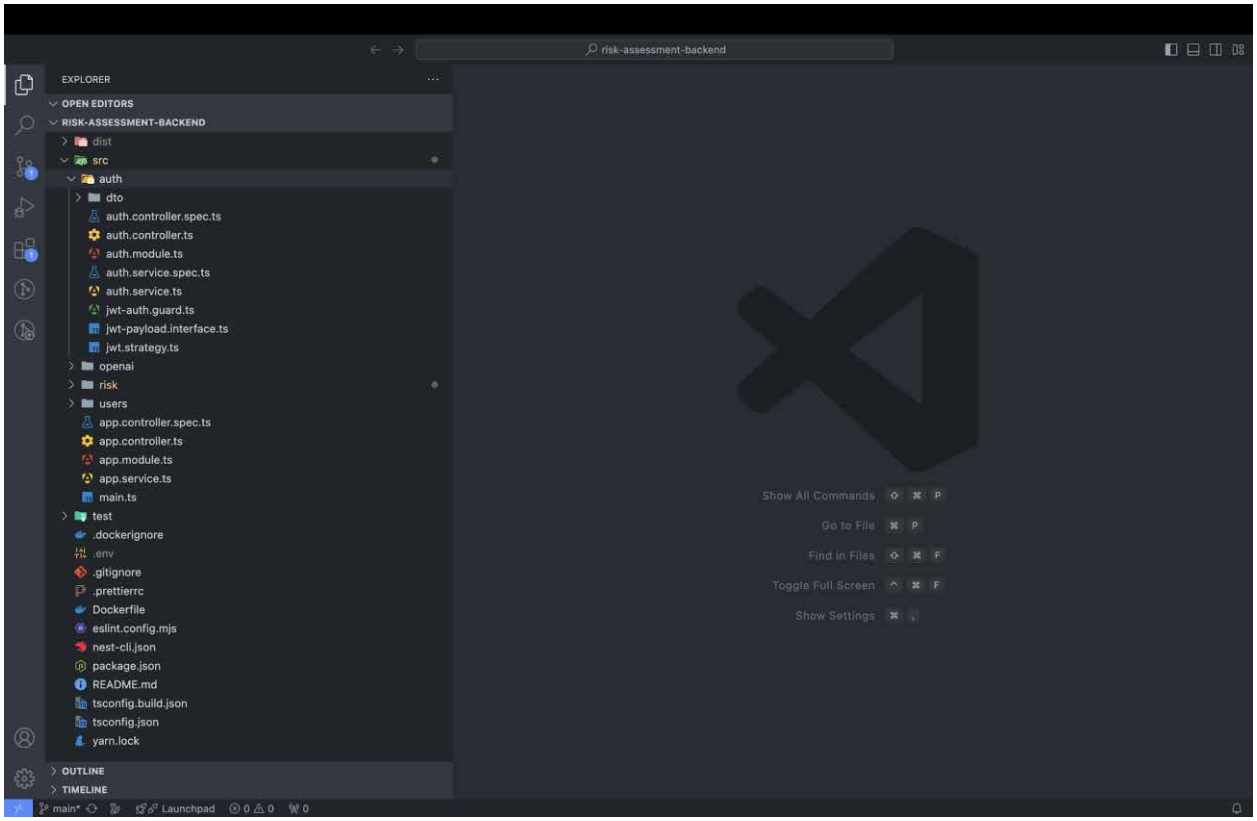

Рисунок 2.4 – структура бекенду

Кожен модуль містить:

- Контролер - відповідає за обробку HTTP-запитів;
- Service - містить бізнес-логіку;
- DTO - об'єкти для типізації даних (Data Transfer Object);
- Schema - для MongoDB (через Mongoose).

2.5.2 Реалізація REST API.

У системі реалізовано окремі контролери:

- AuthController - /api/auth:
  - POST /login - вхід користувача;
  - POST /register – реєстрація
- RiskManualController - /api/risk/manual:
  - POST - приймає значення P, S, E, обчислює R = P × S × E, класифікує ризик, зберігає його в базі даних, повертає результат.
- RiskAiController - /api/risk/ai:
  - POST - отримує опис процесу, створює запит, надсилає запит до OpenAI, отримує оцінку ризику та рекомендації, зберігає, повертає результат.
- HistoryController - /api/history:
  - GET - повертає оцінку поточного користувача

2.5.3 DTO та валідація.

Валідація запитів виконується за допомогою класу-валідатора та класу-трансформатора. Наприклад:

```
import { IsArray, IsNotEmpty, IsString } from 'class-validator';

export class AssessRiskDto {

  @IsString()

  @IsNotEmpty()

  industry: string;

  @IsString()

  @IsNotEmpty()

  processName: string;

  @IsString()
```

```
  @IsNotEmpty()
  processDescription: string;
  @IsArray()
  @IsString({ each: true })
  risksKnownByUser: string[];
}
```

Це дозволяє захистити бекенд від некоректних даних і дотримуватися безпеки типів.

2.5.4 Авторизація та захист.

Авторизація здійснюється за допомогою JWT

- При вході в систему користувач отримує токен доступу
- Цей токен надсилається до захищених маршрутів у заголовку авторизації
- Маршрути захищені NestJS AuthGuard

```
@UseGuards(JwtAuthGuard)
 @Get('history')
 getUserRiskHistory(@Request() req) {
    return this.riskService.getHistoryForUser(req.user.userId);
 }
```

2.5.5 Інтеграція з MongoDB.

NestJS використовує пакет `@nestjs/mongoose` для підключення до MongoDB. Структура схеми RiskRecord (узагальнена для обох сценаріїв):

```
import { Schema, Prop, SchemaFactory } from '@nestjs/mongoose';
import { Document, Types } from 'mongoose';

@Schema({ timestamps: true })
```

```
export class RiskHistory extends Document {

  @Prop({ required: true })

  user: string;



  @Prop({ required: true })

  industry: string;



  @Prop({ required: true })

  processName: string;



  @Prop({ required: true })

  processDescription: string;



  @Prop({ type: [String], default: [] })

  risksKnownByUser: string[];



  @Prop({ required: true })

  response: string;

}

export const RiskHistorySchema =
SchemaFactory.createForClass(RiskHistory);
```

2.5.6. інтеграція з OpenAI API.

Інтелектуальний режим використовує модель gpt-3.5 через HTTP-запит до OpenAI:

– запит формується на основі введеного користувачем тексту (назва процесу, опис, область дії тощо) ;
– запит формується динамічно, з урахуванням контексту;
– запит надсилається на кінцеву точку `https://api.openai.com/v1/chat/completions;`
– модель генерує категорію ризику, пояснення та рекомендації;
– результат зберігається в базі даних і повертається користувачеві.

2.5.7. Управління помилками та дроселювання.

Для забезпечення стабільної роботи системи реалізовано:

– виняткову обробку помилок через HttpExceptionFilter ;
– обмеження частоти запитів до OpenAI - через RateLimiter або механізм Throttle (для уникнення перевищення ліміту);
– логування помилок на рівні сервісу та глобального перехоплювача.

Висновок:

Бекенд додатку побудований за принципами кастомної архітектури з розбиттям на модулі, використанням DTO, валідацією, обробкою помилок та інтеграцією зі сторонніми сервісами. За допомогою бібліотеки NestJS вийшло створити гнучку, масштабовану та безпечну серверну частину, яка повністю покриває функціональні вимоги: авторизація, аналіз, зберігання та відповідь користувачам.

2.6 База даних MongoDB: структура, колекції, індекси, зберігання історії.

У запропонованій системі зберігання даних реалізовано на документно-орієнтованій базі даних MongoDB, яка є частиною обраного технологічного стеку. MongoDB дозволяє зберігати інформацію як гнучкі JSON-подібні документи та колекції, що ідеально підходить для програм із динамічними структурами даних і гнучкими вимогами до масштабованості.

2.6.1 Загальна структура бази даних.

У проекті використовуються такі основні колекції.

Колекція	Призначення
User	Зберігання облікових записів користувачів (електронної пошти, пароля, імені користувача).
RiskHistory	Історія оцінки ризиків користувача (в ручному режимі та режимі AI).

2.6.2 Схема користувача.

```
@Schema()
export class User {
  @Prop({ required: true, unique: true })
  email: string;

  @Prop({ required: true })
  passwordHash: string;

  @Prop({ default: Date.now })
  createdAt: Date;
}
```

- Паролі не зберігаються у вигляді звичайного тексту, а хешуються (за допомогою bcrypt).
- Унікальний індекс в полі електронної пошти гарантує унікальність акаунтів.

2.6.3. Схема оцінки ризиків.

Ця колекція є важливою в контексті бізнес-логіки, оскільки вона зберігає всі оцінки користувача незалежно від режиму.

```
import { Schema, Prop, SchemaFactory } from '@nestjs/mongoose';
import { Document, Types } from 'mongoose';
```

```
@Schema({ timestamps: true })
export class RiskHistory extends Document {
  @Prop({ required: true })
  user: string;

  @Prop({ required: true })
  industry: string;

  @Prop({ required: true })
  processName: string;

  @Prop({ required: true })
  processDescription: string;

  @Prop({ type: [String], default: [] })
  risksKnownByUser: string[];

  @Prop({ required: true })
  response: string;
}

export const RiskHistorySchema =
SchemaFactory.createForClass(RiskHistory);
```

- У поле `user` записується ідентифікатор користувача, який викликав запит
- У полі `industry` вказується галузль
- У полі `processName` вказується назва процеса
- У полі `processDescription` вказується опис процеса
- У полі `risksKnownByUser` вказується інформація яка відома для користувача
- Поле `response` зберігається відповідь

2.6.4 Приклад збереженої інформації в форматі JSON.

```
{
  "user": "642e9f8cde12a71f3c67f90a",
  "processName": "Відправлення електронної пошти з персональними
даними",
  "industry": "Освіта",
  "processDescription ": "Працівники надсилають дані студентів без
шифрування"
```

```
  "response": "
Рівень ризику: Високий
Існує ризик витоку персональної інформації без відповідного захисту.
Рекомендація для зниження ризику: Використовувати шифрування,
налаштувати політики безпеки.
",
"createdAt": "2025-05-26T20:14:35.000Z"
}
```

2.6.5 Індекси та продуктивність.

Для підвищення ефективності відбору з великих масивів даних реалізовано наступні індекси:

Поле	Індекс	Призначення
userId	звичайний	Швидкий пошук в історії конкретного користувача
createdAt	сортування	Створення хронологічного списку

2.6.6 Принцип збереження історії оцінок

Кожна оцінка є окремим документом у записах ризиків, пов’язаним з ідентифікатором користувача. В кінці оцінювання, система виконує наступне:

1. Формує об'єкт оцінки.
2. Зберігає його в базі даних.
3. Повертає підтвердження та результат користувачеві.

Це гарантує:

– повну історію дій користувача;

– можливість подальшого аналізу;

– перегляд і відстеження результатів.

Висновок:

MongoDB — це надійна база даних для реалізації цієї системи з гнучкою структурою та потенційно великим обсягом збережених даних. Завдяки функціям MongoDB — підтримці форматів документів, динамічній схемі, легкій масштабованості та високій продуктивності — можна зберігати історію оцінки ризиків у зручній формі та далі її аналізувати, фільтрувати.

2.7 Інтеграція з OpenAI API.

Однією з головних особливостей створеного веб-додатку є можливість виконувати інтелектуальну оцінку ризиків на основі вільного текстового опису процесу. З цією метою була реалізована інтеграція з OpenAI API, яка надає доступ до мовної моделі GPT-3.5, здатної аналізувати та генерувати логічні та обґрунтовані відповіді.

2.7.1 Мета інтеграції OpenAI.

Інтеграція з OpenAI забезпечує:

- автоматичну інтерпретація текстових описів
- класифікацію рівня ризику
- сформулювання та пояснення результату
- розробка рекомендацій щодо зниження ризиків

Таким чином користувач може отримати експертну оцінку без необхідності вимагати професійного навчання, що особливо корисно для невеликих організацій і непрофесійних користувачів.

2.7.2. Формат запиту API.

Для взаємодії з OpenAI використовується:

`https://api.openai.com/v1/chat/completions`

Формат запиту — JSON:

Модель: gpt-3.5

```
{
  "model": "gpt-3.5",
```

```
  "messages": [
    {
      "role": "system",
      "content": "Ти є експертом з аналізу ризиків. Оціни опис процесу
та поверни рівень ризику, пояснення та рекомендації."
    },
    {
      "role": "user",
      "content": "Назва процесу: Робота з конфіденційними документами.
Опис: Працівники пересилають незашифровані дані через email."
    }
  ],
  "temperature": 0.3,
  "max_tokens": 500
}
```

2.7.3 Формування для промпта для запиту.

Формування промпта для запиту до моделі відбувається динамічно на основі введених даних користувачем.

- Галузь
- Назва процесу
- Опис процесу
- Фактори/ризики які відомі користувачу

Система автоматично складає промпт за шаблоном:

```
“Оціни рівень ризику для невиробничого процесу в галузі
"${dto.industry}":
Назва процесу: ${dto.processName}
Опис процесу: ${dto.processDescription}
Відомі ризики: ${dto.risksKnownByUser?.join(', ') ?? 'немає
інформації'}
Дай відповідь у форматі:
- Рівень ризику: (низький / середній / високий)
```

- Коротка оцінка ризику
- Рекомендації для мінімізації ризику”

2.7.4. Тарифні обмеження до моделі (безпека).

OpenAI має політику обмеження тарифів, яка залежить від моделі та тарифного плану.

Для уникнення помилок при перевищенні ліміту :

- реалізовано обробку помилок 429 (Занадто багато запитів);
- запити обмежуються системою дроселювання (наприклад, максимум 1 запит кожні 2 секунди)
- для кожного користувача фіксується час між запитами.

Управління помилками також включає в себе

- недоступність сервісу (500 Internal Server Error) ;
- неправильний формат запиту;
- таймаут і надлишок токенів.

У разі помилки користувач отримує повідомлення і можливість повторити запит.

2.7.5 Захист конфіденційної інформації.

При відправці запиту до OpenAI :

- всі запити відправляються по протоколу HTTPS;
- не передаються ідентифікаційні дані або конфіденційні дані користувача (наприклад, e-mail)
- інформація очищається від непотрібних деталей до того, як запит буде сформовано.

Таким чином, можна дотриматися основних принципів захисту даних при використанні зовнішнього API.

2.7.6 Приклад відповіді у застосунку.

- Рівень ризику: високий
- Коротка оцінка ризику: Надсилання незахищених конфіденційних даних через email може призвести до їх незаконного доступу і витоку, що може порушити безпеку і порядок.
- Рекомендації для мінімізації ризику: Пройти навчання з правил безпеки даних та конфіденційності, використовувати захищені канали для надсилання конфіденційних даних, встановити і використовувати шифрування для захисту інформації.

Результат передається до фронтенду і відображається у зручному вигляді та зберігається у базі даних користувача.

Висновок:

Інтеграція OpenAI API значно покращує функціональність системи та дозволяє проводити глибокий аналіз ризиків на основі вільного тексту. Коректна генерація підказок, управління нестабільністю API та захист даних є ключовими елементами при впровадженні OpenAI API.

2.8 Авторизація, безпека та захист від шахрайства.

Система оцінки ризиків повинна забезпечувати не тільки функціональність, але й надійний рівень безпеки, при використанні персональних даних користувачів та взаємодії із зовнішніми сервісами. Для цього в розробленому додатку реалізована сучасна система авторизації, валідації, захист API.

2.8.1. Реалізація авторизації.

Для забезпечення обмеженого доступу до голового функціоналу (оцінка ризику, історія запитів тощо) реалізовано систему **авторизації на основі токена JWT (JSON Web Token)**.

2.8.1.1 Реєстрація користувача.

– Введення адреси електронної пошти та пароля.
– Перевірка даних на стороні клієнта та на стороні сервера.
– Пароль хешується за допомогою бібліотеки bcrypt перед тим, як він буде збережений в базі даних.
– Створення облікового запису в колекції користувачів.

2.8.1.2 Доступ користувача.

– Перевірка адреси електронної пошти та пароля.
– Якщо аутентифікація пройшла успішно, генерується JWT, який надсилається користувачеві.
– Токен зберігається в файлі cookie localStorage або httpOnly (залежно від середовища).

2.8.1.3 Захист маршруту.

– На бекенді реалізовано NestJS-guard, який перевіряє наявність та дійсність токену.
– Всі маршрути, пов'язані з оцінкою ризиків та історією, захищені:
  – /api/risk/manual
  – /api/risk/ai
  – /api/history

2.8.2 Безпечне зберігання та обробка даних.

Система дотримується основних принципів інформаційної безпеки:

– Паролі зашифровані (паролі не зберігаються у відкритому вигляді)
– HTTPS-з'єднання при використанні у виробничому середовищі
– Валідація даних за допомогою DTO (валідатор класів), як вхідних, так і вихідних
– Обмеження доступу до критичних API на основі токенів.

Всі взаємодії з базою даних реалізовані з обробкою винятків та обмеженням доступу до сторонніх записів (кожна оцінка ризику пов'язана з ідентифікатором користувача).

2.8.3 Захист від неправомірного використання API.

Через інтеграцію з OpenAI API важливо мінімізувати ймовірність:

1. Надмірної кількості запитів від зовнішніх сервісів
2. автоматизованих атак (ботів)
3. перевантаження ресурсоємних маршрутів.

2.8.3.1 Обмеження.

Реалізованно проміжне програмне забезпечення, яке дозволяє максимум один запит до `/api/risk` кожні 5 секунд. Якщо це значення перевищено, повертається відповідь 429 Too Many Requests.

2.8.3.2 Захист форм.

На фронтенді реалізована валідація форм з перевіркою типу та обов'язкових полів. Також додана обробка некоректних запитів для запобігання надсилання порожніх або маніпульованих значень.

Висновок.

Завдяки використанню сучасних методів авторизації, безпеки маршрутів, контролю частоти запитів і валідації вхідних даних, система забезпечує необхідний рівень безпеки для веб-додатків, що працюють з персональними даними та сторонніми API. Це дозволяє підтримувати цілісність даних, запобігати зловживанням і забезпечувати безпечну взаємодію між користувачем і системою.

3 РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКА

3.1 Налаштування середовища розробки.

Перш ніж почати впровадження веб-додатку, необхідно підготувати робоче середовище для локальної розробки, тестування та налагодження. Враховуючи архітектуру системи, яка передбачає поділ на клієнтську (frontend) і серверну частини (backend), було створено два окремих проекти з незалежними конфігураціями, які спілкуються між собою через REST API.

3.1.1 Вибрані інструменти.

Для реалізації проекту використовувалося таке середовище:

Операційна система	macOS
Редактор коду	Visual Studio Code
Менеджер пакетів	Yarn
Менеджер версій	Git (з GitHub для зберігання репозиторію)
Браузер	Google Chrome

3.1.2 Ініціалізація фронтенд частини React.

1. Створення проєкту:
   ```
   npx create-react-app risk-ui --template typescript
   ```
2. Встановлення основних залежностей:
   ```
   yarn add @mui/material @emotion/react @emotion/styled
   yarn add axios react-router-dom
   yarn add react-hook-form yup
   ```
3. Запуск локального сервера:
   ```
   yarn dev
   ```

Після виконання команди відкриється React-додаток на порту http://localhost:5173.

3.1.3. Ініціалізація бекенд частини NestJS.

1. Створення проєкту:
   ```
   npm i -g @nestjs/cli
   ```

```
nest new risk-backend
```

2. Встановлення залежностей:

```
npm install @nestjs/mongoose mongoose
npm install @nestjs/jwt passport-jwt
npm install axios class-validator class-transformer
```

3. Запуск бекенду:

```
npm run start:dev
```

API системи працює на порту http://localhost:5000 заданому в файлі .env.

3.1.4. Налаштування MongoDB.

В даній системі було використанне локальне підключення MongoDB.

1. Створення та налаштування кластеру на офіційному сайті
2. Підключення через MongoDB через URI

Висновок:

Впровадження середовища розробки є обов'язковим кроком для успішної реалізації веб-додатку. Чіткий поділ на фронтенд і бекенд, незалежне налаштування та використання сучасних інструментів забезпечують систему надійною основою для подальшого додавання логіки взаємодії з користувачем, аналізу ризиків і зберігання даних.

3.2 Реалізація авторизації користувача.

У будь-якому сучасному веб-додатку, який працює з користувацькими даними, механізм авторизації відіграє важливу роль. У цій системі реалізована базова авторизація користувача на основі електронної пошти та пароля з використанням JWT (JSON Web Token) для захисту запитів API та контролю доступу до приватних функцій.

3.2.1 Реєстрація користувача.

На стороні клієнта була створена реєстраційна форма з двома полями:

– Електронна пошта (потрібна перевірка формату)

– Пароль (не менше 6 символів)

Валідація реалізована за допомогою `yup + hookform`

```
const validationSchema = Yup.object({
  email: Yup.string().email().required(),
  password: Yup.string().min(6).required(),
})
```

3.2.1.2 API-запит до бекенду.

Запит до бекенду був реалізований за допомогою бібліотек `Redux + Redux-thunk`.

```
export const registerUser = createAsyncThunk(
  "auth/registerUser",
  async (payload: { email: string; password: string }) => {
    const res = await api.post("/auth/register", payload);
    localStorage.setItem("token", res.data.access_token);
    return res.data;
  }
);
```

3.2.1.3 Реєстрація на бекенді.

1. Вхідні дані валідуються через DTO:

```
export class RegisterDto {
@IsEmail()
email: string;
@IsNotEmpty()
@MinLength(6)
password: string;
}
```

2. Пароль хешується за допомогою бібліотеки bcrypt:

```
    async register(email: string, password: string) {
    const hashed = await bcrypt.hash(password, 10);
    const user = await this.usersService.create(email, hashed);
    return { message: 'User registered successfully', userId:
    user._id };
      }
```

3. Після успішного виконання рєстрації данні зберігаються в базу данних під ключем user.

3.2.2 Вхід користувача (Login).

Форма на фронтенді – аналогічна до форми реєстрації.

3.2.2.1 API-запит до бекенду.

```
export const login = createAsyncThunk(
  "auth/login",
  async (data: { email: string; password: string }, { rejectWithValue
}) => {
    try {
      const response = await api.post("/auth/login", data);
      localStorage.setItem("token", response.data.access_token);
      return response.data;
    } catch (error: any) {
      return rejectWithValue(error.response.data.message || "Login
error");
    }
  }
);
```

3.2.2.2 Бекенд.

Коли бекенд отримує запит на вхід, він знаходить користувача за email, перевіряє на відповідність пароль. У разі успіху бекенд генерує JWT-токен та відає в тілі відповіді.

```
  async login(email: string, password: string) {
```

```
    const user = (await this.usersService.findByEmail(email)) as
UserDocument;
    if (!user || !(await bcrypt.compare(password, user.password))) {
      throw new UnauthorizedException('Invalid credentials');
    }

    return {
      access_token: this.jwtService.sign({
        sub: user._id as string,
        email: user.email,
      }),
    };
  }
```

3.2.3 Генерація та структура JWT.

JWT токен формується таким чином:

```
     access_token: this.jwtService.sign({
        sub: user._id as string,
        email: user.email,
      }),
```

Токен містить userId та email і також част створення токену.

Також файл `.env` зберігає ключ підпису.

```
JWT_SECRET=my-ultra-secret-key
```

3.2.4 Захист API-маршутів.

Для захисту приватних маршутів(оцінка ризику, історія), реалізованно `JwtAuthGuard`

```
  @UseGuards(JwtAuthGuard)
  @Get('history')
  getUserRiskHistory(@Request() req) {
    return this.riskService.getHistoryForUser(req.user.userId);
  }
```

3.2.5 Опрацювання токену на сторіні клієнта.

Після успішного виконання запиту авторизації, фрондент отримує токен та зберігає його в `LocalStorage` на рівні браузера.

```
localStorage.setItem("token", res.data.access_token)
```

Axios автоматично додає токен до кожного запиту через інтерцептор.

```
axios.interceptors.request.use((config) => {
  const token = localStorage.getItem('token')
  if (token) {
    config.headers.Authorization = `Bearer ${token}`
  }
  return config
})
```

3.2.6 Перевірка автентифікації на фронтенді.

Для того щоб перевірити чи авторизований в систему користувач, було створенно компонент `ProtectedRoute`. Після завантаження додатка, цей компонент намагається отримати з `LocalStorage` токен користувача. Якщо токен знайдено, ми відкриваємо головну сторінку, а якщо токен не був знайдений, система переадресовує користувача на сторінку авторизації.

```
import { JSX } from "react";
import { Navigate } from "react-router-dom";

const ProtectedRoute = ({ children }: { children: JSX.Element }) => {
  const token = localStorage.getItem("token");

  if (!token) {
    return <Navigate to="/login" replace />;
  }

  return children;
};
```

```
export default ProtectedRoute;
```

3.2.7 Безпека.

- Паролі не зберігаються у вигляді звичайного тексту;
- Передача даних відбувається через HTTPS (у робочому середовищі);
- JWT має термін дії, після якого користувач повинен увійти знову;
- Обробка помилок: неправильний пароль, неіснуючий користувач.

Висновок:

Механізм авторизації користувачів реалізовано з використанням сучасних підходів: статична перевірка, хешування паролів, токенізація через JWT, захист шляху через guards і зручна інтеграція з фронтендом через перехоплювачі. Це забезпечує безпечну взаємодію та захищений доступ до налаштованих функцій системи.

3.3 Реалізація ручного режиму оцінки ризиків.

Режим ручної оцінки ризиків є найпростішим функціональним блоком системи, в якому користувач вводить цифрові значення параметрів процесу. Це дозволяє швидко отримати результат у вигляді об'єктивного числового показника ризику на основі стандартної формули.

3.3.1 Формула для оцінки ризику.

У ручному режимі рівень ризику розраховується за класичною формулою:

R = P * S *E

де :

- P (Probability) - ймовірність події
- S (Severity) - тяжкість наслідків
- E (Exposure) - частота впливу

Кожен параметр вводиться користувачем в діапазоні від 1 до 5.

3.3.2 Реалізація на клієнті.

Реалізовано компонент з трьома лініями для вибору оцінки кожного з параметру (вибір або тип введення=число) для визначення значення кожного параметру.

```
<Typography gutterBottom>Ймовірність (P)</Typography>
      <Slider
        value={probability}
        onChange={(_, val) => setProbability(val as number)}
        min={1}
        max={5}
        marks={marks}
        valueLabelDisplay="auto"
      />

      <Typography gutterBottom>Серйозність наслідків (S)</Typography>
      <Slider
        value={severity}
        onChange={(_, val) => setSeverity(val as number)}
        min={1}
        max={5}
        marks={marks}
        valueLabelDisplay="auto"
      />

      <Typography gutterBottom>Частота експозиції (E)</Typography>
      <Slider
        value={exposure}
        onChange={(_, val) => setExposure(val as number)}
        min={1}
        max={5}
        marks={marks}
        valueLabelDisplay="auto"
      />

      <Button
        variant="contained"
```

```
        color="primary"
        onClick={calculateRisk}
        sx={{ mt: 3, width: "100%", height: 48, fontWeight: 600 }}
      >
        Обчислити ризик
      </Button>
```

3.3.3 Реалізація на бекенді.

DTO.

```
export class ManualRiskDto {
@IsInt() @Min(1) @Max(5) probability: number;
@IsInt() @Min(1) @Max(5) severity: number;
@IsInt() @Min(1) @Max(5) exposure: number;
}
```

Контролер.

```
@Post()
@UseGuards(JwtAuthGuard)
async evaluateRisk(@Body() dto: ManualRiskDto, @Req() req) {
return this.riskService.evaluateManualRisk(dto, req.user.userId);
}
```

Сервіс

```
evaluateManualRisk(dto: ManualRiskDto, userId: string) {
const r = dto.probability * dto.severity * dto.exposure;

let level: string;
if (r <= 25) level = 'низький';
else if (r <= 50) level = 'середній';
else if (r <= 75) level = 'високий';
else level = 'критичний';

const record = new this.riskModel({
userId,
mode: 'manual',
input: dto,
```

```
result:level,
createdAt: new Date()
});

return record.save().then(() => ({
riskLevel:level,
score: r
}));
}
```

3.3.4 Виведення результату.

Після успішного запиту:

На фронтенді відображається компонент з результатом.

```
        <Paper
          elevation={3}
          sx={{
            mt: 5,
            p: 3,
            borderRadius: 3,
            backgroundColor: "#fafafa",
            borderLeft: `6px solid ${getRiskLevel(result).color}`,
          }}
        >
          <Typography variant="h6" gutterBottom>
            Результат оцінки ризику
          </Typography>

          <Stack spacing={1} mb={2}>
            <Typography variant="body2">
              <strong>Ймовірність (P):</strong> {probability}
            </Typography>
            <Typography variant="body2">
              <strong>Серйозність наслідків (S):</strong> {severity}
            </Typography>
            <Typography variant="body2">
```

```
            <strong>Частота експозиції (E):</strong> {exposure}
          </Typography>
        </Stack>

        <Divider sx={{ my: 2 }} />

        <Typography variant="body1" gutterBottom>
          Формула:{" "}
          <strong>
            R = {probability} × {severity} × {exposure}
          </strong>
        </Typography>
        <Typography variant="h5" fontWeight={700}>
          R = {result}
        </Typography>

        <Typography
          variant="subtitle1"
          fontWeight={600}
          sx={{ mt: 1, color: getRiskLevel(result).color }}
        >
          {getRiskLevel(result).label}
        </Typography>
      </Paper>
```

Висновок:

Реалізація ручного режиму є важливою частиною програми, яка дозволяє швидко і точно розрахувати рівень ризику на основі трьох ключових параметрів. Завдяки інтеграції інтерфейсу з бек-ендом і базою даних користувач отримує чіткий результат з можливістю збереження оцінки для подальшого аналізу.

3.4 Реалізація оцінки ризиків за допомогою штучного інтелекту

Штучний інтелект дозволяє користувачеві отримати прогноз ризику на основі вільного текстового опису процесу. Це стало можливим завдяки інтеграції з мовною

моделлю GPT (через OpenAI API), яка аналізує введену інформацію, визначає категорію ризику, пояснює логіку та надає рекомендації.

3.4.1 Структура форми на клієнті.

Користувачеві пропонується заповнити наступні поля:

- Галузь
- Назва процесу
- Опис процесу
- Відомі ризики або фактори

Форма реалізованна за допомогою бібліотек `react-hook-form` та `yup`

```
<form onSubmit={handleSubmit(onSubmit)}>
     <Controller
       name="industry"
       control={control}
       rules={{ required: "Обов'язкове поле" }}
       render={({ field }) => (
         <TextField
           {...field}
           label="Сфера (наприклад, освіта, IT, фінанси)"
           fullWidth
           margin="normal"
           error={!!errors.industry}
           helperText={errors.industry?.message}
         />
       )}
     />

     <Controller
       name="processName"
       control={control}
       rules={{ required: "Обов'язкове поле" }}
       render={({ field }) => (
         <TextField
```

```
        {...field}
        label="Назва процесу"
        fullWidth
        margin="normal"
        error={!!errors.processName}
        helperText={errors.processName?.message}
      />
    )}
  />

  <Controller
    name="processDescription"
    control={control}
    rules={{ required: "Обов'язкове поле" }}
    render={({ field }) => (
      <TextField
        {...field}
        label="Опис процесу"
        fullWidth
        multiline
        rows={4}
        margin="normal"
        error={!!errors.processDescription}
        helperText={errors.processDescription?.message}
      />
    )}
  />

  <Controller
    name="riskKnownByUser"
    control={control}
    rules={{ required: "Обов'язкове поле" }}
    render={({ field }) => (
      <TextField
        {...field}
```

```
                label="Відомі ризики (за бажанням)"
                fullWidth
                margin="normal"
                error={!!errors.riskKnownByUser}
                helperText={errors.riskKnownByUser?.message}
              />
            )}
          />

          <Button
            type="submit"
            variant="contained"
            color="primary"
            fullWidth
            sx={{ mt: 3, height: 48, fontWeight: 600 }}
          >
            {loading ? <CircularProgress size={24} /> :
" Провести оцінку"}
          </Button>

          {error && (
            <Typography color="error" mt={2}>
              {error}
            </Typography>
          )}
        </form>
```

3.4.2 API-запит до бекенду.

```
export const submitRiskThunk = createAsyncThunk(
  "risk/submit",
  async (input: RiskInput, { rejectWithValue }) => {
    try {
      const token = localStorage.getItem("token");
      const res = await api.post("/risk/assess", input, {
        headers: {
```

```
          Authorization: `Bearer ${token}`,
        },
      });
      return res.data;
    } catch (err: any) {
      return rejectWithValue(
        err.response?.data?.message || "Something went wrong"
      );
    }
  }
);
```

3.4.3 Реалізація бекенду.

DTO.

```
export class AssessRiskDto {
  @IsString()
  @IsNotEmpty()
  industry: string;

  @IsString()
  @IsNotEmpty()
  processName: string;

  @IsString()
  @IsNotEmpty()
  processDescription: string;

  @IsArray()
  @IsString({ each: true })
  risksKnownByUser: string[];
}
```

Сервіс (RiskService).

На цьому етапі ми формуємо промпт до моделі OpenAi використовуючи дані що вів користувач на фронтенді. Після цього ми відправляємо запит до штучного інтелекту та отримуємо відповідь. Цю відповідь ми зберігаємо в нашу базу даних.

```
export class RiskService {
  constructor(
    private readonly openAiService: OpenAiService,
    @InjectModel(RiskHistory.name)
    private readonly riskHistoryModel: Model<RiskHistory>,
  ) {}

  private buildPrompt(dto: AssessRiskDto): string {
    return `
Оціни рівень ризику для невиробничого процесу в галузі
"${dto.industry}":

Назва процесу: ${dto.processName}
Опис процесу: ${dto.processDescription}
Відомі ризики: ${dto.risksKnownByUser?.join(', ') ?? 'немає
інформації'}

Дай відповідь у форматі:
- Рівень ризику: (низький / середній / високий)
- Коротка оцінка ризику
- Рекомендації для мінімізації ризику
    `.trim();
  }

  async assessRisk(
    dto: AssessRiskDto,
    user: JwtPayload,
  ): Promise<{
    response: string;
  }> {
```

```
    const prompt = this.buildPrompt(dto);
    const response = await this.openAiService.generate(prompt);

    await this.riskHistoryModel.create({
      user: user.userId,
      industry: dto.industry,
      processName: dto.processName,
      processDescription: dto.processDescription,
      risksKnownByUser: dto.risksKnownByUser,
      response,
    });

    return {
      response: response,
    };
  }

  async getHistoryForUser(userId: string) {
    return this.riskHistoryModel.find({ user: userId }).sort({
createdAt: -1 });
  }
}
```

3.4.4 Виведення результату на клієнті.

Після успішного виконання запиту, фронтенд опрацьовує тіло відповіді запиту та зберігає данні в redux. Після чого відбувається виведення результату користувачу.

```
<div ref={resultRef}>
        <Paper elevation={6} sx={{ p: 4, borderRadius: 3 }}>
          <Typography variant="h6" gutterBottom fontWeight={600}>
            Результат оцінки
          </Typography>

          {loading ? (
```

```
            <Typography sx={{ mt: 2 }}>Очікування
відповіді...</Typography>
          ) : success && result ? (
            <Paper
              elevation={0}
              sx={{
                p: 3,
                mt: 2,
                backgroundColor: "#f9f9f9",
                borderRadius: 3,
                borderLeft: "5px solid #1976d2",
              }}
            >
              <Typography
                variant="subtitle1"
                fontWeight={700}
                gutterBottom
                color="primary"
              >
                AI Висновок
              </Typography>
              <Divider sx={{ mb: 2 }} />
              <Stack spacing={2}>
                {result.split("\n").map((line, index) => (
                  <Typography
                    key={index}
                    variant="body1"
                    sx={{ whiteSpace: "pre-wrap" }}
                  >
                    {line}
                  </Typography>
                ))}
              </Stack>
            </Paper>
          ) : (
```

```
            <Typography variant="body2" color="text.secondary" sx={{
mt: 1 }}>
              Після надсилання форми тут з’явиться висновок AI на
основі
              введених даних.
            </Typography>
          )}
        </Paper>
      </div>
```

3.4.5 Обробка помилок та обмежень.

- Якщо відповідь GPT не містить очікуваних блоків, повертається повідомлення «Спробуйте змінити опис»;
- У разі помилки API (таймаут, ліміт) повертається повідомлення «Сервіс тимчасово недоступний»;
- Періодичність запиту обмежена (раз на 5 секунд).

Висновок:
Інтелектуальний режим значно підвищує корисність системи, оскільки дозволяє проводити поглиблену оцінку ризиків на основі будь-якого опису. Використовуючи OpenAI API, система автоматично формує аргументований висновок і рекомендації, що робить її доступною для користувачів, які не мають досвіду в управлінні ризиками.

3.5 Огляд системи.

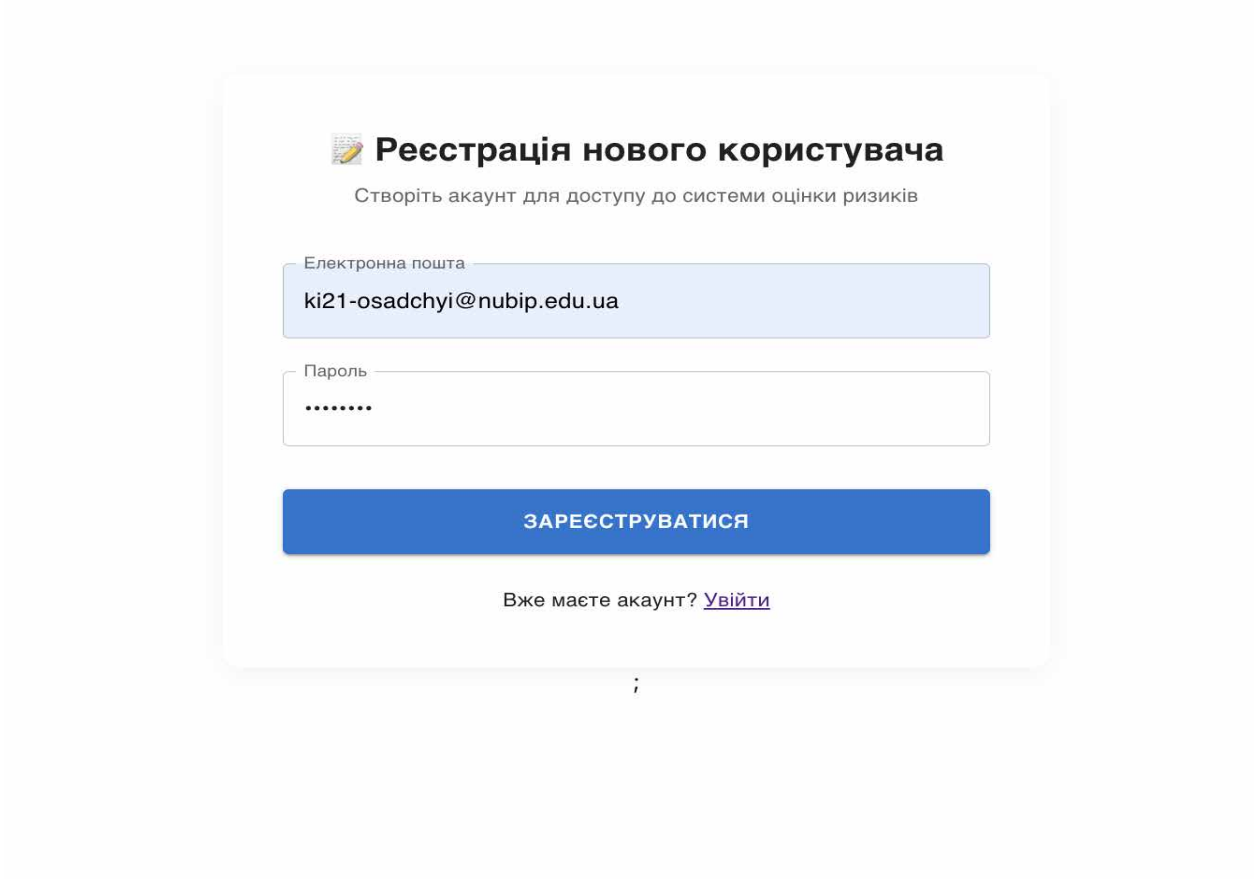

Рисунок 3.1 – форма реєстрації новго користувача

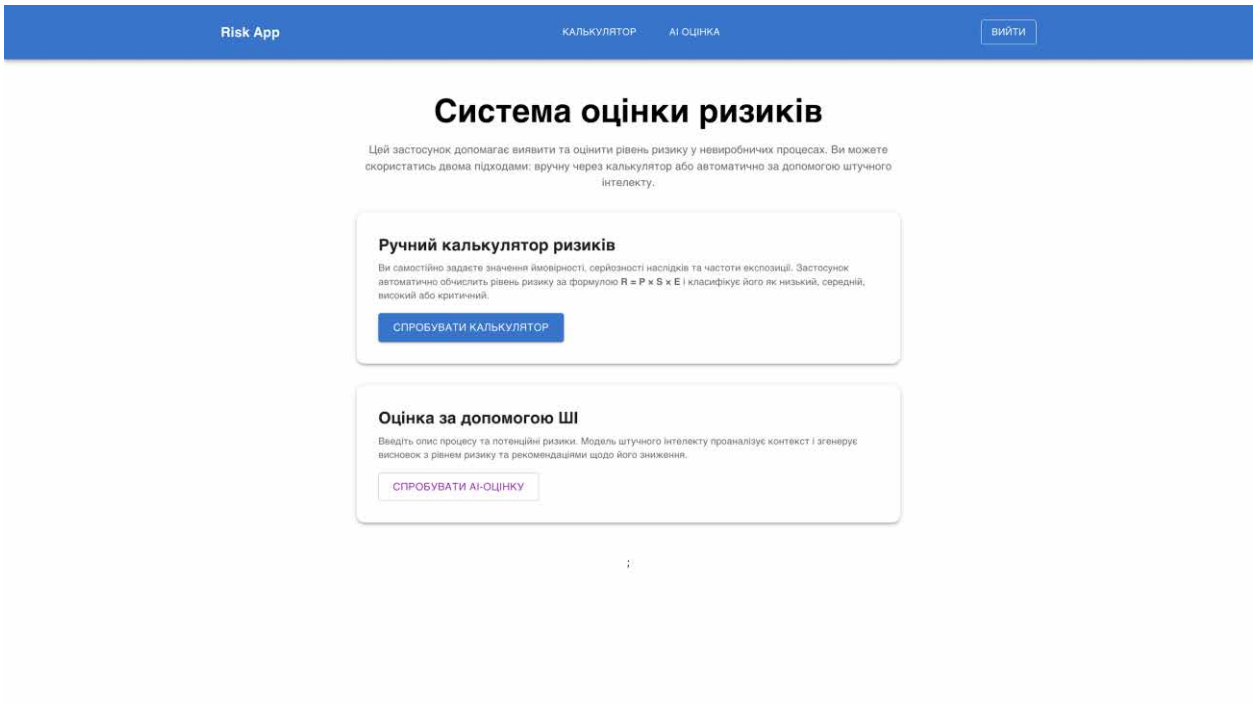

Рисунок 3.2 – головна сторінка системи

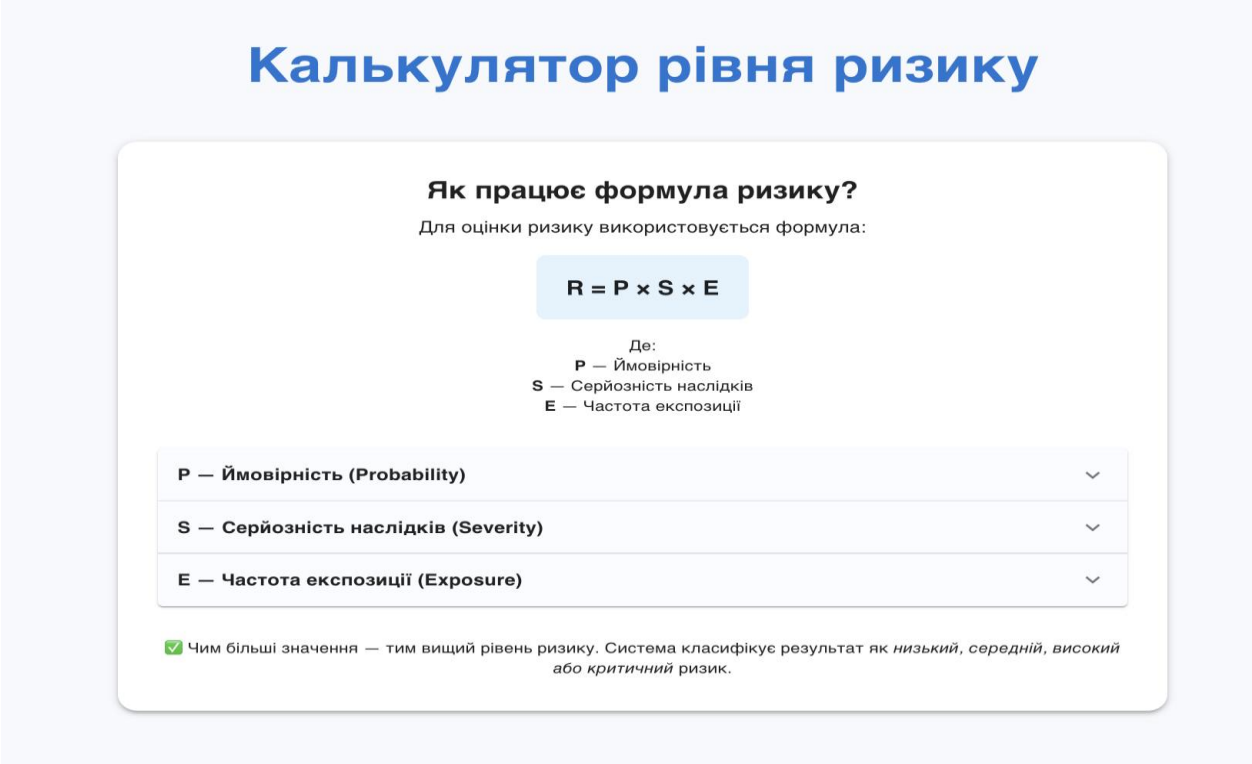

Рисунок 3.3 – сторінка ручного режиму оцінки ризиків

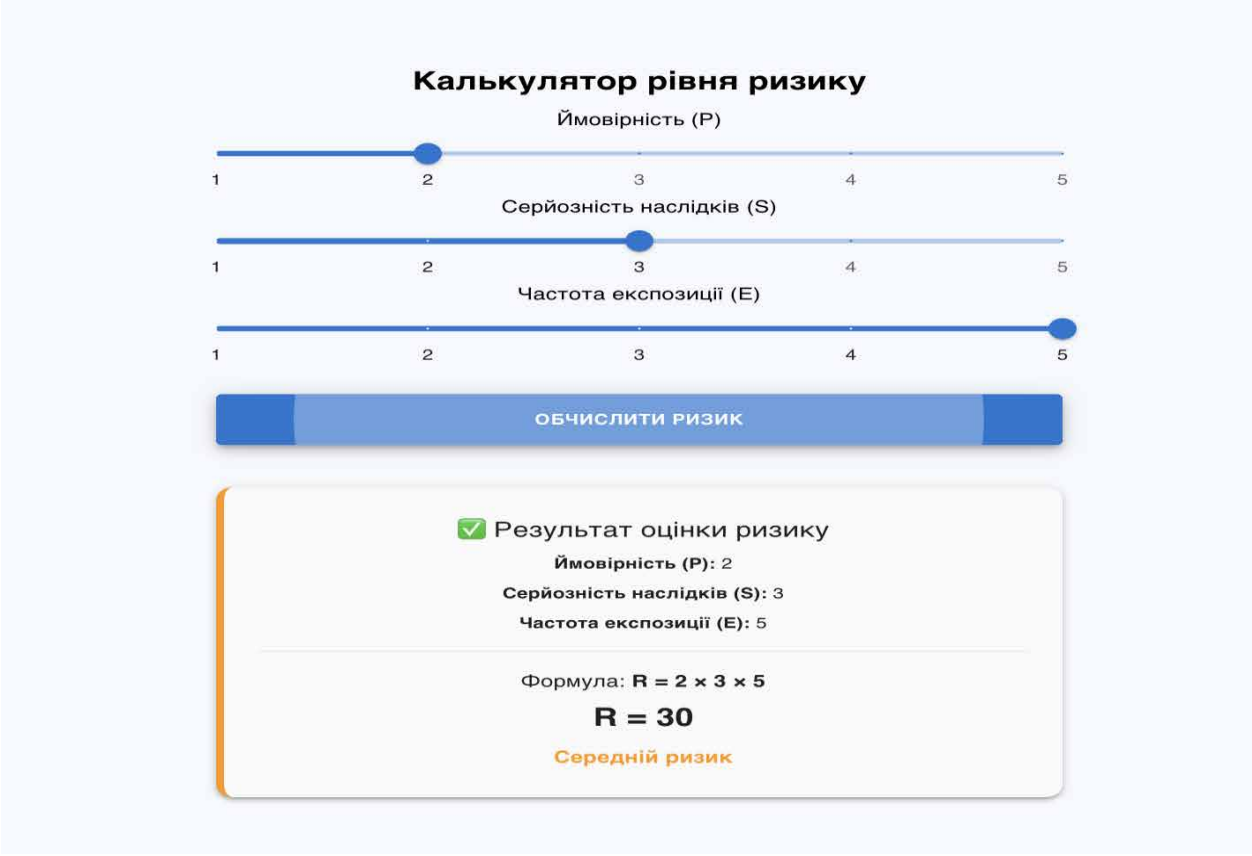

Рисунок 3.4 – калькулятор ручного режиму оцінки ризиків та демонстрація відповіді

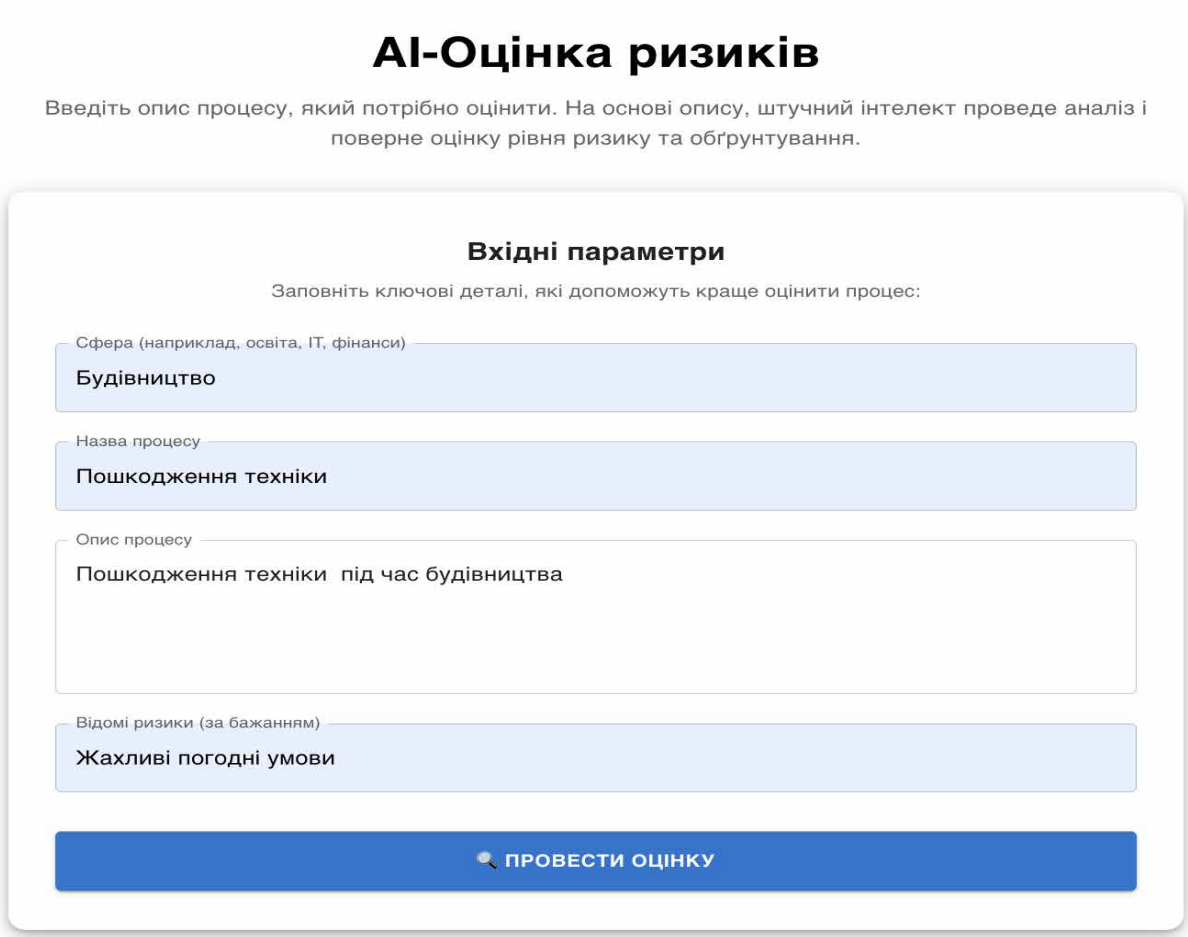

Рисунок 3.5 – сторінка інтелектульного режиму оцінки ризиків

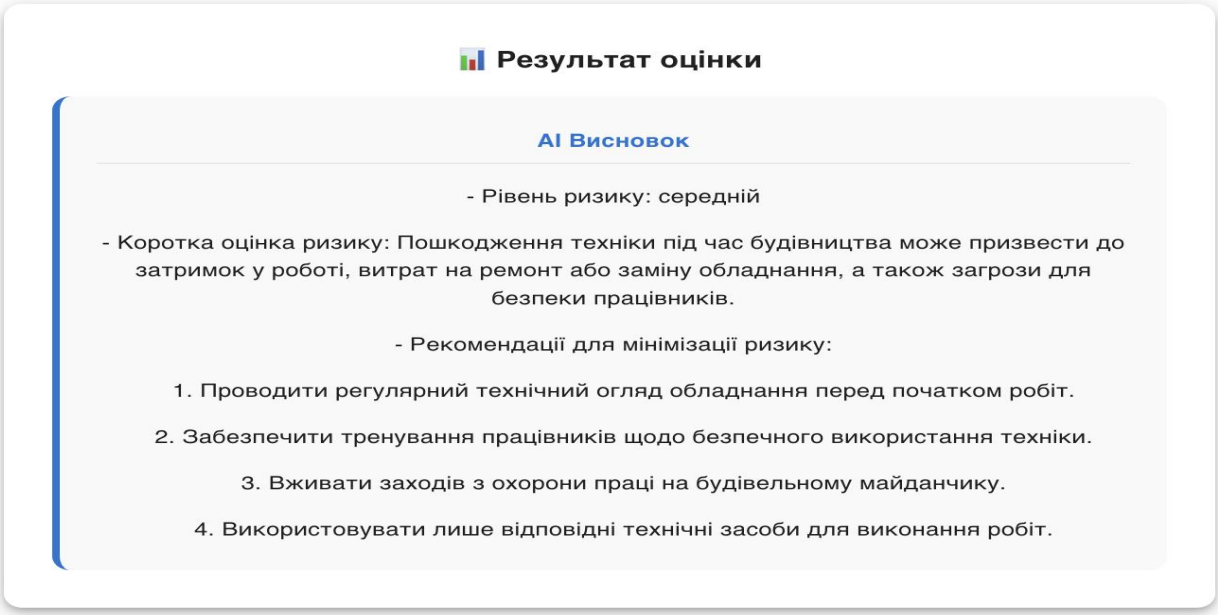

Рисунок 3.6 − демонстрація результату оцінки інтелектульного режиму

ВИСНОВКИ

У рамках дипломної роботи було реалізував повноцінну веб-систему для інтелектуальної та ручної оцінки ризиків у невиробничих процесах. Розробка поєднала в собі сучасні підходи до управління ризиками, інструменти веб-розробки та можливості штучного інтелекту, що дозволило отримати ефективний та простий у використанні програмний продукт.

В ході роботи були отримані наступні результати:

- Аналіз теоретичних основ
- Огляд існуючих програм
- Обґрунтування вибору архітектури та технологічного стеку
- Розробка клієнта та серверної частини
- Реалізація авторизації користувачів
- Тестування системи з використанням прикладних сценаріїв

Розроблена таким чином система успішно виконує завдання оцінки ризиків у невиробничих процесах без залучення спеціалістів, на основі опису ситуації. Це робить її ефективним інструментом для навчальних закладів, органів державної влади, малого бізнесу та інших користувачів, які потребують швидкої та недорогої оцінки ризиків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. «Як проводити оцінку ризиків» [Електронний ресурс] – Режим доступу: https://qftp.org/wp-content/uploads/2020/09/21fc1.pdf
2. «РИЗИКИ В ДІЯЛЬНОСТІ ПРОМИСЛОВИХ ПІДПРИЄМСТВ» [Електронний ресурс] – Режим доступу: http://www.visnyk-econom.uzhnu.uz.ua/archive/17_2_2018ua/31.pdf
3. «Що таке аналіз ризиків» [Електронний ресурс] – Режим доступу: https://visuresolutions.com
4. «П’ять кроків в оцінки ризиків» [Електронний ресурс] – Режим доступу: https://ukrprofzahyst.com.ua/ua/obuchayuschie-statyi/12-pyat-shagov-v-otsenke-riskov-na-proizvodstve
5. «Що таке React?» [Електронний ресурс] – Режим доступу: https://dan-it.com.ua/uk/blog/chto-takoe-react-js-i-dlja-chego-on-nuzhen/
6. «Основи MongoDb» [Електронний ресурс] – Режим доступу: https://devzone.org.ua/post/osnovy-mongodb
7. «Розробка додатків на Nest.js» [Електронний ресурс] – Режим доступу: https://foxminded.ua/nest-js/
8. «Ризики в діяльності промислових підприємств » [Електронний ресурс] – Режим доступу: http://www.visnyk econom.uzhnu.uz.ua/archive/17_2_2018ua/31.pdf
9. «Системи управління ризиками » [Електронний ресурс] – Режим доступу: https://studies.in.ua/mutne-pravo-shpora/4587-sistema-upravlnnya-rizikami.html
10. «Що таке штучний інтелект » [Електронний ресурс] – Режим доступу: https://gigacloud.ua/articles/shho-take-shtuchnyj-intelekt-istoriya-vydy-ta-skladovi/