МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ПРИРОДОКОРИСТУВАННЯ

Факультет механіки, енергетики та інформаційних технологій

Кафедра інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

першого (бакалаврського) рівня вищої освіти

на тему:

«РОЗРОБКА ВЕБ-ЗАСТОСУНКУ ДЛЯ ОБСЛУГОВУВАННЯ ТРУДОМІСТКИХ ЗАПИТІВ КОРИСТУВАЧІВ»

Виконав студент групи –Іт-41
спеціальності 126 – Інформаційні технології

Киця В.В.
(прізвище та ініціали)

Керівник: Смолінський В.Б.
(прізвище та ініціали)

Дубляни 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Перший (бакалаврський) рівень вищої освіти
Спеціальність 126 «Інформаційні системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
(підпис)
д.т.н., професор, Тригуба А. М.
(вч. звання, прізвище, ініціали)
“____” _____202 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Киці Володимира Віталійовича
(прізвище, ім’я, по батькові)

1. Тема роботи «Веб застосунок для обслуговування трудомістких запитів користувачів»

керівник роботи *к. е. н., доцент., Смолінський В. Б.*.
(наук. ступінь, вч. звання, прізвище, ініціали)

затверджені наказом Львівського НУП *від 27.11.2023 року № 641/к-с*.
2. Строк подання студентом роботи *10 червня 2024 року*.
3. Вихідні дані до роботи: визначені тема, об’єкт і предмет дослідження, окреслене коло завдань, на яких має бути зосереджена увага в процесі виконання роботи. Обрано набір технологій, необхідних для реалізації завдання.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
   Вступ
   Аналіз предметної області
   Вимоги та технології
   Програмна реалізація
   Охорона праці та безпека в надзвичайних ситуаціях
   Висновки
   Список використаних джерел
5. Перелік графічного матеріалу
   Графічний матеріал подається у вигляді презентації

6. Консультанти розділів

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата		Відмітка про виконання
		завдання видав	завдання прийняв	
1, 2, 3	*Смолінський В. Б., к.е.н., доцент*			
4	*Городецький І. М., к.т.н., доцент*			

7. Дата видачі завдання *28 листопада 2023 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Відмітка про виконання
1	*Аналіз вимог*	*28.11.2023 – 31.12.2023*	
2	*Відомості про використані технології*	*01.01.2024 – 28.02.2024*	
3	*Проектування та реалізація застосунку*	*01.03.2024 – 30.04.2024*	
4	*Розгляд питань з охорони праці та безпеки у надзвичайних ситуаціях*	*01.05.2024 – 14.05.2024*	
5	*Завершення оформлення розрахунково-пояснювальної записки та презентаційного матеріалу*	*15.05.2024 – 31.05.2024*	
6	*Завершення роботи в цілому. Підготовка до захисту кваліфікаційної роботи*	*01.06.2024 – 10.06.2024*	

Здобувач _____ *Киця В.В.* .
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Смолінський В. Б.* .
(підпис) (прізвище та ініціали)

УДК 004.78

Розробка веб-застосунку для обслуговування трудомістких запитів користувачів. Кваліфікаційна робота. Киця Володимир Віталійович, кафедра інформаційних технологій, Дубляни, Львівський НУП, 2024р.

45с. текст. част.,5 табл., 10 рис., 22 джерела.

Зі зростанням обсягів даних з'являється виклик ефективного управління та обробки цих даних, зокрема, у веб-застосунках, які повинні ефективно обслуговувати користувацькі запити. Сучасні веб-застосунки стають все більш складними, використовуючи різноманітні технології, фреймворки та інтеграції. Це може призводити до збільшення часу оброблення та виконання трудомістких операцій. Користувачі сучасних веб-застосунків очікують миттєвої відповіді та високої продуктивності. Завдяки швидкій роботі інших сервісів (наприклад, пошукових систем чи соціальних мереж), користувачі стають більш вимогливими щодо часу реакції веб-застосунків на їхні запити.

Кваліфікаційна робота присвячена вирішенню цих проблем, а саме в роботі описано веб-застосунок , що обслуговує трудомісткі запити користувачів.

Ключові слова: веб-застосунок, трудомісткі запити, фреймворк ASP.NET, бібліотека React для роботи з JavaScript, СУБД Microsoft SQL Server.

Keywords: web application, resource-intensive queries, ASP.NET framework, React library for working with JavaScript, Microsoft SQL Server DBMS.

Зміст

ВСТУП

Визначення проблематики, актуальність теми

Аналізуючи сучасні тенденції в сфері веб-розробки, можна зауважити, що обслуговування трудомістких запитів користувачів у веб додатках підкреслюють актуальність проблеми.

Наступні фактори є ключовими:

1. Зростання обсягів даних:

Споживачі та компанії генерують та обробляють все більше даних. З цим зростанням обсягів даних з'являється виклик ефективного управління та обробки цих даних, зокрема, у веб-застосунках, які повинні ефективно обслуговувати користувацькі запити.

2. Складність веб-застосунків:

Сучасні веб-застосунки стають все більш складними, використовуючи різноманітні технології, фреймворки та інтеграції. Це може призводити до збільшення часу оброблення та виконання трудомістких операцій.

3. Зростання очікувань щодо продуктивності:

Користувачі сучасних веб-застосунків очікують миттєвої відповіді та високої продуктивності. Завдяки швидкій роботі інших сервісів (наприклад, пошукових систем чи соціальних мереж), користувачі стають більш вимогливими щодо часу реакції веб-застосунків на їхні запити.

4. Зосередження на користувацькому досвіді:

З підвищенням усвідомленості про важливість користувацького досвіду з'являється попит на зручні та швидкі інтерфейси. Велике значення надається спрощенню взаємодії користувача з веб-застосунками.

5. Мобільний та розподілений доступ:

Користувачі звертаються до веб-застосунків з різних пристроїв та місць розташування. Це створює потребу у розподіленій обробці та оптимізації для різних умов доступу.

6. Штучний інтелект та машинне навчання:

Використання штучного інтелекту та машинного навчання для оптимізації роботи веб-застосунків. Це може включати у себе автоматизацію та передбачення трудомістких операцій.

Ці тенденції підкреслюють необхідність розробки ефективних та продуктивних методів оброблення трудомістких запитів у веб-застосунках.

Метою дослідження є розробка та оптимізація веб-орієнтованого інтерфейсу для ефективного оброблення трудомістких запитів користувачів у веб-застосунках.

Завдання дослідження:

- Аналіз сучасних тенденцій в сфері обслуговування трудомістких запитів:

Провести огляд існуючих веб-орієнтованих інтерфейсів та визначити тенденції їхнього використання для трудомістких обчислень.

- Визначення вимог до веб-орієнтованого інтерфейсу:

Визначити основні вимоги до інтерфейсу, що обробляє трудомісткі запити, з урахуванням потреб користувачів та сучасних стандартів.

- Проектування архітектури системи:

Розробити архітектурну модель веб-орієнтованого інтерфейсу, яка відповідає визначеним вимогам та забезпечує ефективну обробку трудомістких запитів.

- Оптимізація веб-орієнтованого інтерфейсу:

Розробити та впровадити оптимізаційні стратегії для забезпечення ефективності роботи інтерфейсу при обробці трудомістких запитів.

- Реалізація та тестування:

Реалізувати розроблену систему та провести тестування її функціональності та продуктивності.

- Висновки та рекомендації:

Сформулювати висновки з результатів дослідження та надати рекомендації для подальших покращень та досліджень у цій області.

Об'єктом дослідження є веб-орієнтований інтерфейс та процеси обслуговування трудомістких запитів.

Предмет дослідження: розробка та оптимізація інтерфейсу, взаємодія користувача та аспекти ефективного оброблення трудомістких запитів.

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Поняття веб-додатку, їх переваги, хмарні технології

Очевидно, що для всіх користувачів персональних комп'ютерів відомо, що таке додаток Windows. Це одна з програм, яка встановлюється на комп'ютер і працює в операційному середовищі ОС Windows. Різноманітні текстові, табличні та графічні редактори, медіаплейери й поштові клієнти тощо. А що таке веб-додаток?

Ось декілька визначень з різних джерел.

Веб-додаток – розподілений додаток, в якому клієнтом виступає браузер, а сервером – веб-сервер [8].

Веб-додатки – це програми, написані скриптовою мовою (Perl, PHP та ін.) або написані мовою високого рівня та відкомпільовані під відповідну ОС (C, C++ та ін.), які працюють на стороні веб-сервера та призначені для створення інтерфейсу між користувачем та веб-сайтом [18].

Отже, **веб-додаток** – це комп’ютерна програма, яка працює в браузері, як Microsoft Word працює в ОС Windows. Тому, для доступу до програми потрібні тільки браузер та Інтернет. Зберігання та обробка інформації при такій організації обчислень відбувається на віддаленому сервері, а веб-переглядач служить програмою-клієнтом і призначеним для користувача інтерфейсом (рис. 1.1) [8].

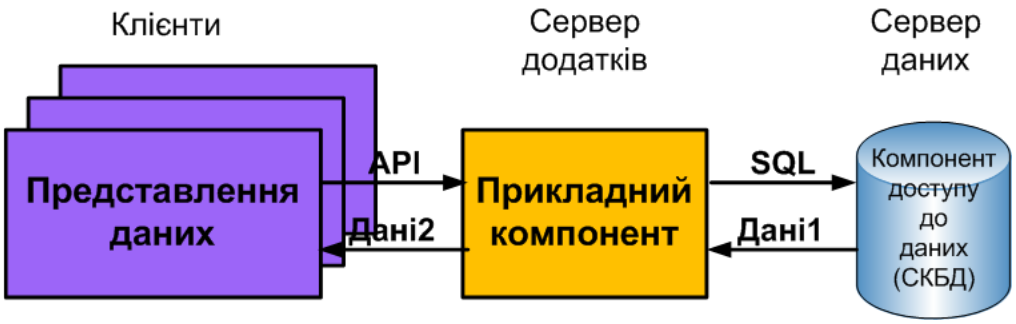

Рис.1.1. Схема роботи веб-додатку

Текстовий пакет Microsoft Office буде працювати тільки під Windows. А ось для веб-додатка операційна система встановлена на комп’ютері не має значення. Тому, що і операційною системою, і користувацьким інтерфейсом веб-додатка є браузер.

В результаті такої універсальності, постійний користувач веб-додатка може абсолютно без проблем працювати зі своєю улюбленою програмою на будь-якому зі своїх девайсів, починаючи з офісного стаціонарного комп'ютера, закінчуючи планшетом і смартфоном.

У прикладному відношенні – веб-додатки мають ту істотну перевагу: вже розроблено багато програм і сервісів, за допомогою яких будь-яка людина, не будучи програмістом і навіть просунутим користувачем, може створювати різні корисні програми для своєї зручності та розваги. Причому абсолютно безкоштовно.

Наприклад, всім відомий Gmail є повноцінним поштовим клієнтом, який робить все, що робить будь-який інший поштовий клієнт, запускається на вашому комп’ютері, і навіть трохи більше, має безліч додаткових функцій. Bloglines – веб-додаток для читання новин, яке безпосередньо конкурує із звичайними аналогічними застосуваннями, конкурує і виграє. Ці веб-додатки працюють на сервері, а їх користувацький інтерфейс (UI) відображається у вигляді веб-сторінок.

1.2. Хмарні технології

Хмарні технології – це парадигма, що передбачає віддалену обробку та зберігання даних [22].

Хмара – це деякий Центр Обробки даних (дата-центр, сервер) або їх мережа, де зберігаються дані та програми, що з’єднуються з користувачами через Інтернет

Хмарні технології дають споживачам можливість використовувати програми без установки їх на робочу машину і надають доступ до особистих файлів з будь-якого комп'ютера, що має доступ в Інтернет. Ця технологія дозволяє значно ефективніше керувати, обробляти та працювати з інформацією за рахунок централізації цієї самої інформації.

Простим прикладом хмарних технологій є сервіси електронної пошти, наприклад, Gmail, Meta і т.д. Вам потрібно всього лише підключення до Інтернету, і Ви зможете відправити пошту, при цьому додаткового програмного забезпечення або сервера не потребуючи.

Хмарні технології являють собою загальний термін для всього, що включає в себе постачання послуги хостингу через Інтернет. Ці послуги, в цілому, можна поділити на три категорії:

• Програмне забезпечення як послуга (SaaS).

По моделі SaaS постачається апаратна інфраструктура і ПЗ, також розробник забезпечує взаємодію з користувачем через інтерфейсний портал. SaaS на даний момент є досить широко розповсюдженим. За SaaS можуть надаватись самі різноманітні послуги, від веб-пошти до управління запасами, обробки БД. Перевагою такої моделі є те, що кінцевий користувач може вільно користуватись послугою з будь-якої точки світу.

• Платформа-як-сервіс (PaaS).

PaaS в хмарі визначається як набір програмних продуктів та засобів розробки, що розміщені на інфраструктурі провайдера. Розробники можуть створювати програми на платформі провайдера через Інтернет. PaaS провайдери можуть використовувати API, сайт-портали, шлюзи, або програмне забезпечення установлене на комп'ютері клієнта.

• Інфраструктура як послуга (IaaS).

IaaS являє собою віртуальний сервер instanceAPI для запуску, зупинки, доступу, налаштування своїх віртуальних серверів та систем збереження. IaaS дозволяє компанії платити саме за стільки потужностей, скільки їй необхідно. Дану модель іноді називають «комунальні обчислення» [5].

1.3. Принцип роботи веб-додатків та прикладного програмного забезпечення

В основу роботи комп'ютерів покладено програмний принцип керування, який полягає в тому, що комп'ютер виконує дії за заздалегідь заданою програмою. Цей принцип забезпечує універсальність використання комп'ютера: у певний момент часу розв'язується задача відповідно до вибраної програми. Після її завершення у пам'ять завантажується інша програма і т.д. Програма – це запис алгоритму розв'язання задачі у вигляді послідовності команд або операторів мовою, яку розуміє комп'ютер. Кінцевою метою будь-якої комп'ютерної програми є керування апаратними засобами [9].

Для нормального розв'язання задач на комп'ютері потрібно, щоб програма була налагоджена, не потребувала дороблень і мала відповідну документацію. Тому стосовно роботи на комп'ютері часто використовують термін програмне забезпечення (software), під яким розуміють сукупність програм, процедур і правил, а також документації, що стосуються функціонування системи оброблення даних.

В підсумку можна сказати, що веб-додатки мають явні переваги перед своїми програмними аналогами в основному це їх мобільність, простота в створенні та використанні, тому ця технологія стрімко набуває популярності, як серед користувачів, так і серед розробників.

РОЗДІЛ 2.
ВИМОГИ ТА ТЕХНОЛОГІЇ

2.1 Визначення вимог до веб-орієнтованого інтерфейсу

Загальні категорії вимог, які мають бути враховані при розробці інтерфейсу для ефективного оброблення трудомістких запитів:

- Інтуїтивність та Легкість використання:

Інтерфейс повинен бути легким у використанні, навіть для користувачів без технічних знань. Інтуїтивність та легкість взаємодії є ключовими для ефективності.

- Швидкість та Відгук:

Запити повинні оброблятися швидко, а інтерфейс має надавати миттєвий відгук на дії користувача.

- Адаптивність та Відповідальний дизайн:

Інтерфейс повинен адаптуватися до різних пристроїв та розмірів екранів, забезпечуючи зручну взаємодію на будь-яких пристроях.

- Надійність та Стабільність:

Система повинна бути надійною та стабільною, особливо при обробці трудомістких запитів.

- Безпека та Конфіденційність:

Забезпечення безпеки і конфіденційності інформації, особливо важливо при роботі з великими обсягами даних.

- Можливості оптимізації та Розширення:

Інтерфейс повинен надавати можливості для оптимізації та розширення, щоб враховувати майбутні потреби та зміни у вимогах.

- Доступність:

Забезпечення доступності для людей з обмеженими можливостями, щоб всі користувачі могли користуватися системою.

- Логічна організація та Зрозумілість:

Інтерфейс повинен мати логічну структуру та організацію, щоб користувачам було легко орієнтуватися та знаходити необхідні функції.

- Документація та Підтримка:

Забезпечення наявності достатньої документації та можливості отримання підтримки для користувачів.

Ці вимоги слід враховувати при розробці веб-орієнтованого інтерфейсу для обслуговування трудомістких запитів користувачів з метою забезпечення ефективної та задовільної взаємодії.

2.2 Використані технології.

Для реалізації даного проекту було використано такі технології: React (для клієнтської частини), ASP.Net (для серверної частини), MS SQL Server (як систему управління базами даних), а також NGinx (як зворотний проксі).

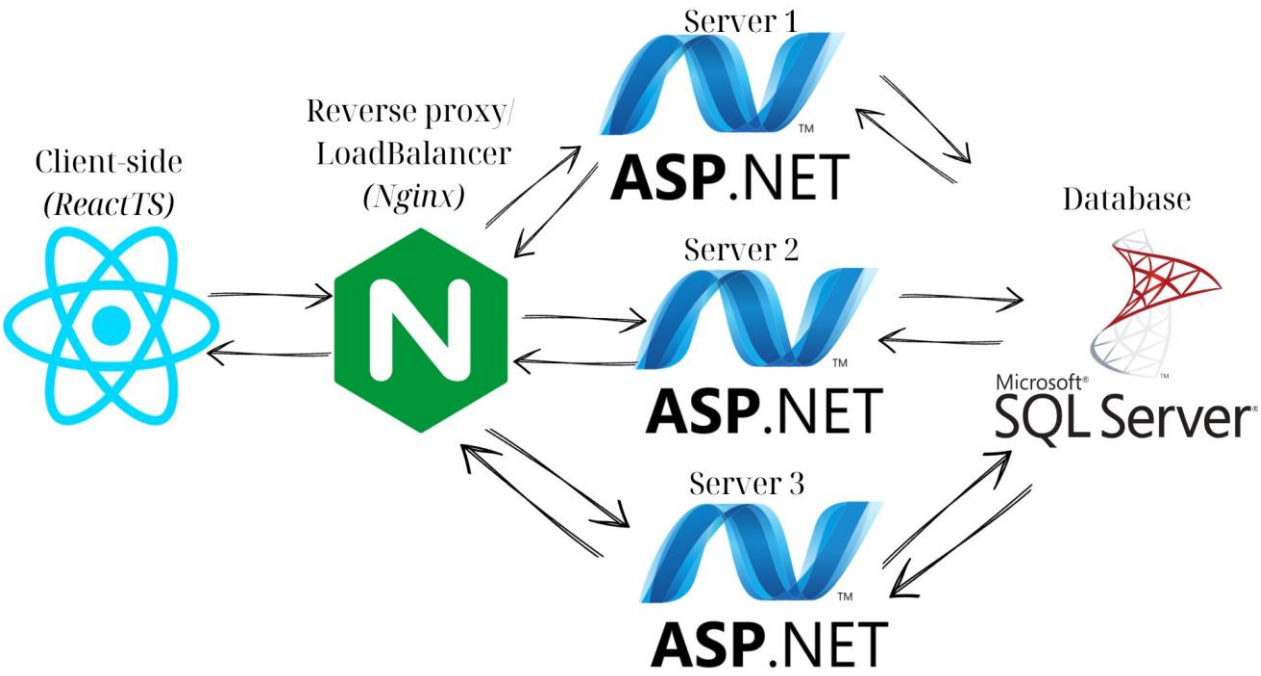

Рис. 2.1 Загальна схема проекту.

Давайте трішки детальніше їх розглянемо:

2.2.1 React

React — це популярна JavaScript бібліотека для створення користувацьких інтерфейсів, розроблена компанією Facebook. Вона використовується для розробки односторінкових додатків (SPA) і мобільних додатків, завдяки своїй гнучкості, продуктивності та можливості створення компонентів. Ось детальний опис основних характеристик React:

Основні характеристики React:

1. Компонентний підхід:

- React використовує компонентну архітектуру, де інтерфейс розбивається на окремі, багаторазові компоненти.

- Кожен компонент інкапсулює свою логіку і вигляд, що робить код більш організованим і легшим для підтримки.

2. JSX:

- JSX (JavaScript XML) — це синтаксис, що дозволяє використовувати HTML-подібні розмітки безпосередньо в JavaScript коді.

- JSX спрощує створення компонентів та опис їхньої структури, що робить код більш читабельним і зрозумілим.

3. Однонаправлений потік даних:

- React використовує однонаправлений потік даних, що означає, що дані передаються від батьківських компонентів до дочірніх через властивості (props).

- Це полегшує розуміння того, як дані проходять через додаток, і робить додаток більш передбачуваним.

4. Віртуальний DOM:

- Віртуальний DOM — це легка копія реального DOM, яка дозволяє React ефективно оновлювати інтерфейс.

- При зміні стану компонентів React оновлює віртуальний DOM, порівнює його з реальним DOM (diffing) і мінімізує кількість маніпуляцій з реальним DOM для підвищення продуктивності.

5. Стан (State):

- Компоненти React можуть мати свій власний стан (state), який визначає поточні дані та вигляд компонента.

- Зміна стану компонента призводить до його повторного рендерингу, що дозволяє динамічно оновлювати інтерфейс користувача.

6. Підтримка серверного рендерингу:

- React підтримує рендеринг на сервері (Server-Side Rendering, SSR), що покращує продуктивність і SEO додатків.

- SSR дозволяє відображати додаток на сервері та відправляти повністю сформовану HTML-сторінку клієнту.

7. React Hooks:

- React Hooks — це функції, що дозволяють використовувати стан та інші можливості React у функціональних компонентах.

- Основні хуки включають `useState`, `useEffect`, `useContext` та інші, які спрощують роботу з функціональними компонентами.

8. React Context:

- React Context API дозволяє передавати дані через компонентне дерево без необхідності передавати властивості через кожен рівень.

- Context використовується для управління глобальним станом додатку, таким як аутентифікація користувача або налаштування теми.

9. Велика екосистема:

- React має велику екосистему бібліотек і інструментів, що допомагають у розробці, таких як React Router для маршрутизації, Redux для управління станом, і інші.

- Інструменти розробки, такі як Create React App, полегшують налаштування і розгортання нових проектів.

10. Гнучкість і розширюваність:

- React може використовуватися разом з іншими бібліотеками або фреймворками, що дозволяє інтегрувати його в існуючі проекти.

- React Native — це фреймворк на основі React, що дозволяє створювати мобільні додатки для iOS та Android з використанням тих самих принципів і підходів.

Переваги використання React:

- Висока продуктивність: Завдяки віртуальному DOM і ефективним алгоритмам оновлення інтерфейсу.

- Повторне використання компонентів: Компоненти можуть бути використані повторно, що спрощує розробку і підтримку додатків.

- Велика спільнота: Широка спільнота розробників, багато ресурсів і бібліотек, що підтримуються спільнотою.

- Гнучкість: Можливість використання в різних проектах, як малих, так і великих, а також інтеграція з іншими технологіями.

React є потужною бібліотекою для створення сучасних веб-додатків, яка забезпечує високу продуктивність, гнучкість і легкість у розробці завдяки своїй компонентній архітектурі та ефективному управлінню станом.

2.2.2 Фреймворк ASP.NET

ASP.NET — це фреймворк для веб-розробки, створений компанією Microsoft, який дозволяє створювати динамічні веб-додатки, сервіси та сайти. Він базується на .NET і пропонує широкі можливості для розробки масштабованих і продуктивних веб-додатків. Ось детальний опис основних характеристик ASP.NET:

Основні характеристики ASP.NET:

1. Підтримка різних архітектур:

- ASP.NET Web Forms: Модель розробки на основі подій, що спрощує створення веб-сторінок з використанням графічного інтерфейсу і серверних контролів.

- ASP.NET MVC: Архітектурний шаблон "Model-View-Controller", який розділяє додаток на три основні компоненти (Модель, Подання, Контролер), що полегшує тестування і розробку.

- ASP.NET Web API: Фреймворк для створення HTTP-сервісів, які можуть бути доступні з будь-якого клієнта, включаючи браузери і мобільні пристрої.

- ASP.NET Core: Кросплатформенний, високопродуктивний, хмарно-орієнтований фреймворк, який об'єднує можливості MVC і Web API.

2. Кросплатформенність (для ASP.NET Core):

- ASP.NET Core може працювати на Windows, macOS і Linux, що забезпечує широку сумісність і гнучкість у виборі платформи для розгортання додатків.

3. Продуктивність і масштабованість:

- ASP.NET відомий своєю високою продуктивністю завдяки оптимізації під сучасні процесори і здатності обробляти високі навантаження.

- Підтримка асинхронного програмування дозволяє підвищити продуктивність і масштабованість додатків.

4. Безпека:

- Вбудовані засоби безпеки, включаючи аутентифікацію та авторизацію користувачів, захист від CSRF, XSS та інших веб-загроз.

- Підтримка різних методів аутентифікації, включаючи JWT, OAuth і OpenID Connect.

5. Розширюваність і модульність:

- Завдяки підтримці middleware, ASP.NET Core дозволяє легко додавати, змінювати або видаляти компоненти обробки запитів.

- Можливість використання сторонніх пакетів і бібліотек через NuGet.

6. Інтеграція з сучасними фронтенд-технологіями:

- ASP.NET легко інтегрується з різними фронтенд-фреймворками і бібліотеками, такими як Angular, React, Vue.js.

- Підтримка сучасних інструментів розробки, таких як Webpack і Babel.

7. Зручні інструменти розробки:

- Потужна інтеграція з IDE, такими як Visual Studio і Visual Studio Code, що забезпечує зручне налагодження, тестування і розробку додатків.

- Підтримка гарячого перезавантаження (Hot Reload), що дозволяє розробникам бачити зміни в реальному часі без необхідності перезавантаження додатку.

8. Робота з базами даних:

- Підтримка різних ORM, таких як Entity Framework Core, що спрощує роботу з базами даних.

- Можливість використання різних типів баз даних, включаючи SQL Server, MySQL, PostgreSQL, SQLite та інші.

9. Хмарні сервіси:

- Глибока інтеграція з Azure, що дозволяє легко розгортати, масштабувати і керувати додатками у хмарі.

- Підтримка інших хмарних провайдерів, таких як AWS і Google Cloud.

Переваги використання ASP.NET:

- Висока продуктивність: Оптимізація під сучасні процесори і здатність обробляти високі навантаження роблять ASP.NET високопродуктивним фреймворком.

- Безпека: Вбудовані засоби безпеки допомагають захистити додатки від сучасних веб-загроз.

- Гнучкість і кросплатформенність: ASP.NET Core забезпечує роботу на різних платформах і легко інтегрується з сучасними технологіями.

- Масштабованість: Підтримка хмарних сервісів і можливість асинхронного програмування дозволяють легко масштабувати додатки.

- Активна спільнота і підтримка: Велика кількість ресурсів, документації і активна спільнота розробників, що сприяє швидкому вирішенню проблем і обміну знаннями.

Використання ASP.NET Core для сучасних додатків

ASP.NET Core є основним вибором для розробки сучасних веб-додатків завдяки своїй гнучкості, продуктивності та широким можливостям. Він дозволяє створювати різноманітні додатки, від простих веб-сайтів до складних веб-сервісів і хмарних рішень.

Приклад створення простого веб-додатку на ASP.NET Core:

```csharp
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllersWithViews();
    }

    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Home/Error");
            app.UseHsts();
        }
        app.UseHttpsRedirection();
```

```
        app.UseStaticFiles();
        app.UseRouting();
        app.UseAuthorization();
        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllerRoute(
                name: "default",
                pattern: "{controller=Home}/{action=Index}/{id?}");
        });
    }
}
```

Цей код показує базову конфігурацію ASP.NET Core додатку з підтримкою MVC, обробкою статичних файлів і маршрутизацією.

ASP.NET є потужним інструментом для створення високопродуктивних, масштабованих і безпечних веб-додатків, що надає розробникам всі необхідні інструменти і можливості для реалізації сучасних веб-рішень.

2.2.3. Microsoft SQL Server

Microsoft SQL Server — це потужна реляційна система управління базами даних (СУБД), розроблена компанією Microsoft. Вона широко використовується для зберігання, управління та обробки даних в різних бізнес-додатках. Ось детальний опис основних характеристик Microsoft SQL Server:

Основні характеристики Microsoft SQL Server

1. Архітектура і компоненти:

- Реляційна база даних: SQL Server зберігає дані в таблицях з використанням реляційної моделі.

- SQL Server Management Studio (SSMS): Інтегроване середовище для управління SQL Server, що надає інструменти для адміністрування, розробки і налагодження баз даних.

- SQL Server Agent: Компонент для автоматизації завдань, таких як резервне копіювання, індексація і виконання скриптів.

2. Висока доступність і відмовостійкість:

- Always On Availability Groups: Технологія, що забезпечує високу доступність і відновлення після збоїв, дозволяючи створювати групи баз даних з реплікацією на кілька вузлів.

- Failover Clustering: Підтримка кластерів для автоматичного перемикання на резервний сервер у разі збою основного.

- Database Mirroring: Технологія, що забезпечує синхронну або асинхронну реплікацію бази даних на інший сервер для підвищення доступності.

3. Безпека:

- Аутентифікація і авторизація: Підтримка різних методів аутентифікації, включаючи Windows Authentication і SQL Server Authentication, а також ролей і прав доступу на рівні баз даних і об'єктів.

- Шифрування даних: Підтримка шифрування даних як на рівні таблиць (Transparent Data Encryption, TDE), так і на рівні окремих стовпців.

- Audit: Механізми аудиту для відстеження доступу до даних і виконання важливих операцій.

4. Продуктивність і оптимізація:

- Індекси: Підтримка різних типів індексів, таких як кластерні і некластерні, індекси для повнотекстового пошуку, Columnstore Indexes для аналітичних запитів.

- Query Optimizer: Оптимізатор запитів, що автоматично вибирає найефективніший план виконання запиту.

- In-Memory OLTP: Технологія для прискорення обробки транзакцій шляхом зберігання таблиць в оперативній пам'яті.

5. Інтеграція і BI:

- SQL Server Integration Services (SSIS): Інструмент для інтеграції даних, що дозволяє створювати потоки даних для імпорту, експорту і трансформації даних.

- SQL Server Analysis Services (SSAS): Інструмент для аналітики, що дозволяє створювати і керувати багатовимірними базами даних і моделями даних.

- SQL Server Reporting Services (SSRS): Інструмент для створення, управління і розповсюдження звітів.

6. Підтримка різних типів даних:

- Основні типи даних: Підтримка числових, символьних, часових типів даних, а також спеціалізованих типів, таких як XML, JSON, геопросторові дані.

- FileStream: Зберігання великих об'єктів (BLOB) в файловій системі з можливістю доступу через T-SQL.

- Graph Data: Підтримка графових баз даних для моделювання складних взаємозв'язків між даними.

7. Реплікація:

- Snapshot Replication: Розповсюдження даних як статичних знімків.

- Transactional Replication: Підтримка синхронної і асинхронної реплікації транзакцій для забезпечення консистентності даних.

- Merge Replication: Розширена реплікація для об'єднання змін з декількох джерел.

8. Розширюваність і інтеграція:

- CLR Integration: Підтримка інтеграції з .NET Framework для створення користувацьких функцій, збережених процедур і тригерів на мові C# або VB.NET.

- Linked Servers: Можливість доступу до інших СУБД і джерел даних з SQL Server.

9. Хмарні можливості:

- Azure SQL Database: Хмарна версія SQL Server, що надає всі можливості SQL Server у хмарі Microsoft Azure з додатковими функціями для масштабування і високої доступності.

- Azure SQL Managed Instance: Повністю кероване середовище SQL Server з підтримкою повної сумісності з локальною версією SQL Server.

Переваги використання Microsoft SQL Server:

- Надійність і відмовостійкість: Підтримка високої доступності і механізмів відновлення після збоїв.

- Безпека: Вбудовані засоби забезпечення безпеки і шифрування даних.

- Продуктивність: Оптимізація запитів і підтримка індексів для підвищення продуктивності.

- Інтеграція з інструментами BI: Підтримка інструментів для інтеграції даних, аналітики і звітності.

- Хмарні можливості: Гнучкі можливості для розгортання в хмарі і управління базами даних.

Microsoft SQL Server є потужною СУБД, яка забезпечує високий рівень продуктивності, надійності і безпеки для критично важливих бізнес-додатків, а також пропонує широкий спектр інструментів для аналітики і управління даними.

Саме хороша інтеграція з продуктами Microsoft, а саме .NET Framework привели до використання цієї СУБД.

2.2.4 Nginx

Nginx — це потужний і високопродуктивний HTTP-сервер та зворотний проксі-сервер, що також може виконувати роль поштового проксі-сервера та балансувальника навантаження. Спочатку розроблений Ігорем Сисоєвим, Nginx набув популярності завдяки своїй здатності обробляти велику кількість одночасних з'єднань із мінімальними ресурсами. Ось детальний опис основних характеристик Nginx:

Основні характеристики Nginx

1. HTTP-сервер:

- Nginx може обслуговувати статичні файли, такі як HTML, CSS, JavaScript, зображення та інші медіафайли, з високою продуктивністю.

- Підтримка різних протоколів, включаючи HTTP/1.1, HTTP/2 і HTTP/3, що забезпечує швидке та безпечне завантаження сторінок.

2. Зворотний проксі-сервер:

- Nginx може працювати як зворотний проксі для балансування навантаження і кешування, розподіляючи запити між кількома серверами додатків.

- Підтримка SSL/TLS для забезпечення безпечного з'єднання між клієнтом і сервером.

3. Балансувальник навантаження:

- Розподіл вхідного трафіку між декількома серверами для забезпечення рівномірного завантаження і підвищення відмовостійкості.

- Підтримка різних методів балансування, таких як круговий алгоритм (round-robin), least connections (найменше з'єднань) і IP hash (хеш IP).

4. Кешування:

- Nginx може виконувати роль кеш-сервера, зберігаючи копії відповідей для прискорення повторних запитів.

- Підтримка кешування на рівні диск і пам'яті для підвищення продуктивності.

5. Поштовий проксі-сервер:

- Nginx може працювати як поштовий проксі-сервер для протоколів IMAP, POP3 і SMTP, забезпечуючи балансування навантаження та безпеку.

6. Модульна архітектура:

- Nginx має модульну архітектуру, що дозволяє розширювати функціональність через додаткові модулі.

- Можливість підключення сторонніх модулів для розширення можливостей серверу.

7. Підтримка Lua:

- За допомогою модуля ngx_lua, Nginx підтримує вбудовування скриптів на Lua, що дозволяє створювати динамічні конфігурації та обробляти запити на більш глибокому рівні.

8. Висока продуктивність і масштабованість:

- Nginx здатен обробляти сотні тисяч одночасних з'єднань з мінімальними ресурсами, що робить його ідеальним вибором для великих і навантажених веб-додатків.

- Легка масштабованість через додавання нових серверів і балансування навантаження.

Архітектура Nginx

Nginx використовує подієву модель і асинхронну архітектуру, що забезпечує ефективну обробку великої кількості одночасних з'єднань. Це досягається через використання основних процесів (workers), кожен з яких обробляє запити без створення нових потоків або процесів для кожного з'єднання. Це дозволяє зменшити накладні витрати і підвищити продуктивність.

Конфігурація Nginx

Файл конфігурації Nginx зазвичай розташований у /etc/nginx/nginx.conf і складається з кількох блоків:

- main: Основні налаштування для всього сервера.
- http: Налаштування для обробки HTTP-запитів.
- server: Налаштування для конкретного віртуального хоста.
- location: Налаштування для обробки конкретних URI або шляхів.

Приклад базової конфігурації:

```nginx
user www-data;
worker_processes auto;
```

```
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
    worker_connections 1024;
}

http {
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;

    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers on;

    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;

    gzip on;
    gzip_disable "msie6";

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}
```

Переваги використання Nginx

- Висока продуктивність: Ефективна обробка великої кількості одночасних з'єднань.

- Масштабованість: Легка масштабованість за рахунок додавання нових серверів і балансування навантаження.

- Гнучкість: Підтримка широкого спектру конфігурацій та налаштувань.

- Безпека: Підтримка SSL/TLS і захист від різних типів атак.

- Модульність: Можливість розширення функціональності через додаткові модулі.

Використання Nginx у різних сценаріях

1. Веб-сервер:

- Обслуговування статичних файлів.

- Використання як основний веб-сервер для простих сайтів.

2. Зворотний проксі-сервер:

- Розподіл навантаження між декількома серверами додатків.

- Кешування відповідей для прискорення роботи додатків.

3. Балансувальник навантаження:

- Розподіл трафіку для забезпечення рівномірного навантаження на сервери.

- Підвищення відмовостійкості додатків.

4. Поштовий проксі-сервер:

- Балансування і захист поштових серверів.

- Підтримка протоколів IMAP, POP3 і SMTP.

Nginx є універсальним інструментом для обробки веб-трафіку, що забезпечує високу продуктивність, масштабованість і гнучкість для різноманітних сценаріїв використання.

РОЗДІЛ 3.
ПРОГРАМНА РЕАЛІЗАЦІЯ

Для початку розглянемо загальну схеми використаної бази даних

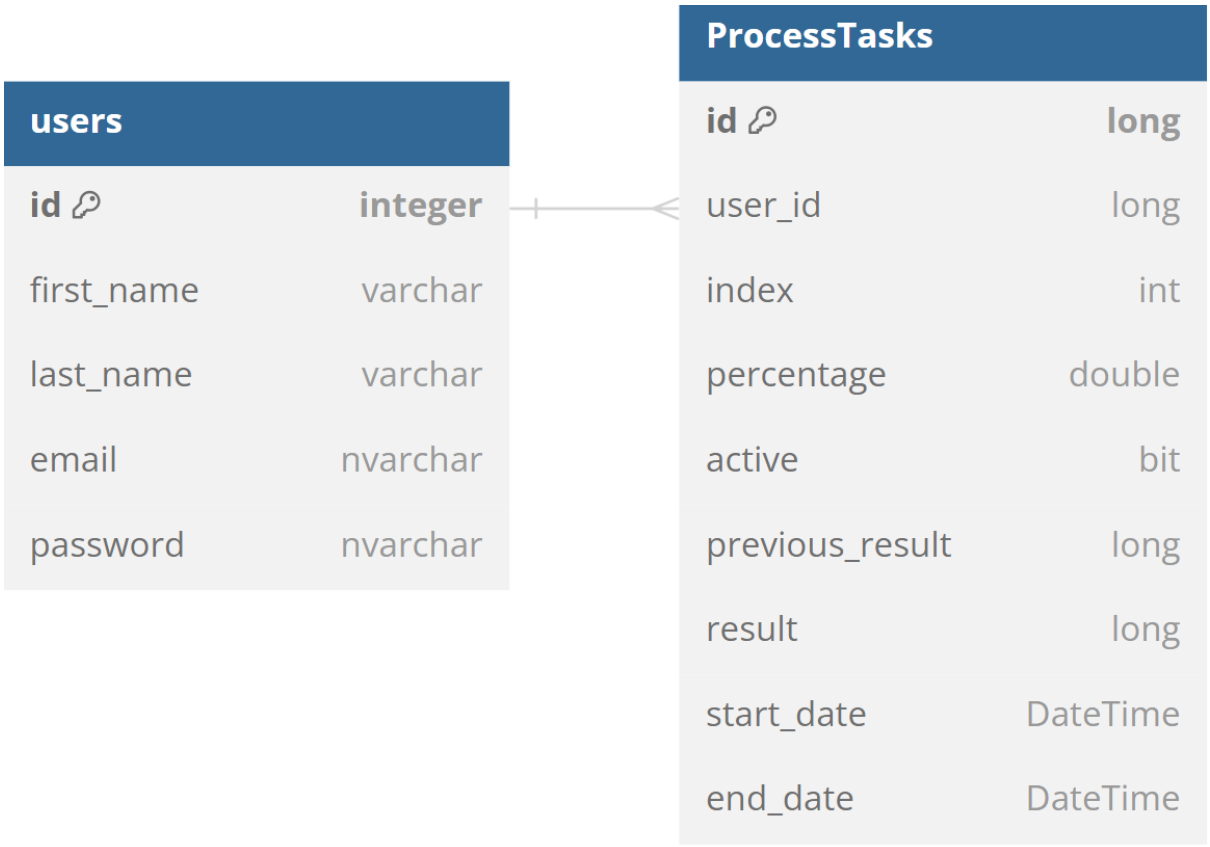

Рис.2.2. Схема бази даних.

На схемі ми бачимо, що дві таблиці з’єднані між собою відношенням «один до багатьох». Це означає, що один користувач може створювати кілька завдань.

3.1. Використання SSL сертифікатів

Самопідписні SSL сертифікати дана програма використовує як на клієнтській частині, так і на сервері. Використання SSL (Secure Sockets Layer) має кілька важливих переваг і функцій:

Для забезпечення конфіденційності і захисту інформації, яка передається через мережу, таку як особисті дані, паролі та інша чутлива інформація, застосовується ***Шифрування Даних***

Ідентифікація Сервера: SSL сертифікати дозволяють серверу підтверджувати свою ідентичність перед клієнтом. Це допомагає уникнути атак типу "Man-in-the-Middle" (MITM), коли зловмисник може спробувати перехопити або змінити комунікацію між клієнтом і сервером.

- ***Довіра Клієнтів***: Використання SSL сертифікату вказує на те, що веб-сайт аутентифікований і захищений, що може збільшити довіру клієнтів до вашого ресурсу.
- ***SEO Підвищення***: Популярні пошукові системи, такі як Google, можуть надавати перевагу веб-сайтам з використанням SSL, піднявши їх в результатах пошуку. Це може бути важливим фактором для покращення видимості сайту в пошукових результатах.
- ***Виключення Загроз Зламу***: Використання SSL допомагає уникнути атак, таких як перехоплення паролів, ін'єкції зловмисного коду, і підвищує загальний рівень безпеки веб-сайту.

3.2. Запуск Nginx та його конфігурація

Перед запуском програми потрібно запустити Nginx для того, щоб запити могли розподілятись та оброблятись декількома серверами. Для коректної роботи Nginx розберемось з його конфігураціями:

```
worker_processes  1;
events {
    worker_connections  1024;
}
http {
    upstream servers {
        least_conn;
```

```
        server localhost:7269;
        server localhost:7268;
        server localhost:7267;
    }
    server {
        listen       443 ssl;
        server_name  localhost;
        ssl_certificate      cert.pem;
        ssl_certificate_key  cert.key;
        ssl_session_cache    shared:SSL:1m;
        ssl_session_timeout  5m;
        ssl_ciphers  HIGH:!aNULL:!MD5;
        ssl_prefer_server_ciphers  on;

        location / {
            proxy_pass https://servers;
        }
    }
}
```

Ця конфігурація Nginx налаштовує сервер з одним робочим процесом *(worker_processes 1)* та визначає параметри для обробки подій *(events)* та кількість можливих з'єднань *(worker_connections 1024)*.

У розділі ***http*** визначено групу серверів *(upstream)*, яка використовує алгоритм вибору сервера з найменшою кількістю активних з'єднань *(least_conn)*. Три сервери налаштовані на локальних адресах і різних портах.

Наступний блок ***server*** визначає сервер, який слухає на захищеному порту 443 за допомогою SSL/TLS *(listen 443 ssl)*. Вказані параметри для сертифікату та приватного ключа (*ssl_certificate* та *ssl_certificate_key*). Також вказані параметри для кешування сесій та криптографічних алгоритмів і налаштувань SSL.

У розділі ***location /*** використовується директива *proxy_pass*, яка направляє HTTP-запити до групи серверів, визначеної в блозі *upstream* як *https://servers*.

При запуску даного веб-застосунку у нас відкривається Swagger для можливості надсилання запитів до серверу без клієнтської частини. А також відкривається front-end на порті 3000.

Спершу ми бачимо сторінку входу (/login).

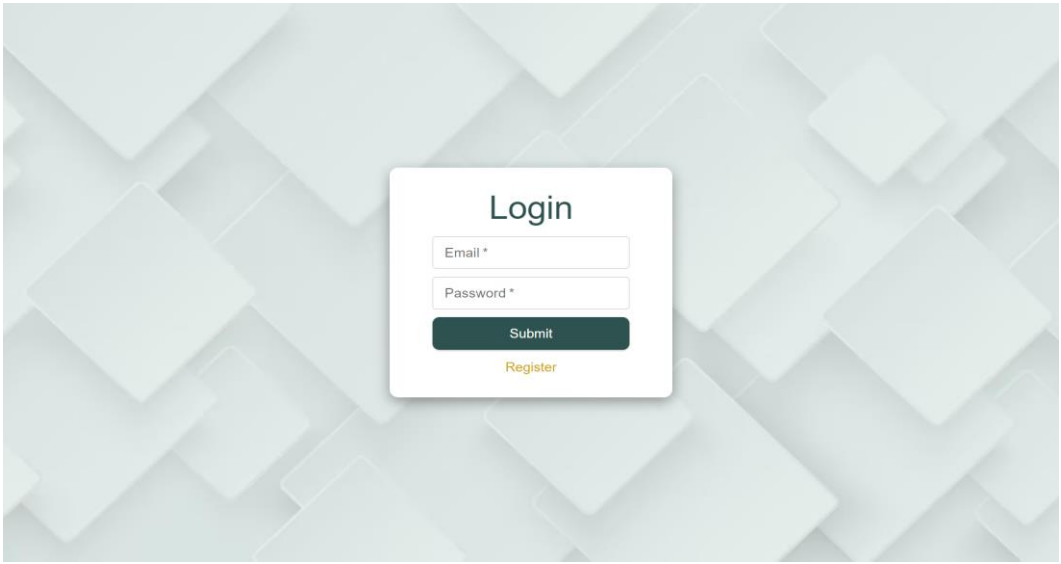

Рис. 3.1. Сторінка входу (логін)

При вході користувача в свій профіль в localStorage веб-сторінки записується JWT Access Token. Тривалість його дії може варіюватись, але загалом бажано використовувати не більше 15 хвилин.

Для оновлення токена доступу (Access Token) використовується токен оновлення (Refresh Token). Зазвичай, якщо токен доступу має дату закінчення терміну дії, після його закінчення користувач повинен повторно автентифікуватися для отримання нового токена доступу. За допомогою токена оновлення цей етап можна пропустити, і з відправленням запиту до API отримати новий токен доступу, який дозволяє користувачеві продовжувати доступ до ресурсів додатка.

Отож ввійшовши в свій профіль, користувач бачить головну сторінку з поточними завданнями. Якщо ж завдань немає, то буде видимим текст “You have no active tasks. Please create them.” (У вас відсутні активні завдання. Будь ласка, створіть їх).

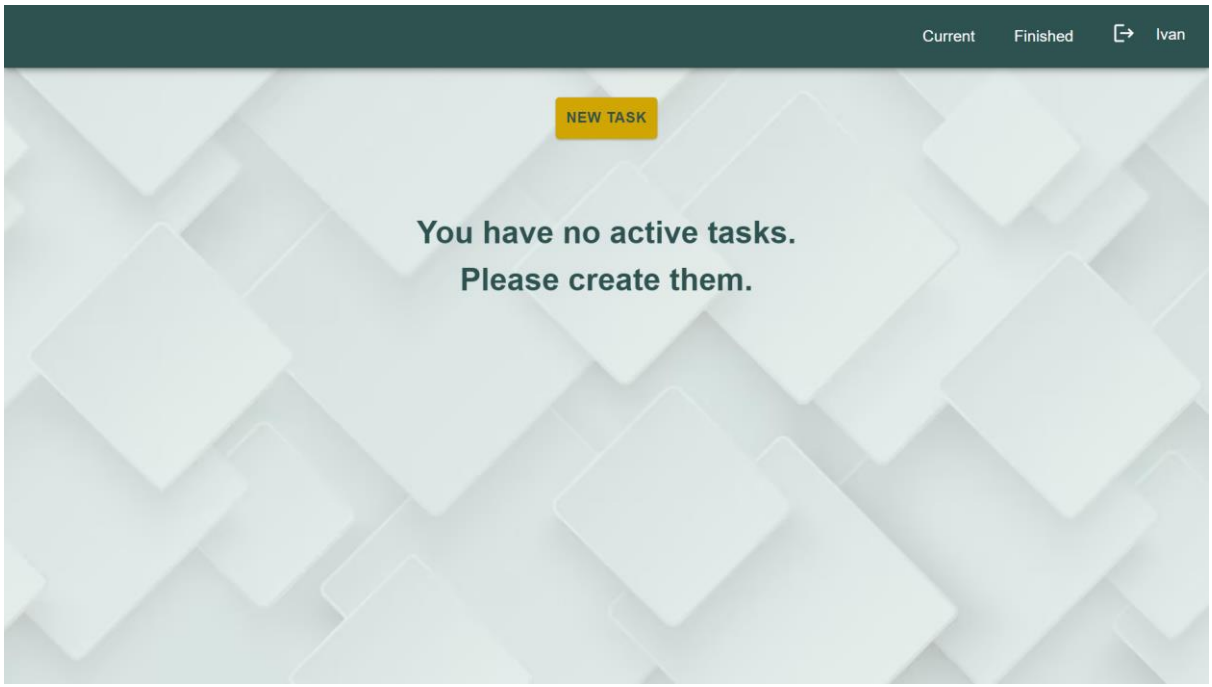

Рис. 3.2 Створення нового завдання

При натисканні на кнопку “New Task” відкривається модальне вікно, у якому потрібно ввести необхідні для створення завдання дані (в цьому випадку лише індекс числа Фібоначчі)

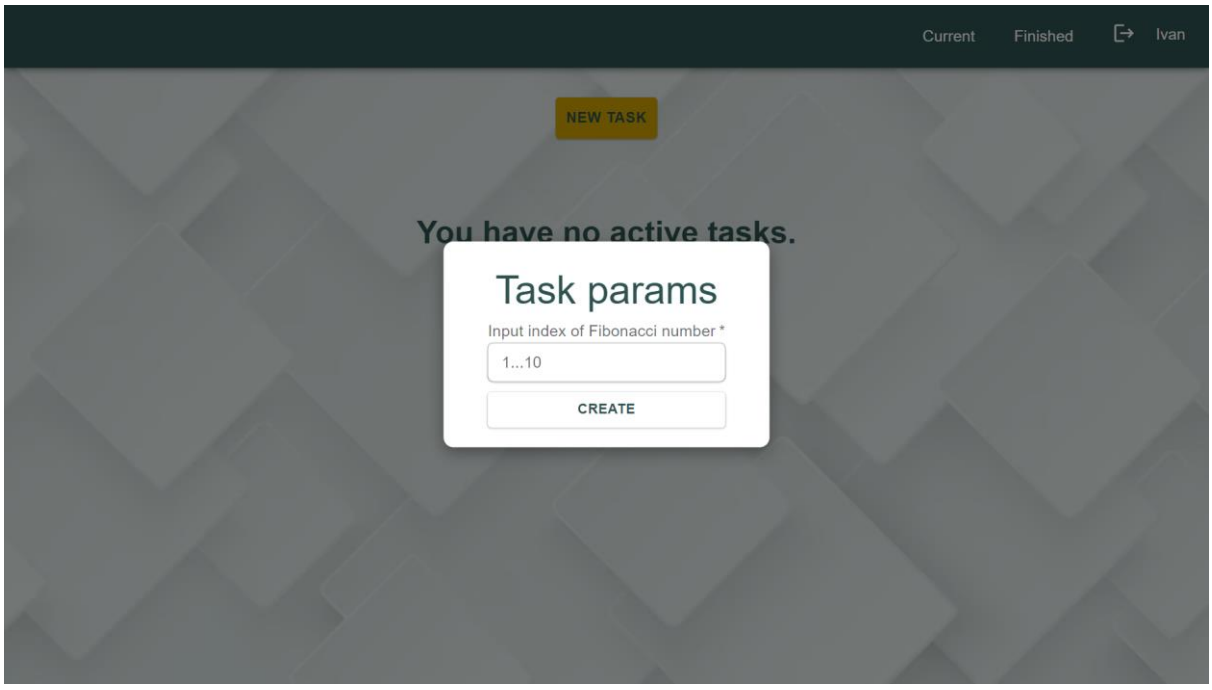

Рис. 3.3 Додавання параметрів для завдання

Після введення коректних даних та натискання кнопки Create, з клієнтської сторони до сервера надсилається запит для створення завдання і через кілька секунд на головній сторінці застосунку з’являється новостворене завдання:

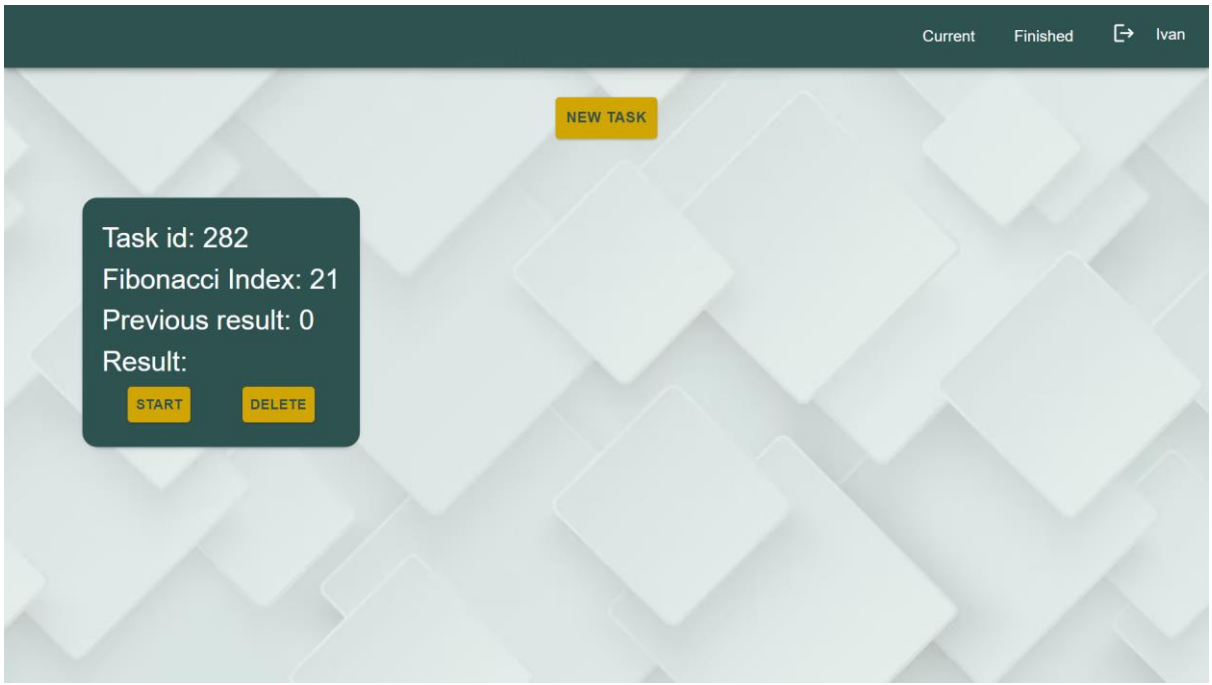

Рис. 3.4. Вигляд перед початком роботи.

Після створення даного завдання ми можемо його розпочати, натиснувши на кнопку Start, щоб мати результат. Або ж є можливість його видалити, натиснувши на кнопку Delete, якщо воно є вже не актуальним або просто не потрібним.

При натисканні на кнопку Start, до сервера (через nginx) надсилається запит для початку обробки завдання. Сервер, з допомогою фреймворку Hangfire, запускає виконання завдання на паралельному потоці і повертає статус Ok(). Тобто зв’язок з сервером закривається за короткий період часу, що забезпечує мінімальний ризик його зламу.

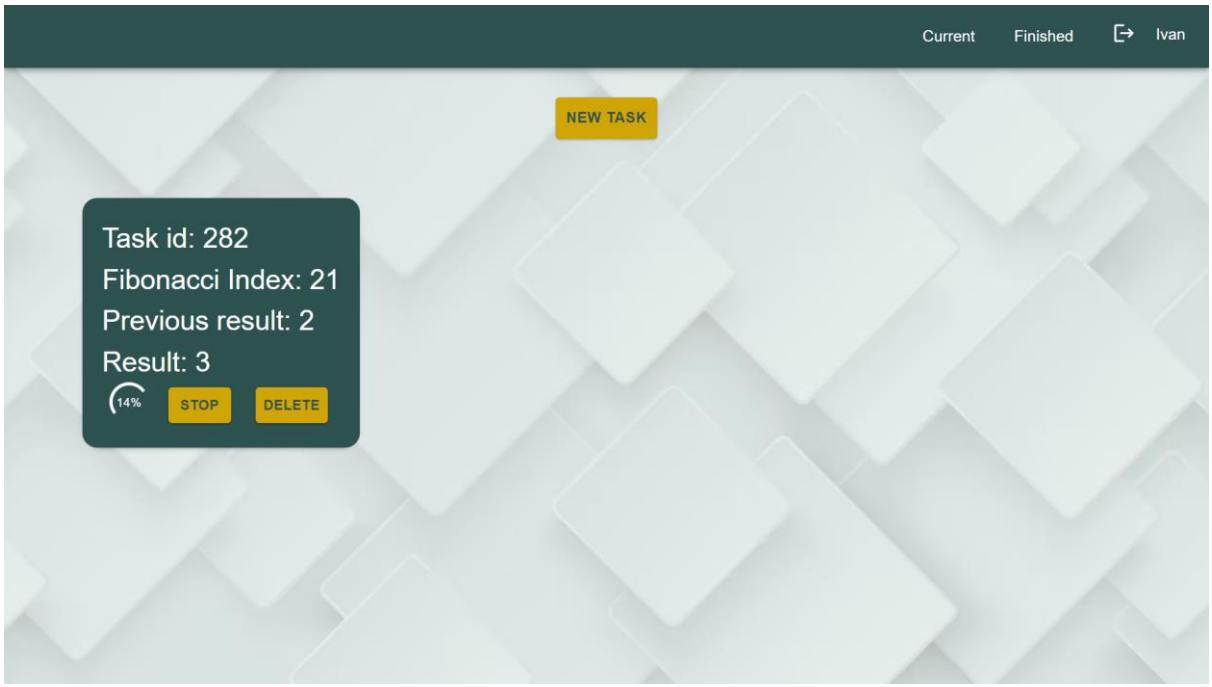

Рис. 3.5. Вигляд поточного завдання з відсотками виконання.

А далі вже за допомогою функції useInterval з фронтенду з певними періодами надсилаються запити для оновлення даних в клієнта. Отримавши дані від сервера, React змінює попередній результат (Previos result), результат (Result), а також прогрес (скільки відсотків роботи вже пророблено).

Також як і в не активного завдання є можливість видалити його (Delete), та звичайно призупинити (Stop). Якщо натиснути на кнопку Stop, на сервер надішлеться запит про зупинку обробки завдання. Сервер зупинить Hangfire.Job, що працювала над цим завданням, та змінить властивість завдання *active* на *false*.

Знову ж таки в нас є можливість видалити завдання або продовжити його обробку з того моменту, де її призупинили.

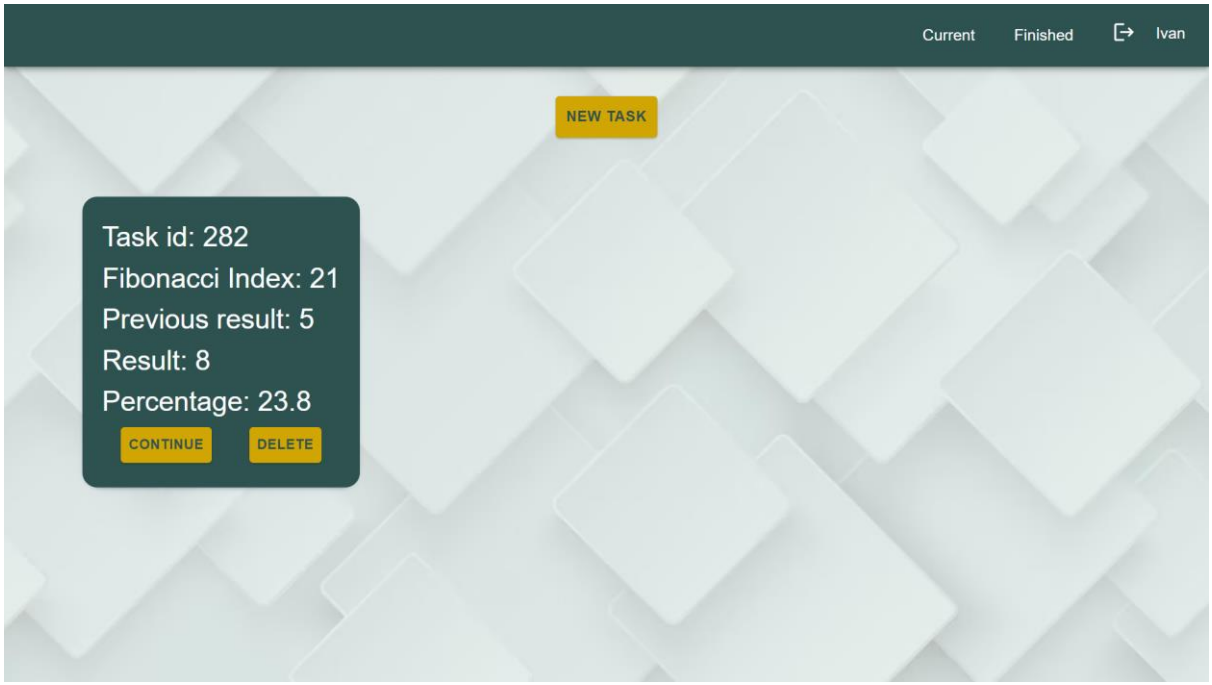

Рис. 3.6. Вигляд призупиненого завдання.

І звичайно дана програма може обробляти декілька завдань водночас, не зменшуючи швидкість відповіді веб-застосунку на дії користувача.

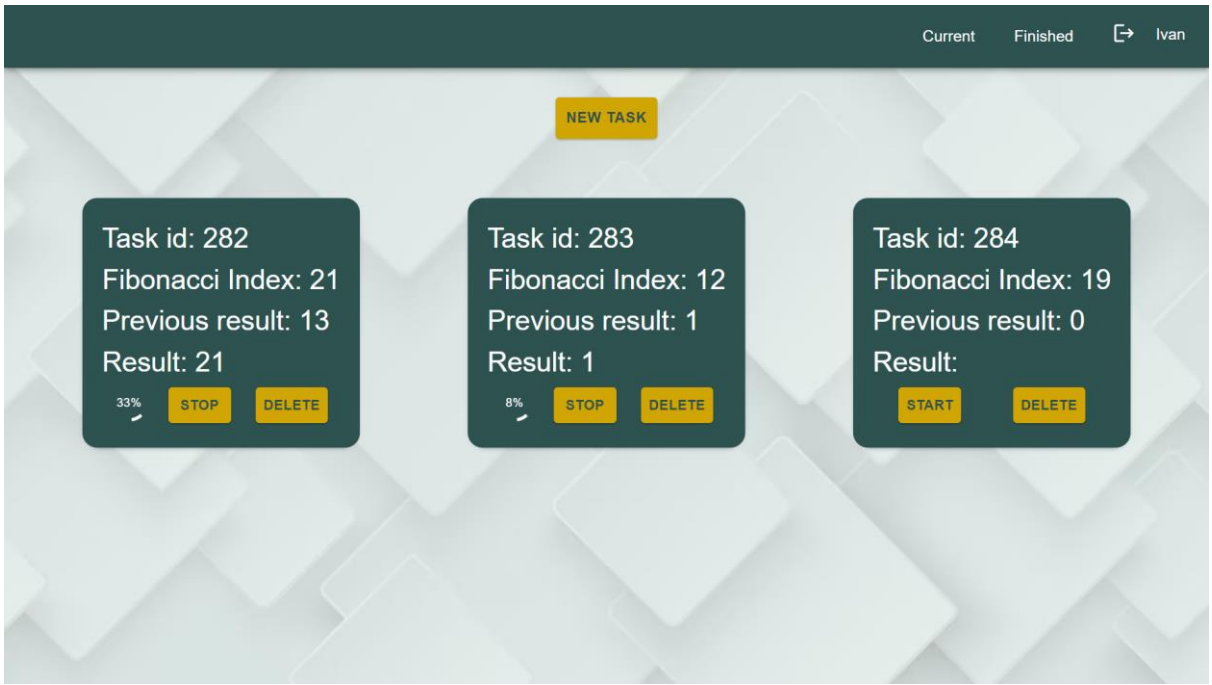

Рис. 3.7. Вигляд всіх завдань, як поточних так і нових.

Також можна перейти на сторінку /finished, клацнувши на навігаційному меню Finished.

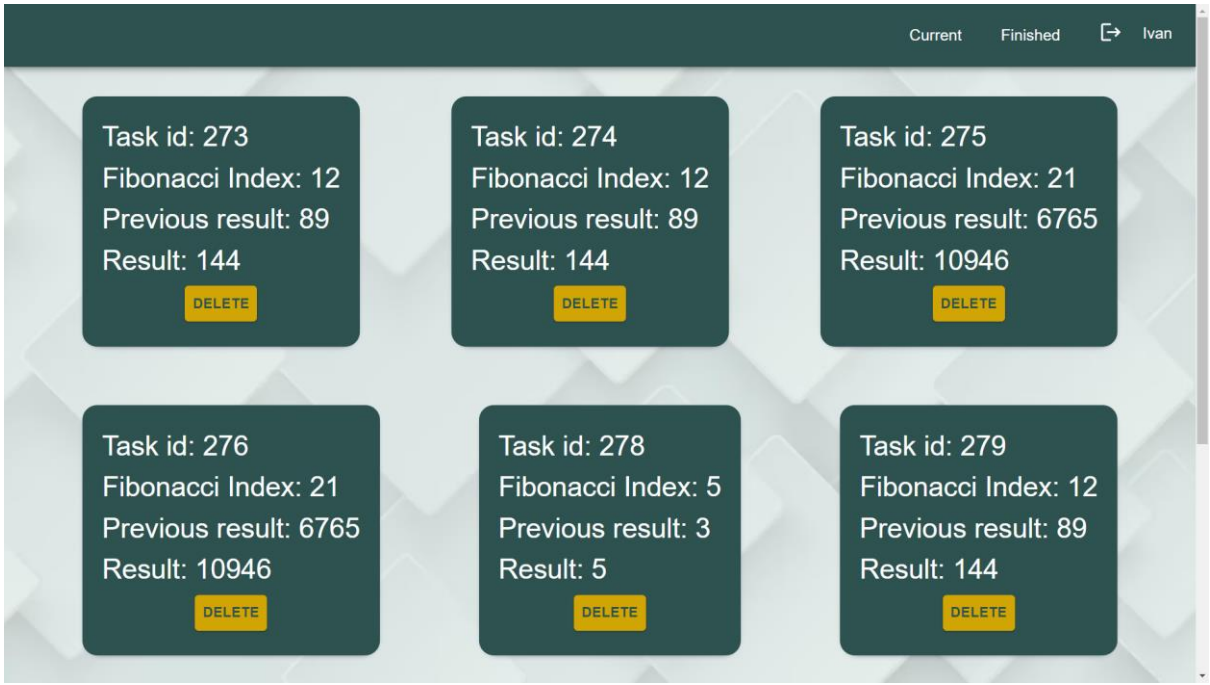

Рис. 3.8. Вигляд вікна закінчених завдань.

На даній сторінці відображені всі завершені завдання користувача. Тут можна їх переглянути або ж видалити вже не потрібні результати.

РОЗДІЛ 4.
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Розробка логіко-імітаційної моделі виникнення травм і аварій

Методикою оцінки рівня небезпеки робочих місць, машин, виробничих процесів та окремих виробництв передбачено пошук об'єктивного критерію рівня небезпеки для конкретного об'єкта. Таким показником вибрана ймовірність виникнення аварії, травми залежно від явища, що досліджується.

Для побудови логіко-імітаційної моделі процесу, формування і виникнення аварії та травми в процесі створення мікрокліматичних умов у приміщенні оцінюють відповідні небезпечні події. Кожній з них присвоїмо ймовірність виникнення:

Шифр	Назва події	Ймовірність
P_1	Відсутність захисного заземлення	0,02
P_2	Пошкодження захисного заземлення	0,04
P_3	Спрацювання складових захисту	0,1
P_4	Неправильна експлуатація захисту	0,02
P_5	Відсутність профілактичних заходів	0,2
P_6	Відсутність захисного щита	0,12
P_7	Недотримання правил вибору взуття	0,15
P_8	Незнання правил техніки безпеки	0,1
P_9	Відсутність засобів індивідуального захисту	0,2
P_{10}	Легковажність	0,08

На основі наведених подій будуємо матрицю логічних взаємозв'язків між окремими пунктами, графічна інтерпретація якої зображено на рис. 5.1.

Розрахуємо ймовірності виникнення подій, що формують логіко-імітаційну модель процесів створення мікрокліматичних умов. Розглянемо травмонебезпечну ситуацію, що виникає за умови роботи працівників із електронебезпекою.

Підставивши дані ймовірностей базових подій у формулу, отримаємо ймовірність події 13: $P_{13} = 0{,}2 + 0{,}4 - 0{,}2 \cdot 0{,}4 = 0{,}0592$.

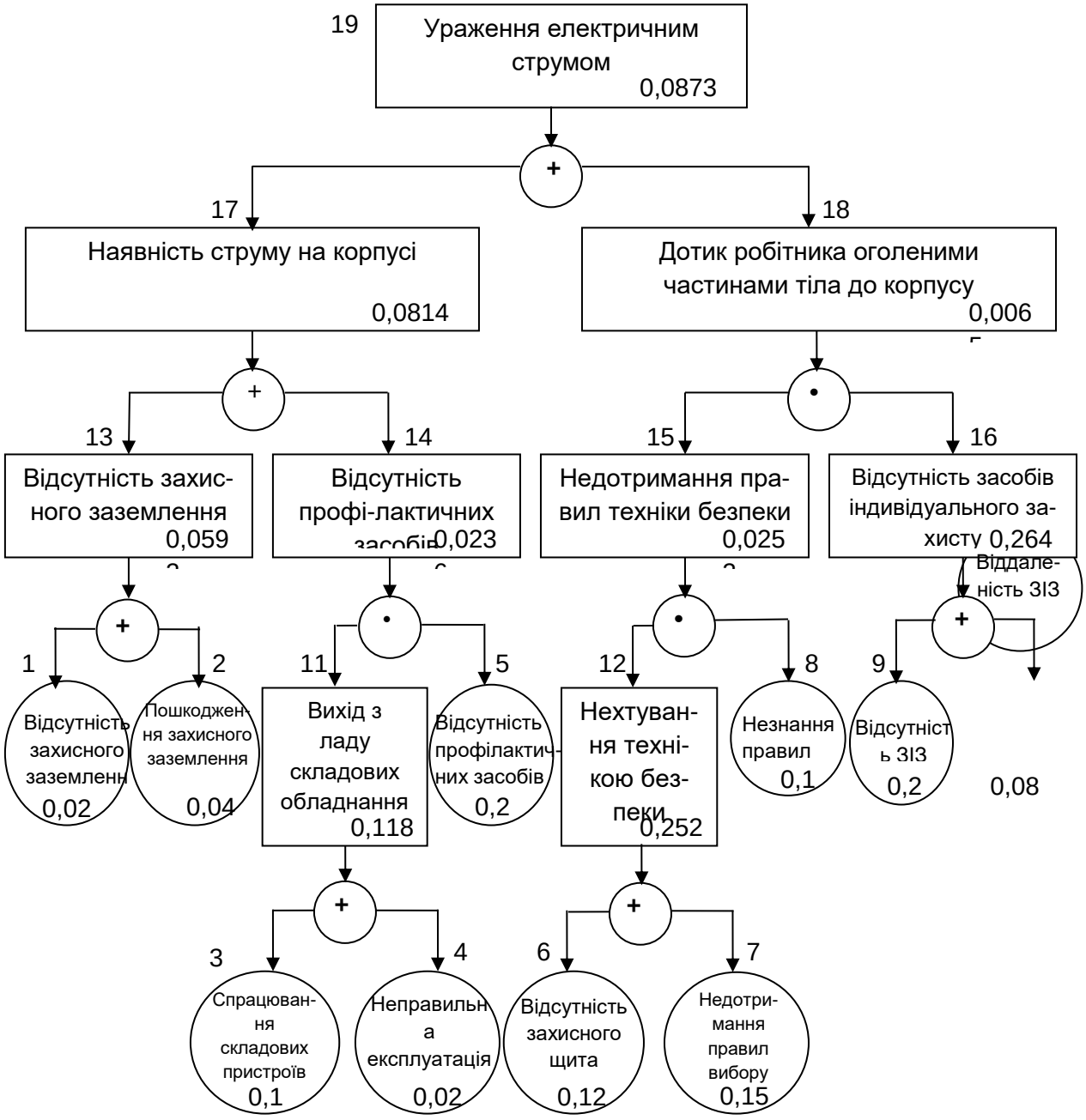

Рис. 4.1. Матриця логічних взаємозв'язків між окремими подіями травмонебезпечної ситуації

Аналогічно визначаємо ймовірність інших подій:

$$P_{11} = P_4 + P_5 - P_4P_5 = 0{,}3 + 0{,}4 - 0{,}3 \cdot 0{,}4 = 0{,}118.$$

$$P_{12} = P_6 + P_7 - P_6P_7 = 0{,}3 + 0{,}5 - 0{,}3 \cdot 0{,}5 = 0{,}252.$$

$$P_{16} = P_9 + P_{10} - P_9P_{10} = 0{,}2 + 0{,}15 - 0{,}2 \cdot 0{,}15 = 0{,}264.$$

$$P_{14} = P_{11} \cdot P_5 = 0{,}118 \cdot 0{,}2 = 0{,}0236.$$

$$P_{15} = P_{12} \cdot P_8 = 0{,}252 \cdot 0{,}1 = 0{,}0252.$$

$$P_{17} = P_{13} + P_{14} - P_{13} \cdot P_{14} = 0{,}592 + 0{,}0236 - 0{,}0592 \cdot 0{,}0236 = 0{,}0814.$$

$$P_{18} = P_{15} \cdot P_{16} = 0{,}264 \cdot 0{,}0252 = 0{,}0065.$$

$$P_{19} = P_{17} + P_{18} - P_{17} \cdot P_{18} = 0{,}0065 + 0{,}0814 - 0{,}0065 \cdot 0{,}0814 = 0{,}0873.$$

Таким чином, ймовірність перекидання машини та наслідкового виникнення травми працівника є досить мала і становить – $P_{19} = 0{,}0873$.

4.2. Планування заходів із покращення умов праці

До заходів щодо покращення умов праці належать всі види діяльності, спрямовані на попередження, нейтралізацію або зменшення негативної дії шкідливих і небезпечних виробничих факторів на працівників.

Рівень умов праці оцінюють порівнянням за фактичними і нормативними значеннями узагальнених (групових) показників.

Заходи щодо поліпшення умов праці здійснюють з метою створення безпечних умов праці шляхом:

- доведення до нормативного рівня показників виробничого середовища за елементами умов праці;
- захисту працівників від дії небезпечних і шкідливих виробничих факторів.

До показників ефективності заходів щодо поліпшення умов праці належать:

а) зміни стану умов праці:

- зміна кількості засобів виробництва, приведених у відповідність до вимог стандартів безпеки праці;
- покращання санітарно-гігієнічних показників;
- покращання психофізичних показників, зменшення фізичних і нервово-психічних навантажень, в т.ч. монотонних умов праці;

- покращання естетичних показників, раціональне компонування робочих місць і впорядкування робочих приміщень;

б) соціальні результати заходів:

- збільшення кількості робочих місць, що відповідають нормативним вимогам;
- зниження рівня виробничого травматизму;
- зменшення кількості випадків професійних захворювань;
- зменшення плинності кадрів через незадовільні умови праці;
- престиж та задоволення працею.

Отже, на покращення охорони праці потрібно виділити кошти на відновлення вентиляційних систем у ремонтних майстернях, естетично оформити приміщення офісу, відновити кабінет з охорони праці, поновити протипожежний інвентар.

4.3. Безпека в надзвичайних ситуаціях

Актуальність проблеми природно-техногенної безпеки для населення і території, зумовлена зростанням втрат людей, що спричиняється небезпечними природними явищами, промисловими аваріями та катастрофами. Ризик надзвичайних ситуацій природного та техногенного характеру невпинно зростає, тому питання захисту цивільного населення від надзвичайних ситуацій на сьогодні є дуже важливе.

У системі цивільної оборони окремого господарства необхідно забезпечити захист населення таким чином:

Укриття в захисних спорудах, якому підлягає усе населення відповідно до приналежності, досягається створенням фонду захисних споруд.

Евакуаційні заходи, які проводяться в містах та інших населених пунктах, які мають об’єкти підвищеної небезпеки, а також у воєнний час, основним способом захисту населення є евакуація і розміщення його у позаміській зоні.

Медичний захист проводиться для зменшення ступеня ураження людей, своєчасного надання допомоги постраждалим та їх лікування, забезпечення епідеміологічного благополуччя в районах надзвичайних ситуацій.

Радіаційний і хімічний захист включає заходи щодо виявлення і оцінки радіаційної та хімічної обстановки, організацію і здійснення дозиметричного та хімічного контролю, розроблення типових режимів радіаційного захисту, забезпечення засобами індивідуального захисту, організацію і проведення спеціальної обробки.

Евакуаційні заходи, які проводяться в містах та інших населених пунктах, які мають об’єкти підвищеної небезпеки, а також у воєнний час, основним способом захисту населення є евакуація і розміщення у позаміській зоні.

ВИСНОВКИ

Оскільки кількість трудомістких обчислень буде тільки зростати, а технічні та програмні характеристики як локальних, так і хмарних серверів будуть дозволяти ширше використовувати паралельні та розподілені обчислення, то дана робота є досить актуальною.

У своїй кваліфікаційній роботі я прагнув, щоб розроблений інтерфейс відповідав вимогам ефективного обслуговування трудомістких запитів.

Ключовим рішенням, що сприяє високій продуктивності та зручності взаємодії, було представлення завдання у вигляді модального вікна, де користувач може змінити параметри завдання не заглиблюючись у структуру коду, а звертаючи увагу на змістовні характеристики завдання.

Важливою перевагою моєї розробки є те, що дана програма може обробляти декілька завдань водночас, не зменшуючи швидкість відповіді веб-застосунку на дії користувача.

Напевно мою роботу можна буде розширити на кілька модулів для спрощення заміни типу трудомістких обчислень, але це на перспективу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Alex Banks. Learning React: Functional Web Development with React and Redux. 1st Edition. O'Reilly Publishing, 2017. 348p.
2. Carl-Hugo Marcotte, Nick Cosentino. Architecting ASP.NET Core Applications - Third Edition: An atypical design patterns guide for .NET 8, C# 12, and beyond. 3rd ed. Edition. Packt Publishing, 2024. 806p.
3. Derek DeJonghe. NGINX Cookbook, 2nd Edition. O'Reilly Media, 2022. 202 с.
4. F5 NGINX Plus: Load Balancing. URL: https://www.nginx.com/products/nginx/load-balancing/ (дата звернення: 14.04.2024).
5. How Google Cloud Platform works.Google Cloud Computing. URL: https://cloud.google.com (дата звернення: 11.04.2024).
6. Jaydeep Patil. Hangfire Introduction and Implementation in .NET Core 6 Web API. URL: https://medium.com/@jaydeepvpatil225/hangfire-introduction-and-implementation-in-net-core-6-web-api-31acfe6c60f1 (дата звернення: 11.04.2024).
7. Joseph Albahari. C# 10.0 in a Nutshell: The Definitive Reference. O'Reilly Media, 2022. - 1058 с.
8. Mark J. PriceC. # 12 and .NET 8 – Modern Cross-Platform Development Fundamentals: Start building websites and services with ASP.NET Core 8, Blazor, and EF Core 8 8th ed. Edition, Packt Publishing, 2023. 826p.
9. React JS Notes for Professionals. Goalkicker.com, 2018. 110 с.
10. React: JavaScript-бібліотека для створення користувацьких інтерфейсів. URL: https://uk.legacy.reactjs.org/ (дата звернення: 14.04.2024).
11. Riccardo Terrell. Concurrency in .NET: Modern patterns of concurrent and parallel programming. Manning, 2018. 568 с.
12. Robin Wieruch. The Road to React: The React.js with Hooks in JavaScript Book. O'Reilly Publishing, 2024. 484p.
13. Samer Buna. React Succinctly. Syncfusion, 2016. 102 с.
14. Stephen Cleary. Concurrency in C# Cookbook: Asynchronous, Parallel, and Multithreaded Programming, 2nd Edition. O'Reilly Media, 2014. 251 с.

15. Swati Gupta. An Approach For Concurrency Control Algorithm In Real-Time Database. LAP Lambert Academic Publishing, 2019. 64 с.

16. Will Reese. Nginx: the High-Performance Web Server and Reverse Proxy. URL: https://www.linuxjournal.com/article/10108 (дата звернення: 20.04.2024).

17. Відкрита JS бібліотека React? URL: https://web-developer.in.ua/assets/articles/react/react-js-library/react-js-library.html (дата звернення: 18.04.2024).

18. Е. Лок, ASP.NET в дії. Фоліо: Харків, 2021. 320с.

19. Платформа як послуга. URL: http://www.softline.kiev.ua/ua/khmarni-poslugi/poslugi-u-khmarakh/platforma-yak-posluga.html (дата звернення: 10.04.2024).

20. Сергій Бондаренко. Що таке ASP.NET? Принцип функціонування та моделі розробки. URL: https://highload.today/uk/shho-take-asp-net-printsip-funktsionuvannya-ta-modeli-rozrobki/ (дата звернення: 10.04.2024).

21. Хмарні технології. URL: http://j.parus.ua/ua/358/ (дата звернення: 18.04.2024).

22. Що таке React JS? Як почати вивчати Реакт? Навички для react developer. URL: https://cases.media/article/sho-take-react-js-yak-pochati-vivchati-reakt-navichki-dlya-react-developer (дата звернення: 16.04.2024).

ДОДАТКИ:
ПРИКЛАДИ КОДУ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ВЕБ ЗАСТОСУНКУ

Додаток А.
Контролер авторизації

```
[AllowAnonymous]
        [HttpPost(Name = "Login")]
        public async Task<IActionResult>
LoginController([FromBody] UserLoginDto login)
        {
            IActionResult response = Unauthorized();
            var user = await _context.Users.FirstOrDefaultAsync(u
=> u.Password == Hashing.getHash(login.Password) && u.Email ==
login.Email);
            if (user != null)
            {
                var tokenString =
JwtUtils.GenerateJSONWebToken(_config, user);
                response = Ok(new { token = tokenString });
            }
            return response;
        }
```

Додаток Б.
Приклад контролера завдань (Task controller.cs)

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using WebServer.DTO;
using WebServer.Models;
using WebServer.Services;
using Hangfire;
using System.Net.Http.Headers;


namespace WebServer.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class TaskController : ControllerBase
    {
        private readonly ILogger<TaskController> _logger;
        private readonly ApiDbContext _context;
        private IProcessService _processService;
        private static Dictionary<long?, CancellationTokenSource> cts =
new Dictionary<long?, CancellationTokenSource>();

        public TaskController(ILogger<TaskController> logger,
ApiDbContext context, IProcessService processService)
        {
            _logger = logger;
            _context = context;
            _processService = processService;
        }

        [HttpGet(Name = "GetUserTasks")]
        [Authorize]
        public async Task<ActionResult> GetAllUserTaskController(bool?
done)
        {
            var claimsIdentity = User.Identity as ClaimsIdentity;
            if (claimsIdentity != null)
            {
                var userId =
long.Parse(claimsIdentity.FindFirst(ClaimTypes.NameIdentifier)?.Value);
                var response = _context.ProcessTasks.Where(e => e.UserId
== userId).ToList();
                if (done != null && response != null)
                {
                    if (done == true)
                    {
                        response = response.Where(e => e.percentage ==
100).ToList();
                    }
                    else
                    {
```

```
                        response = response.Where(e => e.percentage <
100).ToList();
                    }
                }
                return Ok(response);
            }
            return BadRequest();
        }
        [Authorize]
        [HttpPut(Name = "StartTask")]
        public async Task<IActionResult>
StartTaskController(StartTaskParams taskParams)
        {
            cts[taskParams.TaskId] = new CancellationTokenSource();

            var i = BackgroundJob.Enqueue(() =>
_processService.StartProcess(taskParams.TaskId,
cts[taskParams.TaskId].Token));

            return Ok(i);
        }

        [Authorize]
        [HttpPatch("StopTask/", Name = "StopTask")]
        public async Task<ActionResult> StopTaskController(long? id)
        {
           var origin = HttpContext.Request.Headers["Origin"];

           if (cts.ContainsKey(id))
           {
               cts[id].Cancel();
               cts[id].Dispose();
               cts.Remove(id);
               return Ok("Task removed");
           }
           else if (origin == "https://localhost:3000")
           {
               using (HttpClient client = new HttpClient())
               {
                   client.Timeout = TimeSpan.FromSeconds(10);
                   try
                   {
                       string apiUrl =
$"https://localhost:7268/Task/StopTask/{id}";
                       var content = new StringContent("");
                       if
(HttpContext.Request.Headers.TryGetValue("Authorization", out var
authorizationHeaderValue))
                       {
                           client.DefaultRequestHeaders.Authorization =
new AuthenticationHeaderValue(authorizationHeaderValue);
                       }
                       await client.PatchAsync(apiUrl, content);
                       return Ok();
                   }
                   catch (HttpRequestException ex)
                   {
                       return BadRequest();
```

```
                }
            }
        }
        else
        {
            return BadRequest();
        }
    }

    [Authorize]
    [HttpGet("{id}", Name = "GetTask")]
    public async Task<IActionResult> GetTaskController(long? id)
    {
        var task = await _context.ProcessTasks.FirstOrDefaultAsync(x
=> x.Id == id);
        if (task != null)
        {
            return Ok(task);
        }
        return BadRequest();
    }

    [HttpPost(Name = "CreateProcessTask")]
    [Authorize]
    public async Task<ActionResult> CreateTaskController(TaskParams
taskParams)
    {
        var claimsIdentity = User.Identity as ClaimsIdentity;

        if (claimsIdentity != null)
        {
            var userId =
long.Parse(claimsIdentity.FindFirst(ClaimTypes.NameIdentifier)?.Value);
            var sender = await _context.Users.FirstOrDefaultAsync(u
=> u.Id == userId);
            ProcessTask activeProcess = new ProcessTask()
            {
                index = taskParams.Index,
                percentage = 0,
                active = false,
                UserId = userId,
                User = sender
            };
            await _context.ProcessTasks.AddAsync(activeProcess);
            _context.SaveChanges();
            return Ok();
        }
        return BadRequest("User validation error.");
    }
    [HttpDelete("{id}")]
    [Authorize]
    public async Task<IActionResult> DeleteProcessTask(long? id)
    {
        var processTask = await _context.ProcessTasks.FindAsync(id);

        if (processTask == null)
        {
            return NotFound();
```

```
            }

            _context.ProcessTasks.Remove(processTask);
            _context.SaveChanges();

            return NoContent(); // Return a 204 No Content response to
indicate successful deletion.
        }
    }
}
```

Додаток В.
Форма для реєстрації написана на TypeScript (фронтенд)

```
import React, { useState } from "react";
import './userFormComponentStyle.css';
import { Form, Link, useNavigate } from "react-router-dom";
import { Button, FormControl } from "@mui/base";
import { Snackbar, TextField } from "@mui/material";
import axios from "axios";
import { GeneralURL } from "../../index";

export enum userFormType {
    login,
    register
}

export default function UserForm(props: { formType: userFormType }) {

    const navigate = useNavigate();
    const [email, setEmail] = useState<string>('');
    const [password, setPassword] = useState<string>('');
    const [firstName, setFirstName] = useState<string>('');
    const [lastName, setLastName] = useState<string>('');
    const [open, setOpen] = useState<boolean>(false);

    const [emailError, setEmailError] = useState<boolean>(false);
    const [passwordError, setPasswordError] = useState<boolean>(false);
    const [firstNameError, setFirstNameError] = useState<boolean>(false);
    const [lastNameError, setLastNameError] = useState<boolean>(false);

    const nameRegexPatter = new RegExp("^[a-zA-Z0-9]*$");
    const emailRegexPatter = new RegExp("^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-
]+\\.[a-zA-Z]{2,}$");
    const passwordRegexPatter = new RegExp("^(?=.*[A-Za-z])(?=.*\\d)[A-
Za-z\\d]{8,}$");

    const HandleEmailChange = (e: React.ChangeEvent<HTMLInputElement>) =>
{
        setEmail(e.target.value);
        if (!emailRegexPatter.test(email)) {
            setEmailError(true);
        }
        else {
            setEmailError(false)
        }
    }

    const HandlePasswordChange = (e: React.ChangeEvent<HTMLInputElement>)
=> {
        setPassword(e.target.value);
        if (!passwordRegexPatter.test(password)) {
            setPasswordError(true);
        }
        else {
            setPasswordError(false)
        }
```

```
    }

    const HandleFirstNameChange = (e:
React.ChangeEvent<HTMLInputElement>) => {
        setFirstName(e.target.value);
        if (!nameRegexPatter.test(firstName)) {
            setFirstNameError(true);
        }
        else {
            setFirstNameError(false)
        }
    }

    const HandleLastNameChange = (e: React.ChangeEvent<HTMLInputElement>)
=> {
        setLastName(e.target.value);
        if (!nameRegexPatter.test(lastName)) {
            setLastNameError(true);
        }
        else {
            setLastNameError(false)
        }
    }

    const handleLoginSubmit = async () => {
        try {
            const response = await
axios.post(`${GeneralURL}/Authentication`, {
                email: email,
                password: password
            });
            localStorage.setItem("jwtToken", response.data.token);

            if (localStorage.getItem !== null) {
                navigate("/");
            }
        } catch (error) {
            console.log(error);
            setOpen(true);
        }
    }

    const handleRegisterSubmit = async () => {
        try {
            const response = await
axios.post(`${GeneralURL}/Registration`, {
                firstName: firstName,
                lastName: lastName,
                email: email,
                password: password
            });
            localStorage.setItem("jwtToken", response.data.token);
        } catch (error) {
            console.log(error);
            setOpen(true);
        }
        if (localStorage.getItem != null) {
            navigate("/");
```

```
        }
    }

    const handleClose = (event: React.SyntheticEvent | Event) => {
        setOpen(false);
    };

    return (
        <div className="login-container">
            <Form className="login-form" onSubmit={props.formType ===
userFormType.login ? handleLoginSubmit : handleRegisterSubmit}>
                <span className="form-title">{props.formType ===
userFormType.login ? "Login" : "Register"}</span>
                {
                    props.formType == userFormType.register &&
                    <>
                        <FormControl required className="login-element">
                            <TextField
                                error={firstNameError}
                                onChange={HandleFirstNameChange}
                                required
                                placeholder=""
                                className="input-box"
                                size="small"
                                id="outlined-error"
                                label="First name"
                            />
                        </FormControl>
                        <FormControl required className="login-element">
                            <TextField
                                error={lastNameError}
                                onChange={HandleLastNameChange}
                                required
                                placeholder=""
                                className="input-box"
                                size="small"
                                id="outlined-error"
                                label="Last name"
                            />
                        </FormControl>
                    </>
                }
                <FormControl required className="login-element">
                    <TextField
                        error={emailError}
                        onChange={HandleEmailChange}
                        required
                        placeholder="example@gmail.com"
                        className="input-box"
                        size="small"
                        id="outlined-error"
                        autoComplete="off"
                        label="Email"
                    />
                </FormControl>
                <FormControl required className="login-element">
                    <TextField
                        type="password"
```

```
                        error={passwordError}
                        onChange={HandlePasswordChange}
                        required
                        placeholder="Password"
                        className="input-box"
                        size="small"
                        id="outlined-error"
                        label="Password"
                    />
                </FormControl>
                <Button type='submit'>
                    Submit
                </Button>
                <Link className="register-link login-element"
                    to={props.formType === userFormType.login ?
"/register" : "/login"}>
                    {props.formType === userFormType.login ? "Register" :
"Log in"}
                </Link>
                <Snackbar
                    open={open}
                    onClose={handleClose}
                    autoHideDuration={4000}
                    message="Wrong user data!"
                />
            </Form>
        </div>
    );
}
```

Додаток Г.
Контролер реєстрації

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using WebServer.Models;
using WebServer.Utils;
using WebServer.Validation;

namespace WebServer.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class RegistrationController : ControllerBase
    {
        private readonly ILogger<AuthenticationController> _logger;
        private readonly ApiDbContext _context;
        private readonly IConfiguration _config;

        public RegistrationController(ILogger<AuthenticationController>
logger, ApiDbContext context, IConfiguration config)
        {
            _logger = logger;
            _context = context;
            _config = config;
        }

        [HttpGet(Name = "GetUsers")]
        [AllowAnonymous]
        public async Task<ActionResult<IEnumerable<User>>> GetUsers()
        {
            return Ok(await _context.Users.ToListAsync());
        }

        [HttpPost(Name = "Register")]
        [AllowAnonymous]
        public async Task<ActionResult> Register(User user)
        {
            if (ValidationManager.UserIsValid(user))
            {
                user.Password = Hashing.getHash(user.Password);
                await _context.Users.AddAsync(user);
                await _context.SaveChangesAsync();
                var tokenString = JwtUtils.GenerateJSONWebToken(_config,
user);
                return Ok(new { token = tokenString });
            }
            return BadRequest("User validation error.");
        }
    }
}
```

Додаток Д.
Сервіс самого виконання завдання

```
using Microsoft.EntityFrameworkCore;
using WebServer.Models;
using System.Diagnostics;

namespace WebServer.Services
{
    public interface IProcessService
    {
        Task StartProcess(long? taskId, CancellationToken cancel);
    }
    public class ProcessService : IProcessService
    {
        private readonly ApiDbContext _context;
        private int _result;
        private int _prev_result;
        private float _percentage;
        public ProcessService(ApiDbContext context)
        {
            _context = context;
            _prev_result = 0;
            _result = 1;
            _percentage = 0;
        }
        public async Task StartProcess(long? taskId, CancellationToken
cancel)
        {
            if (taskId != null)
            {
                var task = await
_context.ProcessTasks.FirstOrDefaultAsync(i => i.Id == taskId);
                if (task?.percentage < 100)
                {
                    task.startDate = DateTime.Now;
                    task.active = true;
                    await _context.SaveChangesAsync();
                    if (task?.percentage > 0)
                    {
                        _percentage = (int)task.percentage;
                        _prev_result = (int)task.previousResult;
                        _result = (int)task.result;
                    }

                    for (int i = 1 + (int)(_percentage / 100 *
task.index); i < task.index; i++)
                    {
                        task = await
_context.ProcessTasks.FirstOrDefaultAsync(i => i.Id == taskId);
                        if (!cancel.IsCancellationRequested &&
task?.active == true && task?.percentage < 100)
                        {
                            await Task.Delay(5000); ;
                            int help = _result + _prev_result;
                            _prev_result = _result;
                            _result = help;
                            task.percentage += 100.0 / task.index;
```

```
                        task.result = _result;
                        task.previousResult = _prev_result;
                        await _context.SaveChangesAsync();
                    }
                    else
                    {
                        task.active = false;
                        await _context.SaveChangesAsync();
                        await Task.CompletedTask;
                        cancel.ThrowIfCancellationRequested();
                        return;
                    }
                }
                await Task.Delay(5000);

                task.percentage = 100;
                task.active = false;
                task.endDate = DateTime.Now;
                task.result = _result;
                task.previousResult = _prev_result;
                await _context.SaveChangesAsync();
                await Task.CompletedTask;
            }
        }
    }
}
```

Додаток Е.
Головна сторінка (дашборд, фронтенд)

```
import { NavLink } from "react-router-dom";
import jwt_decode from "jwt-decode";
import { AppBar, Box, Toolbar } from "@mui/material";
import "./mainComponentStyle.css";
import LogoutIcon from "@mui/icons-material/Logout";
import CurrentTaskList from
"./TaskList/CurrentTaskList/currentTaskListComponent";
import FinishedTaskList from
"./TaskList/FinishedTaskList/finishedTaskListComponent";

export enum TaskListType {
  current,
  finished
}

export default function Root(props: Readonly<{ listType: TaskListType }>)
{
  const token = localStorage.getItem("jwtToken");

  const tokenClaims = token? jwt_decode<{ name: string }>(`${token}`) :
undefined;

  return (
    <div className="menu-container">
      <Box sx={{ flexGrow: 1 }}>
        <AppBar position="static">
          <Toolbar className="navbar">
              <NavLink to="/" className="nav-button">
                Current
              </NavLink>
              <NavLink to="/finished" className="nav-button">
                Finished
              </NavLink>

            <NavLink className="nav-button" to="/login"onClick={() => {
localStorage.removeItem("jwtToken") }}>
              <LogoutIcon sx={{color:'white'}}/>
            </NavLink>
            <div className="name-info">
              {token && `${tokenClaims?.name}`}
            </div>

          </Toolbar>
        </AppBar>

        {props.listType == TaskListType.current ?
            <CurrentTaskList /> : <FinishedTaskList />
        }
      </Box>
    </div>
  );
}
```