Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Гордієнко Олександра Юрійовича*
(ПІБ)

академічної групи *121-19ск-1*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Розробка веб-додатка порівняння двох документів формату "txt" засобами React.js та веб-API ASP.NET*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*проф. Бердник М.Г.*			
розділів:				
спеціальний	*проф. Бердник М.Г.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент	*доц.Шедловський І.А.*			
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2022

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2022 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-19ск-1* *Гордієнко Олександра Юрійовича*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка веб-додатка порівняння двох документів формату "txt" засобами React.js та веб-API ASP.NET*

затверджена наказом ректора НТУ «ДП» від 18.05.2022 р. № 268-с

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2022 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПО й тривалості його розробки***	***27.05.2022 р.***

Завдання видав _____ *проф. Бердник М.Г.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Гордієнко О.Ю.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2022 р.*

Термін подання кваліфікаційної роботи до ЕК: *13.06.2022 р.*

РЕФЕРАТ

Пояснювальна записка: 84 с., 47 рис., 6 табл., 2 дод., 26 джерел.

Об'єкт розробки: веб-додаток порівняння двох документів формату «txt» засобами React.js та веб-API ASP.NET.

Мета кваліфікаційної роботи: створення програмного забезпечення призначеного спростити та пришвидшити процес пошуку відмінностей між двома варіантами тексту та надання зрозумілої та наглядної інформації стосовно того де саме знаходиться відмінності.

У вступі розглядаються та обґрунтовуються: аналіз та загальний стан проблеми; мета кваліфікаційної роботи; актуальность теми; постановка завдання.

У першому розділі проводиться дослідження предметної області, існуючих проблем та рішень, розроблюється формулювання завдання.

У другому розділі обираються та описуються: середовище розробки, мова програмування та допоміжні інструменти розробки; проєктування програми; алгоритми та структури функціонування системи; вхідні та вихідні дані; графічний інтерфейс та робота програми.

В економічному розділі визначається: трудомісткість розробленої інформаційної системи; вартість роботи та час на створення програмого забезпечення.

Список ключових слів: ВЕБ-ДОДАТОК, КЛІЄНТ-СЕРВЕР, ТЕКСТ, ДОКУМЕНТ, ГРАФІЧНИЙ ІНТЕРФЕЙС, АВТОРИЗАЦІЯ, БАЗА ДАНИХ, СХОВИЩЕ ДАНИХ, КОМПОНЕНТ.

ABSTRACT

Explanatory note: 84 pages, 47 figures, 6 tables, 2 appendices, 26 sources.

Development object: web application to compare two "txt" documents using React.js and ASP.NET.

The purpose of the qualification work: to create software designed to simplify and accelerate the process of finding differences between the two versions of the text and to provide clear and visual information about where the differences are.

The introduction discusses and justifies: the analysis and the general state of the problem; the purpose of the qualification work; the relevance of the topic; problem statement.

The first chapter conducts a study of the subject area, existing problems and solutions, develops a formulation of the problem.

The second section selects and describes: development environment, programming language and development tools; program design; algorithms and structures of system operation; input and output data; graphical interface and program operation.

The economic section determines: the labor intensity of the developed information system; the cost of work and time to create the software.

Keyword list: WEB APPLICATION, CLIENT-SERVER, TEXT, DOCUMENT, GRAPHIC INTERFACE, AUTHORIZATION, DATABASE, DATA STORAGE, COMPONENT.

ЗМІСТ

ВСТУП

Зазвичай люди, навіть іноді цього не помічаючи, порівнює речі у побуті, це може бути порівняння товару у магазині, щоб зробити вигіднішу покупку; зроблені витрати за поточний з минулим тижнем; порівнюючи свої вчинки з іншими, тобто люди доволі часто порівнють щось.

Також порівняння використовують у роботі, наприклад, порівняння двох версій документів на встановлення актуальнішої.

Доволі часто такі порівняння робляться вручну та ведуть за собою ряд труднощів:

– витрачення часу та сил на опрацювання всіх вхідних матеріалів, фіксацію відмінностей та осмислення результату;

– ймовірність допустити помилку;

– чим більше обсяг матеріалу для опрацювання, тим більше на це знадобиться часу, сил та вірогідність допустити помилку збільшується.

Усе вищезазначене потребую людських ресурсів, а отже з'являється привід автоматизувати даний процес та замість того, щоб вручну шукати відмінності між двома текстами, це можна зробити за допомогою комп’ютеру за лічені секунди та отримати наглядний результат порівняння.

З огляду на це мною було обрано тему кваліфікаційної роботи: «Розробка веб-додатка порівняння двох документів формату "txt" засобами React.js та веб-API ASP.NET».

Кваліфікаційна робота має призначення спростити та пришвидшити процес пошуку відмінностей між двома варіантами тексту та надання зрозумілої та наглядної інформації стосовно того де саме знаходиться відмінності.

Обрання даних технологій дає змогу чітко розмежувати реалізацію даної роботи на окремі частини, що забезпечить легкість перевикористання, заміни або розширення кожної окремої складової не зважаючи на інші частини, що має позитивно вплинути на життєвий цикл програмного забезпечення.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

Порівняння тексту або документів — процес у якому беруть участь, принаймні, два або більше вхідних одиниці.

Вхідні одиниці, у межах даної кваліфікаційної роботи це текст, який може бути представлений як набір символів, або як документ формату «txt».

Під порівнянням розуміється виконання двох завдань:

– порівняння вхідних одиниць;

– надання наглядового результату порівняння, бо без представлення результату в одному порівнянні немає сенсу.

Результат порівняння — вихідна одиниця після процесу порівняння, представляється також у вигляді тексту.

Кожне з наведених завдань складається з менших етапів та низки проблем які виникають с процесі поглиблення у тему, наприклад:

1. Кількість варіацій комбінування символів їх групування та представлення.

Складність полягає у тому, що через це процес написання одного цільного алгоритму який має охоплювати комбінації символів та при цьому спрацьовувати стабільно, швидко та якісно.

2. Швидкість та якість процесу порівняння.

Складність у вірному балансуванні та поєднанні цих понять впливають на користувальницький досвід. Від часу порівняння залежить час очікування, чим більший час очікування, тим менша ймовірність що користувач дочекається результату. Якщо жертвувати якістю для швидшого отримання результату – то є ймовірність що користувача може не задовольнити отриманий результат через гіршу точність та наочність.

3. Представлення результату порівняння.

Складність полягає у формуванні наочного візуального представлення про відмінності які виникають в результаті порівняння між двома вхідними одиницями та надання користувачу додаткового функціонала для взаємодії з результатом для формування ліпшого користувальницького досвіду.

Обрання оптимального підходу та вирішення даних проблем визначають успішність та конкурентоспроможність даного продукту на ринку.

1.2. Призначення розробки та галузь застосування

Розробка веб-додатка порівняння двох документів формату «txt» засобами React.js та веб-API ASP.NET.

Призначене для спрощення та пришвидшення пошуку відмінностей між двома варіантами тексту та надання зрозумілої та наглядної інформації стосовно того де саме знаходиться відмінності.

Дана розробка не належить до конкретної галузі, бо може буди використана усюди де є взаємодія з текстом і необхідність побачити відмінності у двох варіантах.

Тому може бути, як частиною та доповненням для великих розробок, наприклад git (технологія широко використовується в IT галузі для контролю версій розробки), так й самостійною системою для вирішення малих завдань.

1.3. Підстави для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу.

Підставами для розробки (виконання кваліфікаційної роботи) є:

– освітня програма 121 «Інженерія програмного забезпечення»;

– навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровська політехніка» № 268-с від 18.05.2022 р;

– завдання для кваліфікаційної роботи на тему: «Розробка веб-додатка порівняння двох документів формату "txt" засобами React.js та веб-API ASP.NET».

1.4. Постановка завдання

1. Вхідні дані.

1.1. Вхідні дані мають представлятися у наступних форматах:

– текст;

– документ «txt» формату.

2. Налаштування результату.

2.1. Користувач повинен мати змогу обирати тип відображення результату у наступних форматах:

2.1.1. Результат поєднано в один текст.

Відмінності з обох вхідних текстів зливаються та відображаються в одному спільному результуючому тексті.

2.1.2. Результат розділено на окремі тексти.

В залежності від типу відмінності мають представлятися окремо для кожного вхідного тексту, де мають помічатися відмінності лише притаманні даному вхідному тексту.

2.2. Користувач повинен мати змогу налаштовувати колір відмінностей — обирати з запропонованої палітри кольорів окремо для кожної вхідного тексту, якими мають помічатися відмінності у результуючому тексті (або текстах).

3. Результат порівняння.

3.1. Користувач повинен мати змогу візуально ідентифікувати та відрізнити зміни у результуючому тексті. Передбачається що відмінності будуть виділятися від спільного тексту.

3.2. Незалежності від обраного типу представлення результату додатково передбачається виведення всіх відмінностей в окремому списку, з яким користувач має змогу взаємодіяти та при обранні конкретного

елементу зі списку — ця відмінність також має помічатися у результуючому тексті.

4. Авторизація.

Авторизація має здійснюватися за бажанням користувача.

4.1. Автентифікація — входження в обліковий запис.

Має передбачатися введення та обробка наступних вхідних даних для ідентифікації користувача:

– «email» — логін (елемент вхідних даних що ідентифікує користувача);

– пароль.

4.2. Реєстрація — створення нового облікового запису.

Має передбачатися захист від штучних облікових записів — необхідно додатково реалізувати механізм відправлення коду підтвердження та подальшої обробки.

Для процесу реєстрації має передбачатися обробка наступних вхідних даних:

– «email» — пошта на яку має відправлятися код підтвердження;

– пароль;

– код підтвердження.

4.3. Заміна пароля.

Має передбачатися відновлення пароля, для ситуацій коли користувач забув пароль від облікового запису.

Для ідентифікації користувач, щонайменше, має пам’ятати пошту, яка була прив’язана до облікового запису.

Для процесу зміни пароля має передбачатися обробка наступних вхідних даних:

– «email» (логін) — пошта на яку має відправлятися код підтвердження;

– код підтвердження;

– новий пароль.

4.4. Переваги авторизованого користувача:

a) автоматичне збереження інформація про порівняння;

b) відображення інформації про минулі порівняння у вигляді списку;

c) взаємодія зі збереженими результатами порівняння:

– збереження обраного зі списку результату та збереження у форматі «txt» документа на пристрій користувача;

– обрання та відображення обраного результату у форматі поєднаного тексту або розділеного в залежності від типу відмінностей, виведення всіх відмінностей в окремому списку та можливість взаємодіяти з елементами списку, налаштування зовнішнього вигляду відмінностей – тобто можливість взаємодіяти зі збереженим результатом як до збереження.

5. Зберігання даних.

Дані про користувачів та результати мають зберігатися у базі даних наступним чином:

a) опис користувача:

– Id — ідентифікатор користувача;

– Type — тип (роль);

– Email — пошта (логін);

– Password — пароль.

b) опис результату порівняння:

– ResultId — ідентифікатор результату;

– UserId — ідентифікатор користувача якому належить даний результат;

– CreateDate — дата створення;

– ResultFile — шлях до збереженого документа формату «txt»;

– ResultList — шлях до збереженого документа з інструкцією для результату.

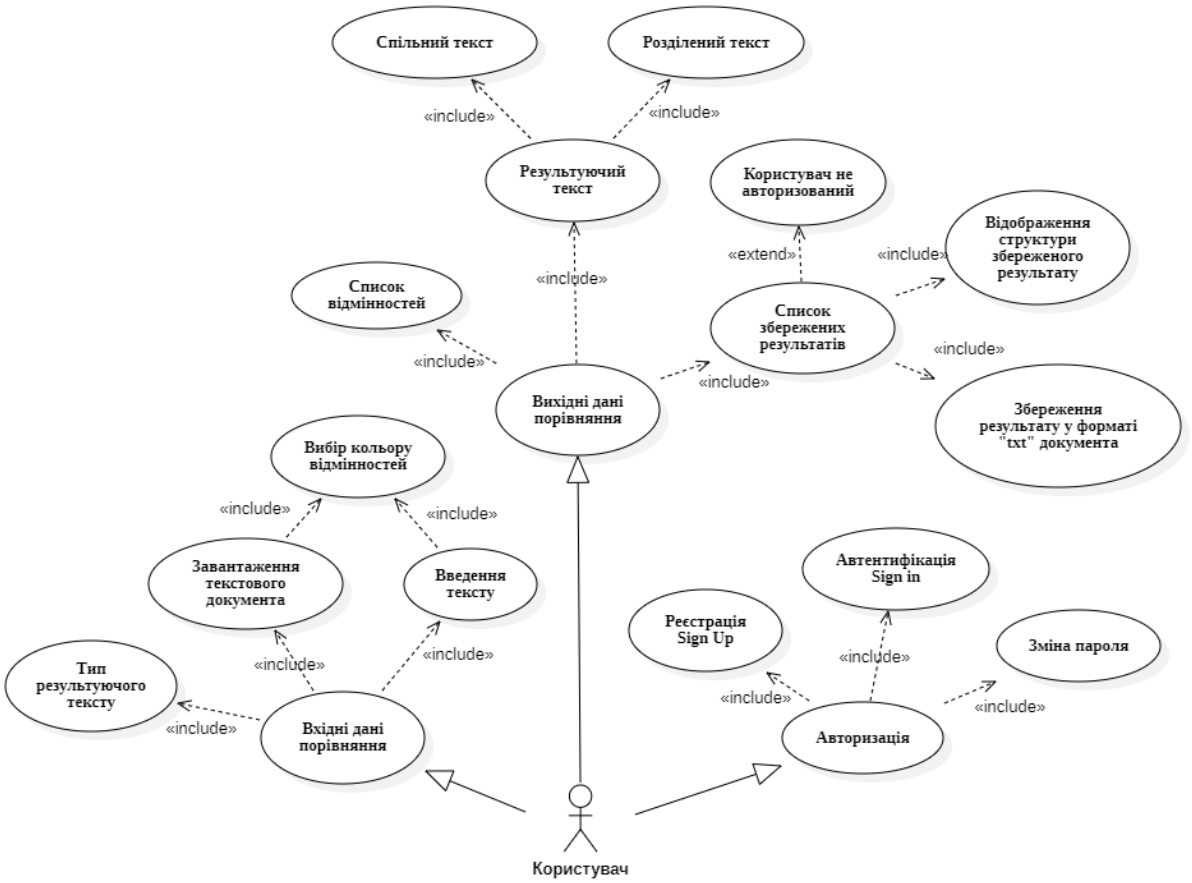

Рис. 1.1. Схема використання ПЗ

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Вхідних дані:

1. Порівняння:

1.1. Вхідні одиниці для порівняння, можуть бути представлені у наступних форматах:

a) текст — передбачається що користувач може, або ввести вручну, або скопіювати та ставити до спеціально відведених елементів;

b) текстовий документ формату «txt» — передбачається що користувач може перетягти курсором документ до меж спеціально відведених елементів, таким чином виконується процес завантаження документів.

1.2. Вибір кольору відображення відмінностей (для кожної вхідної одиниці тексту окремо).

2. Авторизація:

– «email» (логін);

– пароль;

– код підтвердження.

Вихідні дані:

– результат порівняння у форматі тексту (спільний або розділений);

– список відмінностей;

– список збережених результатів.

1.5.2. Вимоги до інформаційної безпеки

В процесі авторизації передбачається захист від штучних реєстрацій облікових записів — механізм підтвердження особистості. Механізм захисту полягає у введені користувачем коду підтвердження, який відправляється на попередньо введену пошту.

Для захищеної передачі даних про зареєстрованого користувача використовується механізм використання токену (зашифрована послідовність символів з даними про обліковий запис), з яким взаємодіє система та контролює доступ, таким чином навіть перехопивши та отримавши такий рядок зловмисник не зможе них скористатися, таким чином захищаються передача особистих даних.

1.5.3. Вимоги до складу та параметрів технічних засобів

Мінімальні технічні засоби серверної частини:

– операційна система — Windows 7;

– процесор — з тактовою частотою не нижче 1,8 ГГц;

– оперативна пам’ять — 2 ГБ;

– місце на жорсткому диску — 20 ГБ вільного місця з урахуванням необхідного запасу для майбутнього заповнення бази даних.

Мінімальні технічні засоби користувальницької частини:

a) операційні системи:

– Windows 7;

– MacOS 10.12 (macOS Sierra);

– ОС Linux — будь-яка, яка підтримує наступні браузери;

– iOS 11.4;

– Android 7.0.

b) браузери:

– Chrome 64;

– Firefox 78;

– Safari 13;

– Microsoft Edge 79.

c) процесор — з тактовою частотою не нижче 1,5 ГГц.

d) оперативна пам’ять — 2 ГБ.

1.5.4. Вимоги до інформаційної та програмної сумісності

1. Рекомендується використовувати Visual Studio 2019 як інтегроване середовище розробки для частини back-end з наступними модулями:

– розробка класичних застосунків .NET;

– ASP.NET та розробка веб-додатків;

– сховище та обробка даних.

2. Застосунок node.js — необхідно для розгортання користувальницької частини.

3. SQL Server Express — системи управління реляційними базами даних.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Передбачається що в програмі мають вестися дані про користувачів та дані про результати порівняння.

Ведення даної інформації може бути корисним як:

- моніторинг кількості користувачів;
- моніторинг активності використання порівняння;
- моніторинг активності кожного користувача.

2.2. Опис застосованих математичних методів

Математичні методи не застосовуються.

2.3. Опис використаної архітектури та шаблонів проєктування

Реалізація поділена на дві частини:

- клієнтська частина (front-end) — графічний інтерфейс для взаємодії користувача з системою, введенням початкових даних та виведенням результату;
- серверна частина (back-end) — основна обробка даних, взаємодія з базою даних та іншими сервісами.

Для взаємодії між частинами системи використовується технологія — API (інтерфейс прикладного програмування.).

Для взаємодії та передачі даних необхідно передбачити в кожній частині методи відправлення запитів та приймання відповідей:

1. front-end частина:

Ініціює зв’язок з іншою частиною та має включати:

- методи відправлення запитів до іншої частини;
- методи прийняття відповідей — результати запитів.

2. back-end частина.

Очікує запити на обробку за спеціально визначеними адресами — endpoint (заздалегідь визначені адреси для взаємодії з серверною частиною, що використовується у технології API), обробляється запит, відправляється відповідь.

Відповідь може складатися з необхідних даних та поділяється за типом на[1-2, 5]:

– 1xx – інформація про процес передачі;
– 2xx – інформація про вдале приймання та оброблення запиту клієнта;
– 3xx – переадресація;
– 4xx – інформація про помилку з боку клієнта;
– 5xx – інформація про помилку з боку сервера.

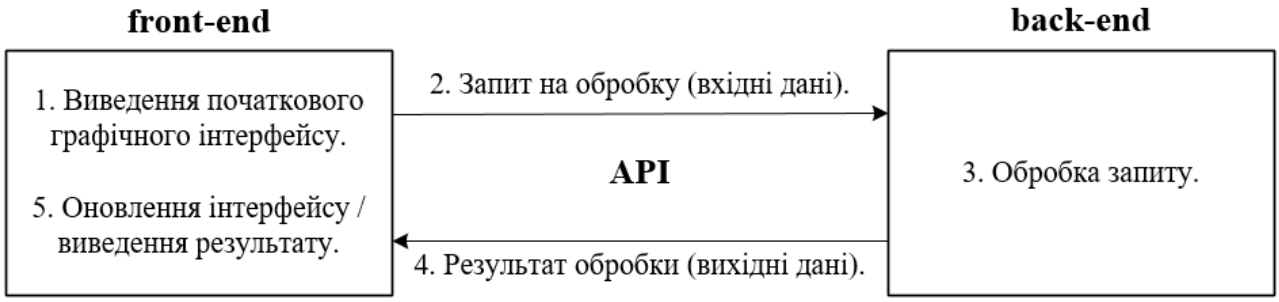

Рис. 2.1. Структура взаємодії частин програми

2.4. Опис використаних технологій та мов програмування

1. Клієнтська частина:

– JavaScript (JS) — динамічна, об’єктно-орієнтована прототипна мова програмування, широко використовується для створення сценаріїв веб-сторінок, що надає можливість на боці клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки[2-3];

– Node.js — середовище для побудови масштабованих мережевих додатків, поєднання та взаємодії інших бібліотек та технологій[4];

– React — JavaScript-бібліотека[5-6] для створення інтерфейсів користувача, поєднує базові інструменти для написання web-додатків, тим самим оптимізує процес розробки[7];

– Redux — JavaScript-бібліотека для оптимізації обробки сховища даних[8-9] в результаті взаємодії користувача з сайтом, що у свою чергу дозволяє динамічно оновлювати графічний інтерфейс до змін[10];

– MaterialUI — JavaScript-бібліотека, включає комплексний набір інструментів інтерфейсу користувача, що дозволяє легше та швидше описувати логіку та зовнішній вигляд компонентів [11];

– Sass — модернізована мова опису стилів CSS, для налаштування зовнішнього вигляду елементів графічного інтерфейсу[12].

Основною технологією для розробки клієнтської частини було обрано React — тому що дана технологія оптимізує та полегшую використовуваний трафік інтернету для відображення графічного інтерфейсу за рахунок того, що передається, в основному, не вже готові сторінки, а більшу частину модернізованого javaScript коду, який обробляється та формує результуючу сторінку.

Також дана технологія розвивається та являється, на цей час, однією з найпопулярніших, що свідчить про гарну взаємодію та поєднання зі сторонніми технологіями, що в результаті може значно полегшити розробку, пропонувати ліпші розв’язки існуючих проблем та підвищити швидкість розробки.

2. Серверна частина:

– C# (See Sharp) — об’єктно-орієнтована та типізовано-безпечна мова програмування, що дозволяє створювати безліч типів безпечних та надійних програм, що працюють у середовищі .NET[14-15];

– ASP.NET — відкрите програмне забезпечення для розробки як серверної частини, так і повноцінних web-додатків в цілому[16-17];

– WEB API — шаблон у середовищі ASP.NET, архітектура та технології дозволяє взаємодіяти з технологією передачі даних — API, отримувати запити, обробляти та формувати відповіді[18-19];

– Microsoft SQL Server Express – безкоштовна система керування даними, що забезпечує функціональне та надійне сховище даних для веб-сайтів [20].

Дані технології були обрані, у більшій частині, завдяки мові програмування C#, за впевненість у можливості реалізації задуманої ідеї порівняння текстів. На основі мови було обрано технологію WEB API саме за її гнучкість, тобто написання на цій технології дає змогу використання для будь-якого іншого графічного інтерфейсу, написаного для настільних, мобільних або web-додатків, тобто дозволяє чітко розмежувати реалізацію кваліфікаційної роботи.

3. Технологія передачі даних.

API (Application Programming Interface) — інтерфейс прикладного програмування. Це готовий код, що складається з набору функцій, бібліотек, методів і процедур, що дозволяє різним додаткам взаємодіяти один з одним[21].

2.5. Опис структури програми та алгоритмів її функціонування

Як було описано раніше, кваліфікаційну роботу поділено на дві взаємопов’язані частини та відповідно реалізація яких також відрізняються:

1. Опис структури front-end частини (рис. 2.2).

Дана частина складається з взаємопов’язаних компонентів, завдання яких описати представлення та логіку графічного інтерфейсу для користувача.

Важлива роль полягає в організації даних всередині між компонентами.

Є наступні шляхи організації даних:

– локальне сховище — реалізовується всередині компоненту та може буди передано дочірнім компонентам, але складність полягає в передачі даних до батьківських компонентах;

– загальне сховище — доступне для усіх компонентів і може бути викликано у будь-якому місці, але організувати та контролювати подібне сховище доволі складно.

Реалізовуються обидва типи сховищ, кожне з яких має особливу роль:

– локальне сховище — потрапляють дані необхідні для роботи компонента та для його дочірніх компонентів;

– загальне сховище — потрапляють дані з отриманої відповіді від частини back-end.

В архітектурі передбачається що кожен компонент має включати наступні файли:

– файл зі структурою компонента та розподілення даних;

– файл налаштування зовнішнього вигляду;

– файл обробки особливої специфічної логіки компонента (наприклад обробка функціонала форми).

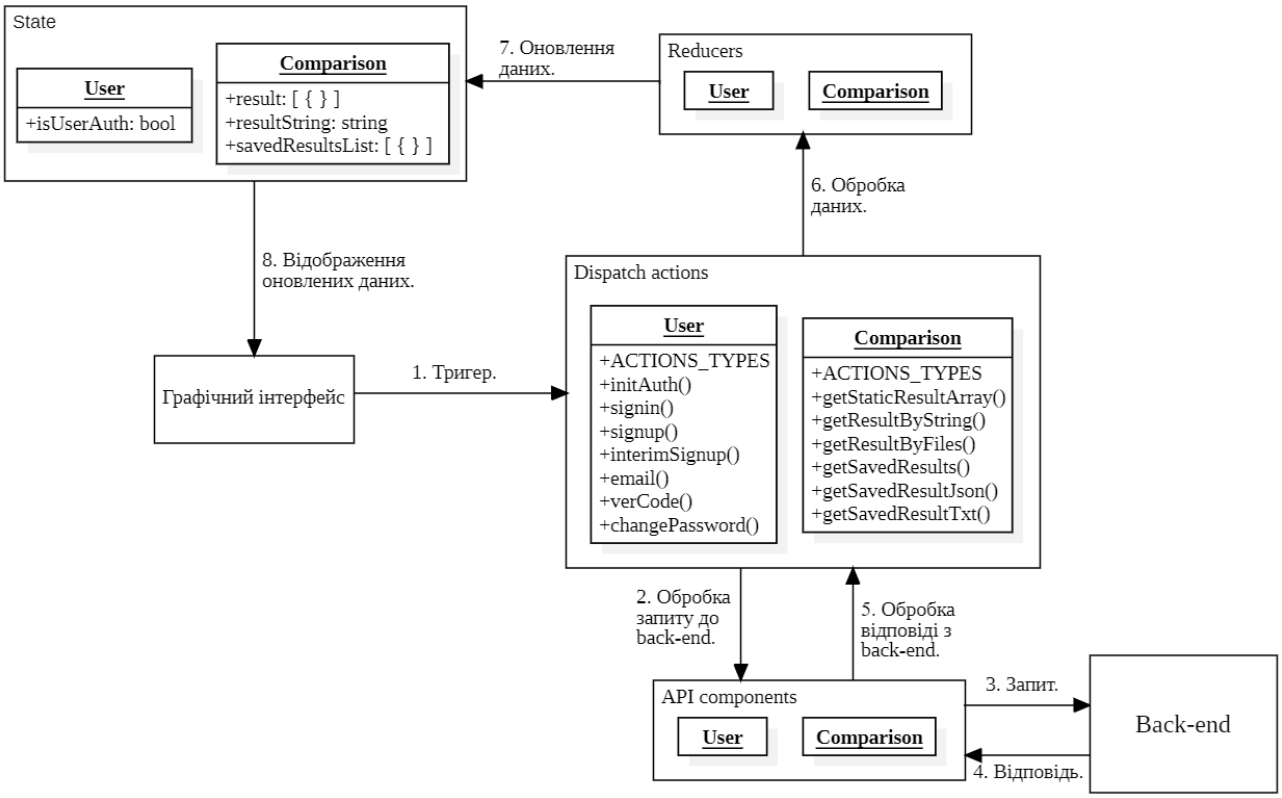

Рис. 2.2. Схеми взаємодії компонентів **(**front-end частина)

Також дана частина включає процеси авторизації, за допомогою яких виконується обробка ведення даних про користувачів та результати порівняння.

Авторизація включає в себе наступні процеси:

– реєстрація — створення нового облікового запису;

– автентифікація — входження в обліковий запис;

– зміна пароля.

Процес реєстрації поділяється на наступні етапи (рис. 2.3):

1 Введення користувачем даних для облікового запису.

2 Відправлення на серверну частину даних для перевірки на відсутність з подібних записів.

3 Відправлення з серверної частини на вказану пошту коду перевірки особистості.

4 Перевірка користувачем пошти на наявність листа з верифікаційним кодом.

5 Введення користувачем коду до спеціально призначеного поля.

6 Відправлення на серверну частину даних для перевірки на коректність верифікаційного коду.

7 Завершення реєстрації — ідентифікація користувача.

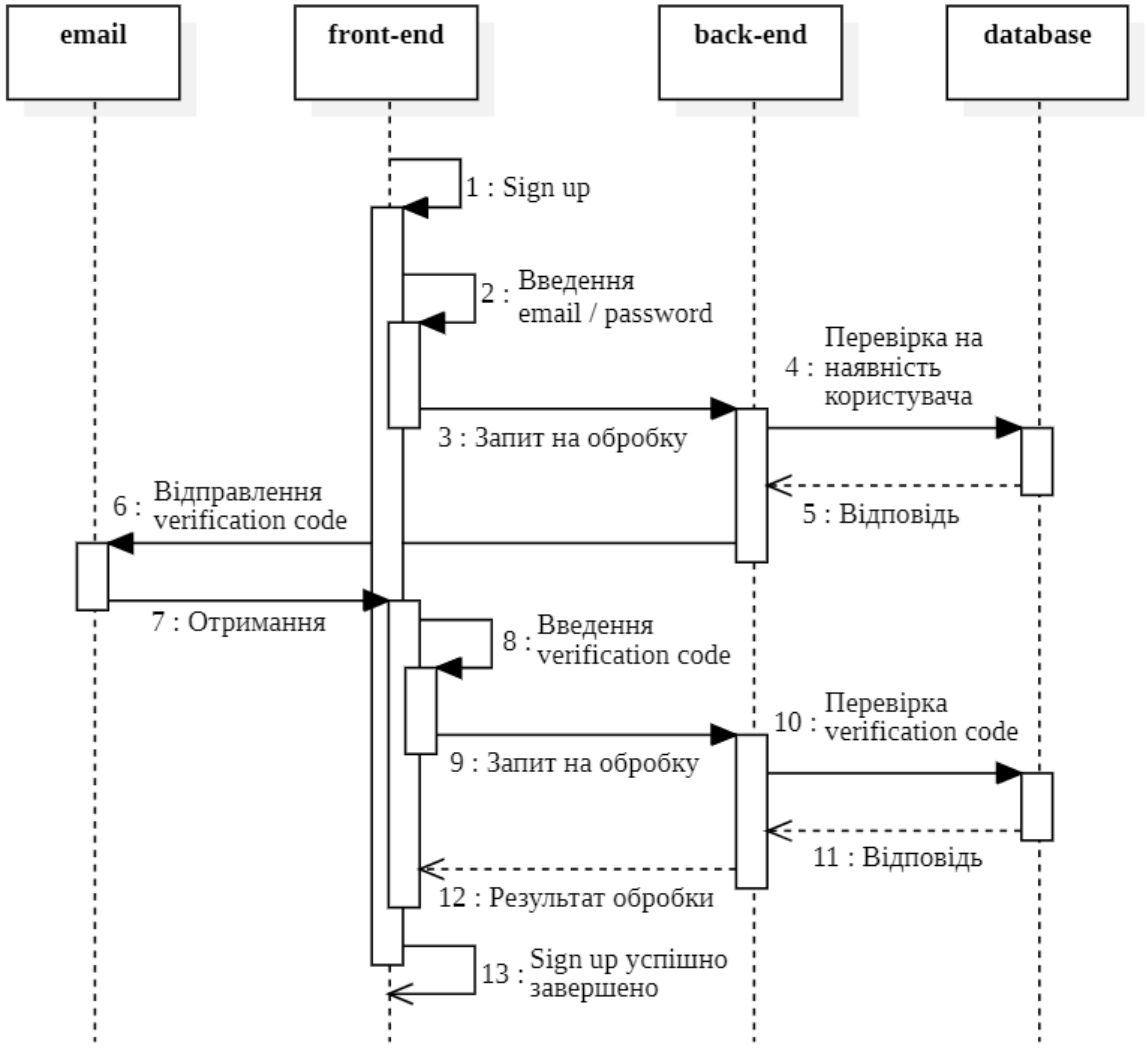

Рис. 2.3. Схема послідовності процесу реєстрації

Процес зміни пароля поділяється на декілька схожих за структурою етапів (рис. 2.4):

1 Введення користувачем поетапно даних (пошта-логін, верифікаційний код відправлений на пошту, новий пароль).

2 Після кожного етапу введення даних виконується запит на серверну частину для перевірки на відповідність.

Після останнього етапу, користувач автоматично ідентифікується, без необхідності проходження процесу автентифікації.

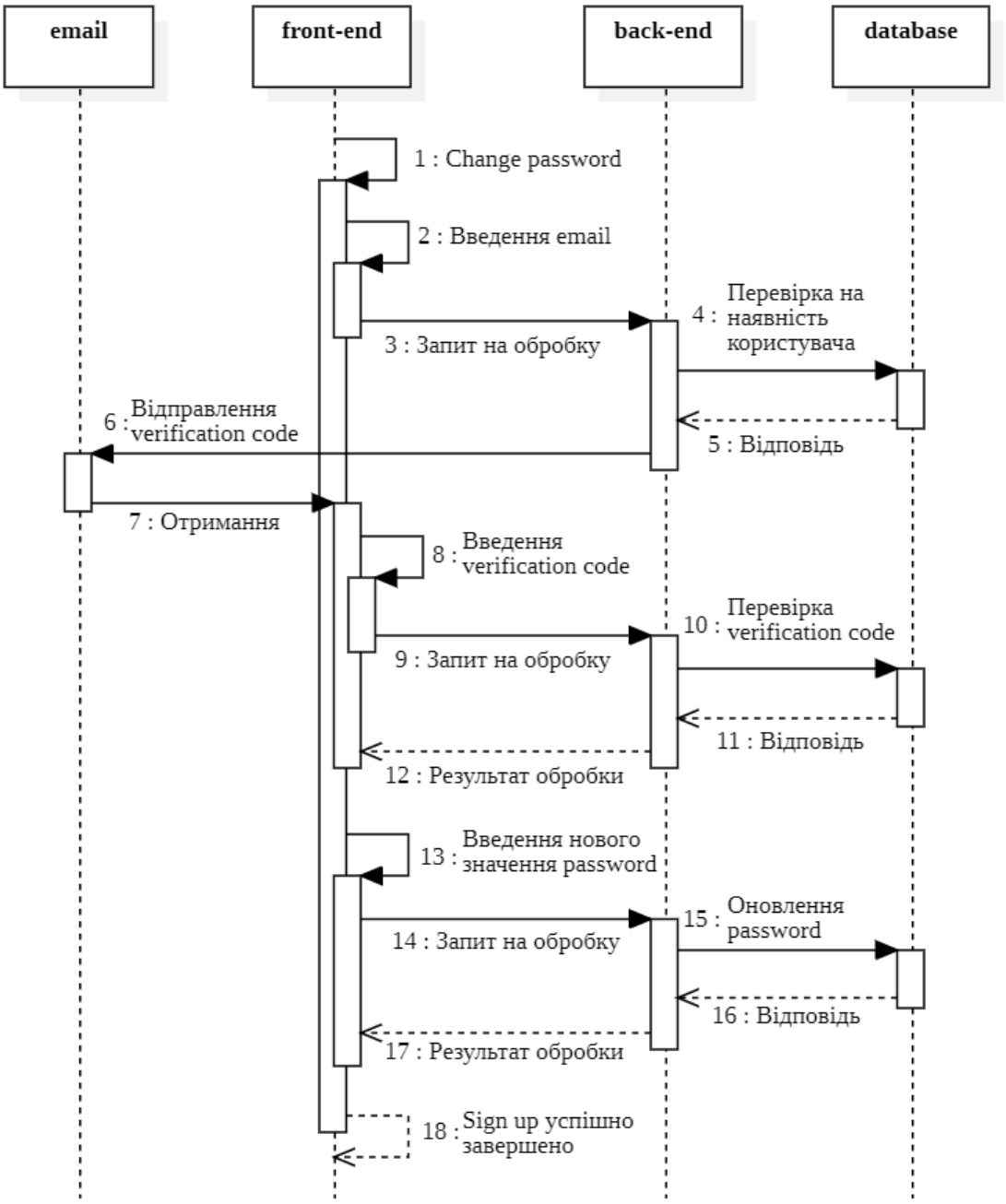

Рис. 2.4. Схема послідовності процесу зміни пароля

Процес автентифікації поділяється на наступні етапи (рис. 2.5):

1 Введення користувачем даних про облікового запису.

2 Відправлення на серверну частину даних для перевірки на відповідність.

3 Завершення автентифікації — ідентифікація користувача.

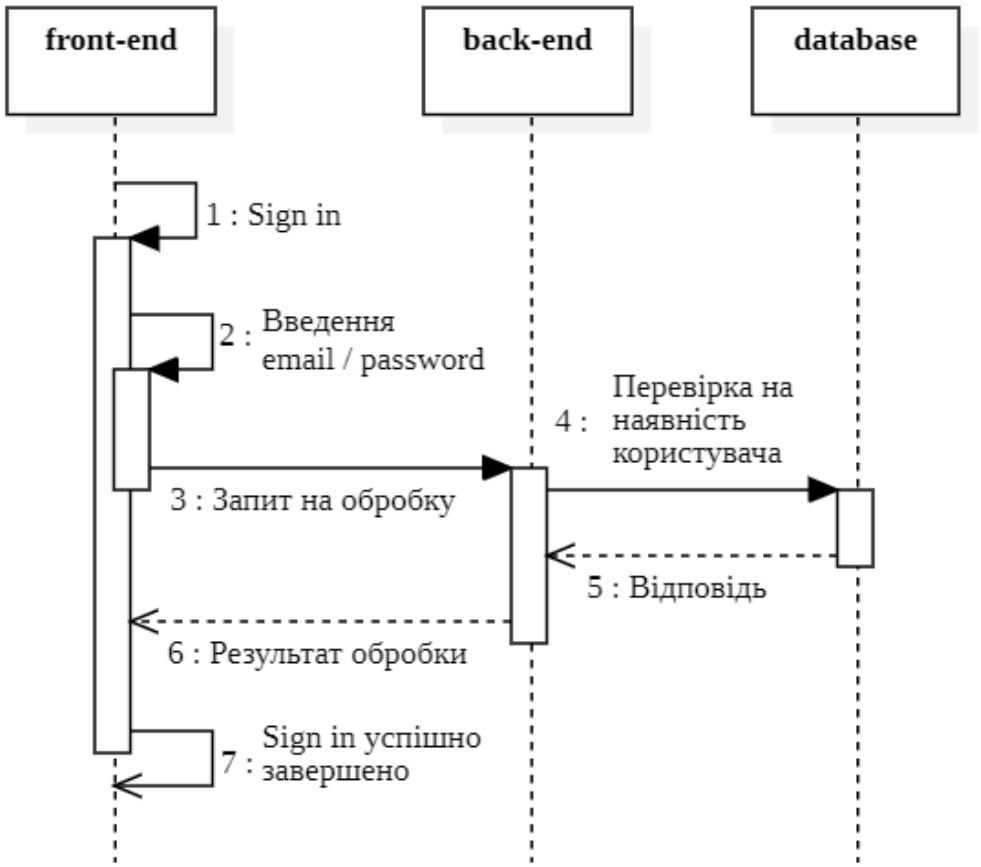

Рис. 2.5. Схема послідовності процесу автентифікації

2. Опис структури back-end частини (рис. 2.8).

Основний алгоритм який присутній в даній частині — відстань Левенштейна[24].

Для розрахунку відстані Левенштейна використовується матриця розміром (n + 1) * (m + 1), n і m — довжини порівнюваних рядків (кожен рядок та стовпець відповідає символу з відповідної послідовності). Окрім цього вартість операцій вилучення, заміни та вставки вважається однаковою.

Тобто послідовність алгоритму можна описати наступними діями:

1 Побудова матриці суміжності (n + 1) * (m + 1), n і m — довжини порівнюваних рядків.

2 Заповнення матриці у відповідності за формулою (рис. 2.6).

3 Побудова найкоротшого шляху, відповідно до коефіцієнтів у комірках. Шлях будується з правого-нижнього краю матриці до лівого-верхнього.

$$D_{0,0} = 0$$

$$D_{i,j} = min \begin{cases} D_{i-1,j-1} & +0 \text{ (equal, nochange)} \\ D_{i-1,j-1} & +1 \text{ (replace)} \\ D_{i-1,j} & +1 \text{ (insert)} \\ D_{i,j-1} & +1 \text{ (delete)} \end{cases}$$

Рис. 2.6. Формула розрахунку коефіцієнту схожості

```
int LevenshteinDistance(char str1[1..lenStr1], char str2[1..lenStr2])
   // d таблиця кількість рядків = lenStr1+1 та кількість стовпців = lenStr2+1
   declare int d[0..lenStr1, 0..lenStr2]

   // i та j використовуються для індексування позиції у str1 та у str2
   declare int i, j, cost

   for i from 0 to lenStr1
       d[i, 0] := i
   for j from 0 to lenStr2
       d[0, j] := j

   for i from 1 to lenStr1
       for j from 1 to lenStr2
           if str1[i] = str2[j] then cost := 0       //однакові
                                else cost := 1       //заміна
           d[i, j] := minimum(
                                d[i-1, j  ] + 1,     // вилучення
                                d[i  , j-1] + 1,     // вставка
                                d[i-1, j-1] + cost   // заміна або однакові
                            )

   return d[lenStr1, lenStr2] //значення відстані Левенштайна в останній клітинці матриці
```

Рис. 2.7. Псевдокод алгоритму

Тобто даний алгоритм займається перевіркою кожного символу з однієї послідовності з усіма іншими символами з другої послідовності та в результаті отримується послідовність порівняння.

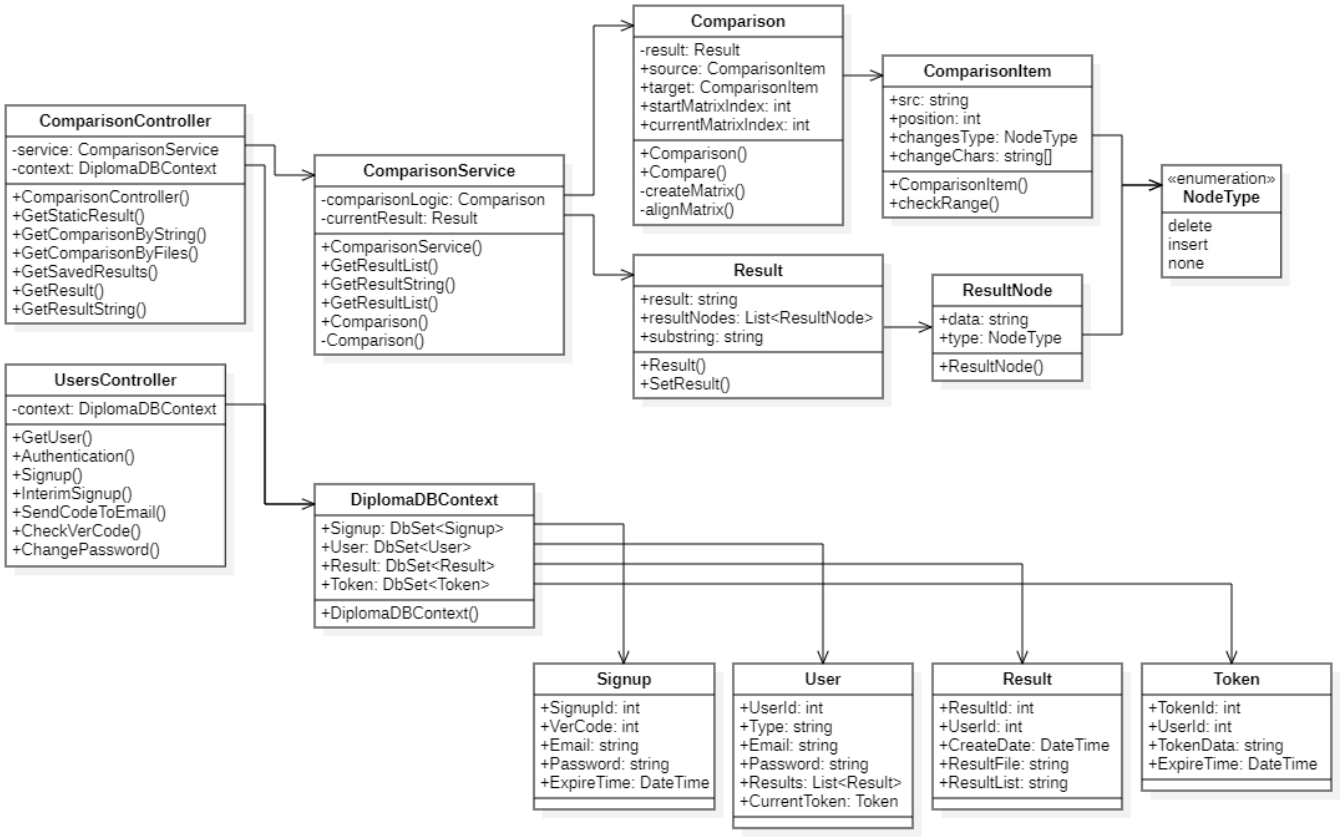

Рис. 2.8. Схеми взаємодії класів **(**back-end частина)

3. Опис структури бази даних.

Таблиця «User» — призначена для ведення інформації про зареєстрованих користувачів, (табл. 2.1).

Таблиця 2.1.

Опис таблиці «User»

Назва	Тип	Призначення
UserId	INTEGER	Ідентифікатор користувача
Type	VARCHAR	Роль
Email	VARCHAR	Пошта (логін)
Password	VARCHAR	Пароль

Таблиця «Token» — призначена для ведення інформації для захищеної передачі даних про зареєстрованого користувача. Пов’язана з таблицею «User» типом — один до одного.

Передбачається що окремий користувач може мати один запис у даній таблиці, який має постійно оновлюватися (табл. 2.2).

Таблиця 2.2.

Опис таблиці «Token»

Назва	Тип	Призначення
TokenId	INTEGER	Ідентифікатор таблиці
UserId	INTEGER	Ідентифікатор користувача
TokenData	VARCHAR	Токен
ExpireTime	DATETIME	Дата та час терміну дії

Таблиця «Result» — призначена для ведення інформації про результати порівняння користувачів. Пов’язана з таблицею «User» типом – один до багатьох.

Передбачається що кожен зареєстрований користувач може мати декілька збережених результатів про порівняння (табл. 2.3).

Таблиця 2.3.

Опис таблиці «Result»

Назва	Тип	Призначення
ResultId	INTEGER	Ідентифікатор результату
UserId	INTEGER	Ідентифікатор користувача
CreateDate	DATETIME	Дата та час створення
ResultFile	VARCHAR	Шлях до збереженого документа формату «txt»
ResultList	VARCHAR	Шлях до збереженого документа з інструкцією

Таблиця «Signup» — призначена для ведення інформації про процес реєстрації користувачів. Пов’язана з таблицею «User» типом – один до одного.

Передбачається що дані про реєстрацію користувачів зберігаються поки користувач не підтвердить особистість, тобто поки не пройде успішно процес реєстрації (табл. 2.4).

Таблиця 2.4.

Опис таблиці «Signup»

Назва	Тип	Призначення
SignupId	INTEGER	Ідентифікатор реєстрації
UserId	INTEGER	Ідентифікатор користувача
VerCode	INTEGER	Код підтвердження особистості
ExpireTime	DATETIME	Дата та час терміну дії коду підтвердження

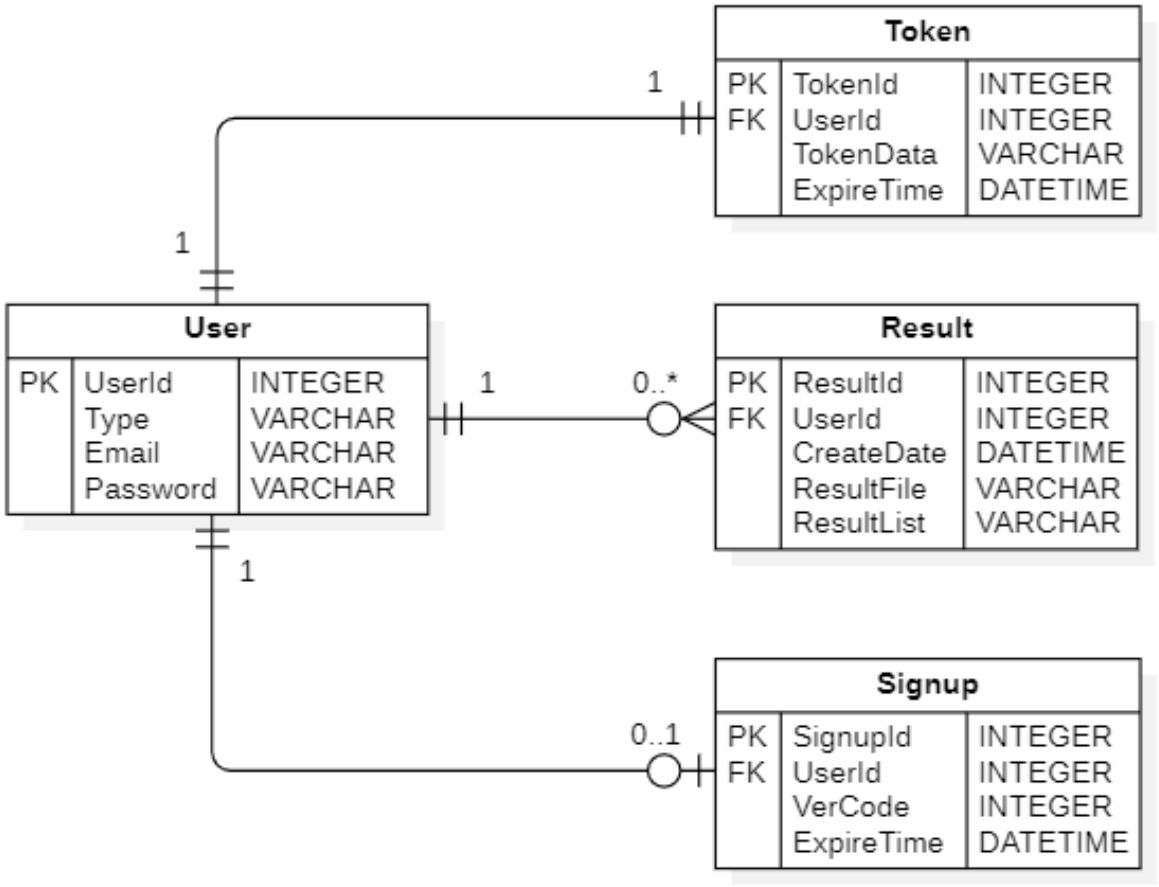

Рис. 2.9. Загальна структура бази даних

4. Опис відправлення коду підтвердження .

В процесі реєстрація для підтвердження особистості користувача використовується взаємодія з сервісом «smtp.elasticemail», за допомогою якого відбувається відправлення повідомлення з кодом підтвердження.

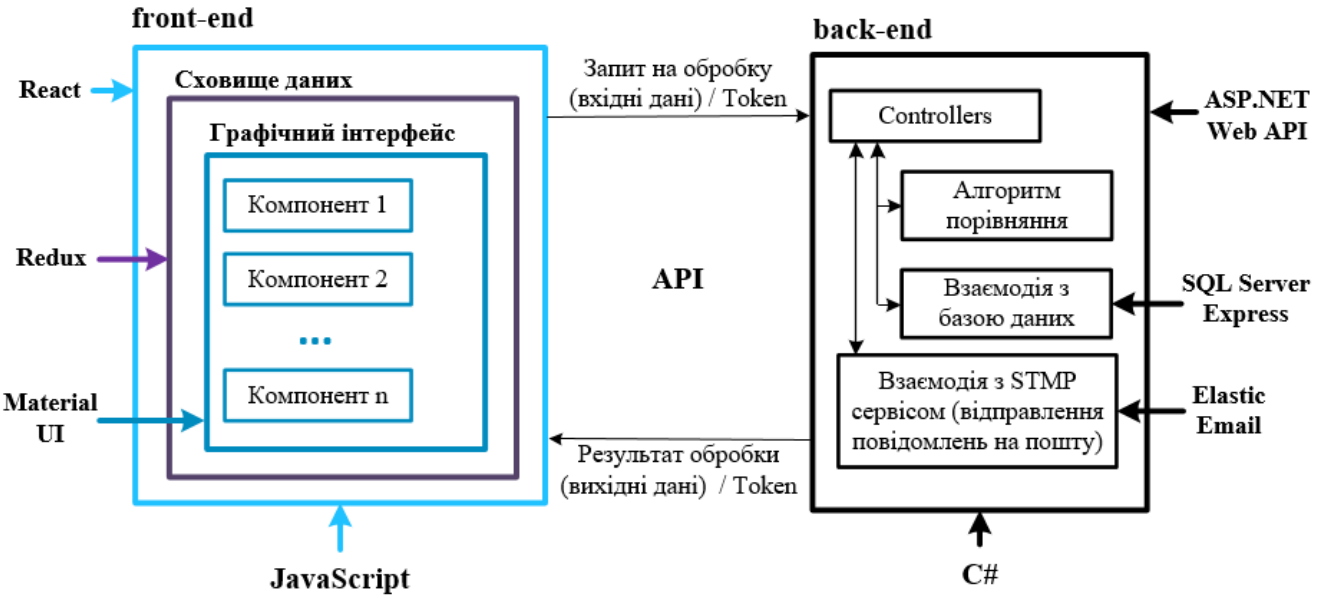

Рис. 2.10. Структура взаємодії частин програми

2.6. Обґрунтування та організація вхідних та вихідних даних програми

2.6.1. Організація вхідних даних

Основною частиною вхідних даних є введення значень для порівняння. Процес фіксації вхідних значень для порівняння відбувається введенням тексту до спеціально відведених полів. Передбачається що вхідних значень для порівняння буде у кількості двох одиниць.

Також вхідні одиниці для порівняння можуть бути представлені у вигляді документів формату «txt». Для фіксації вхідних даних відбувається перетягуванням курсором документів до меж спеціально відведених полів. У подальшому передані файли оброблюються та формуються у текст для порівняння.

2.6.2. Організація вихідних даних

До вихідних даних належить результат порівняння — результуючий текст. Передбачається що відмінності будуть виділятися від спільного тексту, для ліпшої ідентифікації різниці та наглядності.

До вихідних даних належить результат порівняння — результуючий текст. Передбачається що відмінності будуть візуально виділятися від спільного тексту, для ліпшої ідентифікувати різниці та наглядності.

Перед представленням результату виконується наступна обробка:

1. Результат у вигляді послідовних «json» об’єктів (табл. 2.5, ліст. 2.1), де кожний об’єкт зберігає інформацію про частину результату надходить до обробника.

2. Послідовно оброблюється кожен об’єкт та в залежності від структури кожному об’єкту надається зовнішній вигляд.

3. Результат виводиться на спеціально призначене поле в структурі сторінки.

Таблиця 2.5.

Опис результату у вигляді структури «json» об’єктів

Поле	Значення	Призначення
parentInputValue	source / target / both	Показник належності даної одиниці об’єкту до конкретних вхідних даних, де: – source — належність лише до першого вхідного значення для порівняння; – target — належність лише до другого вхідного значення для порівняння; – both — належність до обох вхідних значень.
isChanged	true / false	Показник, що дана одиниця об’єкту являється відмінністю між двома вхідними даними, де: – true — являється відмінністю; – false — не являється відмінністю.
data	будь-яку кількість символів	Зміст одиниці об’єкту.

Лістинг 2.1

Структура результату у вигляді «json» об’єктів

```
[
   {
      "data": "Lorem ",
      "parentInputValue": "both",
      "isChanged": false
   },
   {
      "data": "  ",
      "parentInputValue": "source",
      "isChanged": true
   },
   {
      "data": "ipsum dolor sit amet, consectetur adipiscing elit, ",
      "parentInputValue": "both",
      "isChanged": false
   },
   {
      "data": "sed ",
      "parentInputValue": "target",
      "isChanged": true
   },
   {
      "data": "do eiusmod tempor incididunt ut labore et dolore magna aliqua",
      "parentInputValue": "both",
      "isChanged": false
   },
   {
"data": ". ",
      "parentInputValue": "target",
      "isChanged": true
   },
]
```

Результат представляються як вихідні дані у вигляді «txt» файлу, який користувач може завантажити собі на пристрій.

Для представлення вихідних даних у вигляді документа виконується наступна обробка:

1. На стороні клієнта створюється тимчасовий документ куди записується результат у вигляді тексту (відмінності помічаються спеціально відведеними символами).

2. Документ вмонтовується в структуру сторінки, та очікує взаємодії з користувачем.

3. Після взаємодії з користувачем документ видаляється зі структури сторінки.

2.7. Опис розробленого програмного продукту

2.7.1. Використані технічні засоби

Опис складу технічних засобів для обчислювання системи складається з однієї обчислювальної машини з наступними технічними засобами:

– операційна система — Windows 7 та новіші версії;

– процесор — з тактовою частотою не нижче 2,4 ГГц;

– оперативна пам’ять — 4 ГБ;

– місце на жорсткому диску — 20 ГБ вільного місця з урахуванням необхідного запасу для майбутнього заповнення бази даних.

2.7.2. Використані програмні засоби

Опис складу програмних засобів для використання клієнтської частини:

a) Node.js — середовище для побудови масштабованих мережевих додатків, поєднання та взаємодії інших бібліотек та технологій;

b) Пакет бібліотек — необхідних для коректної роботи системи:

– react, react-dom, react-router-dom — створення загального інтерфейсу користувача, поєднує базові інструменти для написання web-додатків;

– redux, react-redux, redux-thunk — інструменти для оптимізації обробки сховища даних;

– axios — інструменти для полегшення взаємодії з API;

– mui/material, mui/styles, mui/icons-material — бібліотеки з набором інструментів для створення візуальних елементів в інтерфейсі користувача;

– sass — модернізована мова опису стилів CSS, для налаштування зовнішнього вигляду елементів графічного інтерфейсу;

– classnames — інструменти для поєднання декількох описів стилів для одного елементу графічного інтерфейсу;

– react-dnd, react-dnd-html5-backend — інструменти для описання логіки завантаження файлу методом пересування курсором.

Опис складу програмних засобів для використання серверної частини:

a) C# (See Sharp) — мова програмування;

b) Visual Studio — середовище розробки;

c) ASP.NET — програмне забезпечення для розробки веб-застосунків;

d) WEB API — шаблон у середовищі ASP.NET, архітектура та технології дозволяє взаємодіяти з технологією передачі даних — API;

e) Microsoft SQL Server Express — система керування даними.

2.7.3. Виклик та завантаження програми

Опис способу виклику серверної частини:

1. Встановити на обчислювальну машину середовище розробки — Visual Studio.
2. Встановити на обчислювальну машину систему управління даними — Microsoft SQL Server Express (для зручності використання рекомендується завантажити графічне середовище управління — Microsoft SQL Server Management Studio).
3. З носія даних завантажити back-end частину на обчислювальну машину.
4. За допомогою Visual Studio відкрити застосунок та термінал за шляхом: Tools → NuGet Package Manager → Package Manager Console.
5. У терміналі використати команду – «Update-Database» (дана команда виконує міграцію та ініціалізує базу даних).
6. У середовищі Visual Studio натиснути кнопку «ISS Express» (дана команда виконує запуск серверної частини).

Опис способу виклику клієнтської частини:

1. Встановити на обчислювальну машину середовище для розробки — Node.js.
2. Для зручності рекомендується встановити графічне середовище розробки (наприклад Visual Studio Code) або можна використовувати термінал.
3. З носія даних завантажити front-end частину на обчислювальну машину.
4. Перейти через термінал або графічне середовище розробки до кореневої

папки (у середині має знаходитись файл «package.json»).

5. Використовуючи пакетний менеджер npm (встановлюється автоматично з Node.js) виконати команду у терміналі «npm install» (дана команда має завантажити усі необхідні бібліотеки для коректної роботи застосунку).

6. Виконати команду «npm run start» (дана команда виконує запуск клієнтської частини).

2.7.4. Опис інтерфейсу користувача

Загальний вигляд інтерфейсу користувача

Загальний вигляд інтерфейсу користувача для взаємодії з процесом порівняння може бути представлений у двох виглядах:

– звичайний вигляд (рис. 2.11);

– з додатковими панелями з обох боків екрана (рис. 2.12).

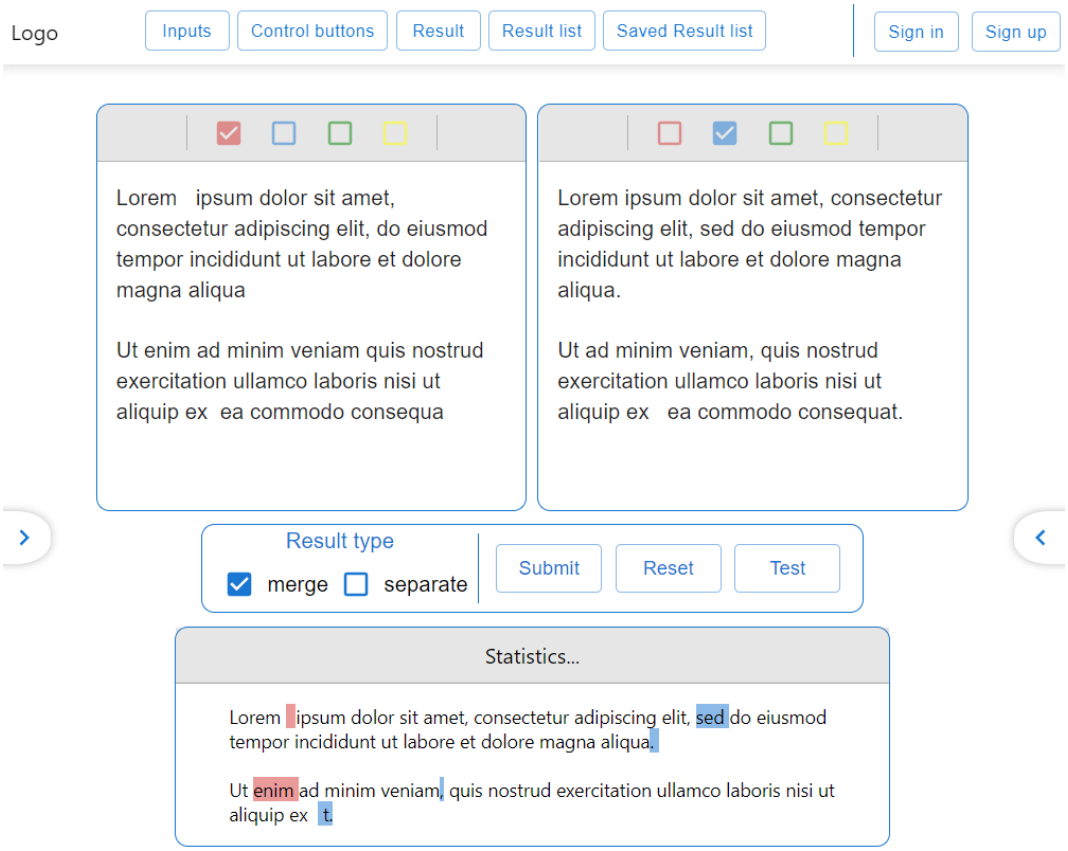

Рис. 2.11. Загальний вигляд інтерфейсу

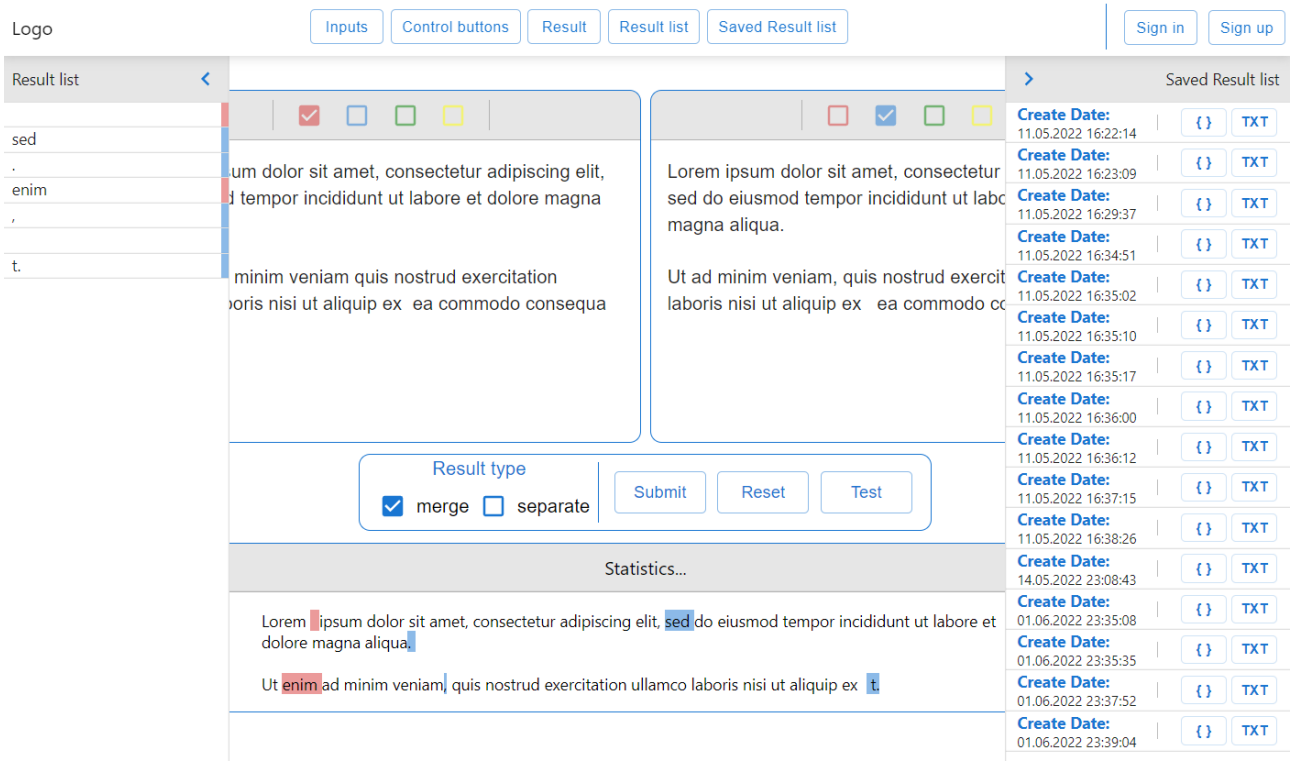

Рис. 2.12. Загальний вигляд інтерфейсу з додатковими панелями

Опис інтерфейсу ведення вхідних даних

Користувальницький графічний інтерфейс передбачає компоненти для введення двох окремих вхідних одиниць для порівняння (рис. 2.13), компоненти мають тип – багаторядковий, що дозволяє вводити текст у декілька рядків (рис. 2.14).

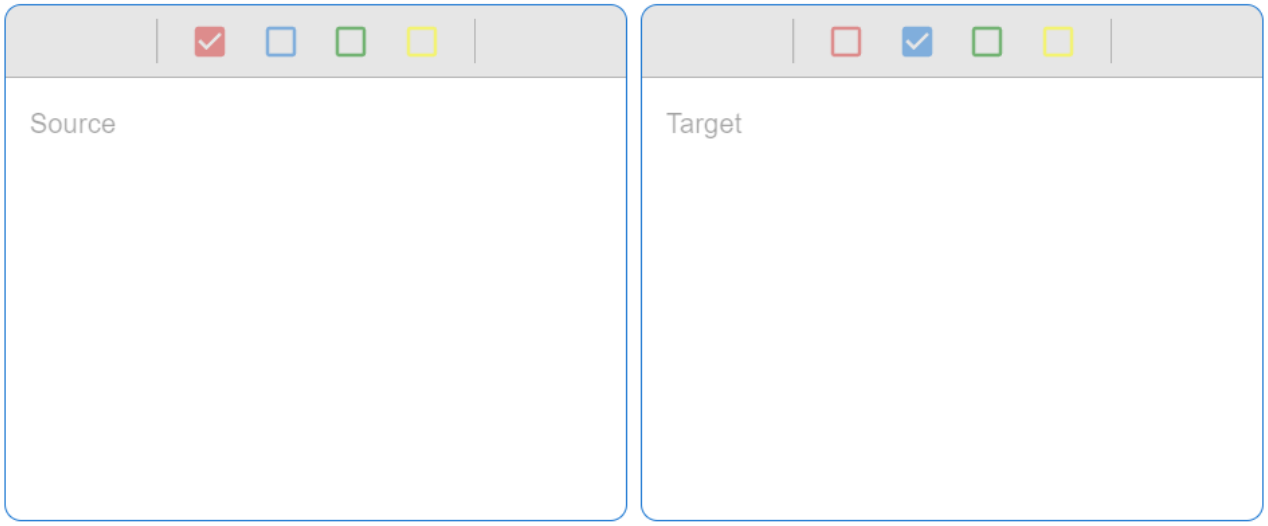

Рис. 2.13. Поля для введення вхідних даних для порівняння

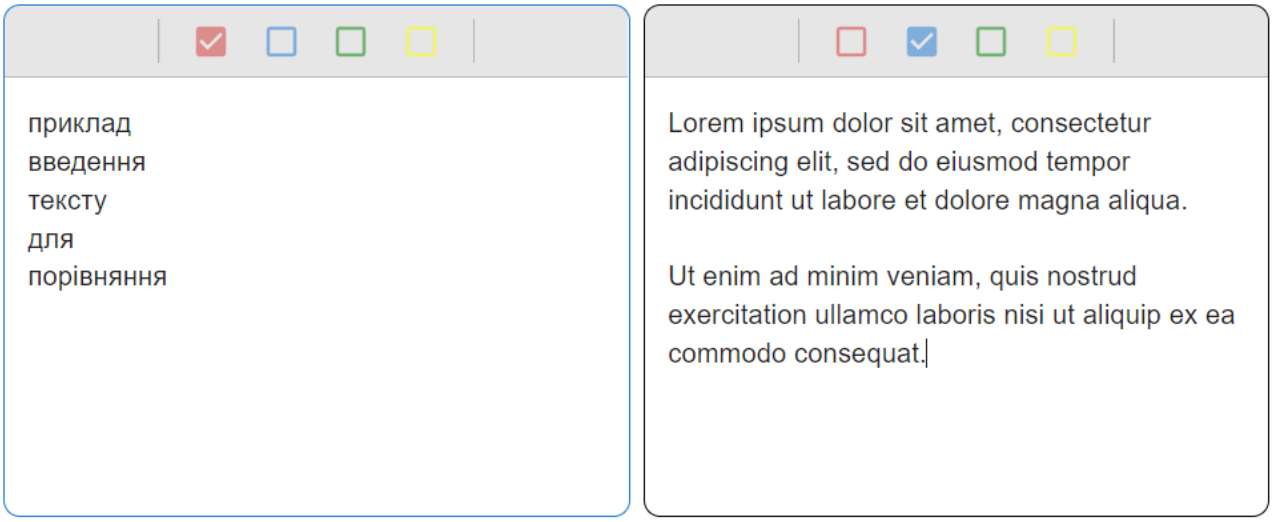

Рис. 2.14. Приклад введення багаторядкового тексту

Окрім можливості вручну вводити вхідні дані для порівняння, передбачено можливість завантаження текстових документів.

Для завантаження необхідно обрати з носія даних документ, виділити його курсором та перетягти до вище зазначених полів.

Для ліпшого інформування користувача передбачено декілька станів для полів введення вхідних даних:

– активний для завантаження файлу — активується коли користувач переносить документ до меж компонента (рис. 2.15);

– файл завантажено — активується коли користувач перетягнув та залишив документ у компоненті (рис. 2.16).

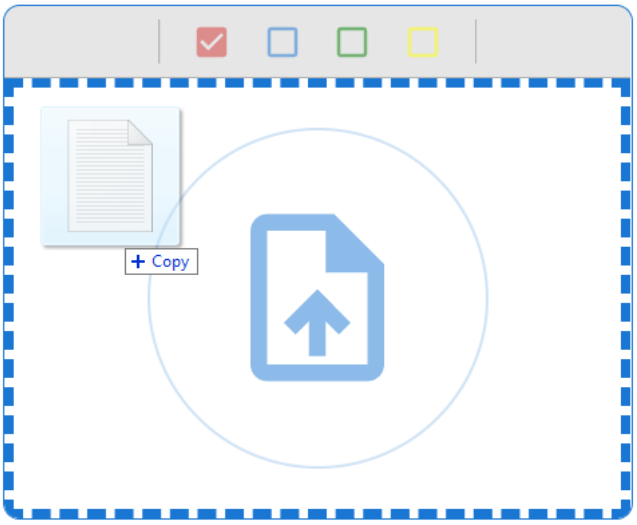

Рис. 2.15. Зображення компонента під час перетягування документа

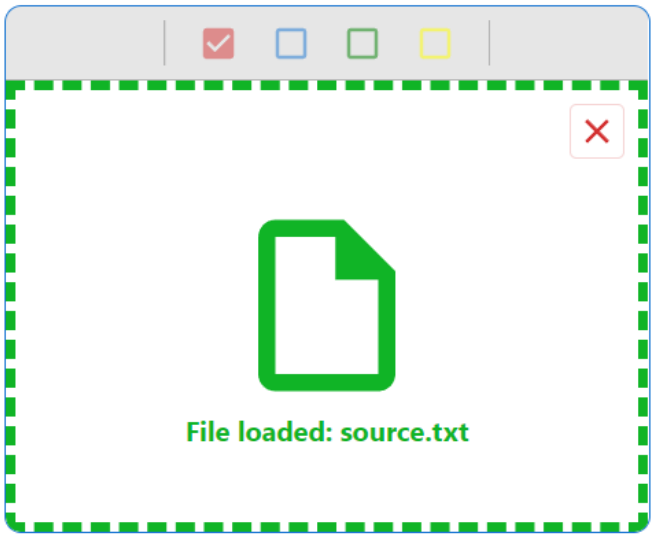

Рис. 2.16. Зображення компонента після завантаження документа

Користувач може скасовувати завантаження документа, для цього необхідно натиснути на кнопку у лівому верхньому куті компоненту (2.17).

Рис. 2.17. Кнопка скасування завантаження документу

У верхній частині компонента для введення даних для пориднення розташовується панель обрання кольору відмінностей (рис. 2.18) характерних даному вхідному значенню. Це означає, якщо у результуючому тексті будуть наявні відмінності з даного вхідного тексту, то вони будуть помічатися обраним кольором (рис. 2.20).

На вибір у користувача представлені наступні кольори: червоний, блакитний, зелений, жовтий.

Для кожного вхідного значення кольори обираються окремо. Кольори не можуть повторюватися.

За замовченням до першого вхідного значення обрано червоний колір, для другого — блакитний.

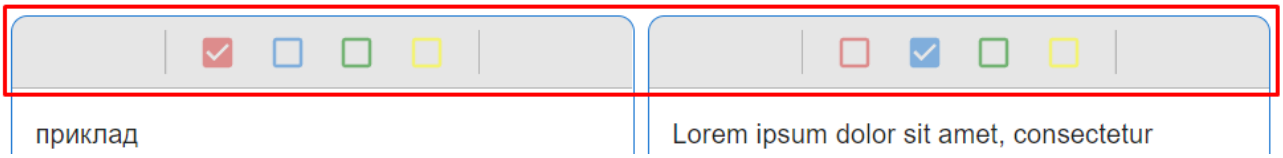

Рис. 2.18. Панель обрання кольору відображення відмінностей

Панель керування

Нижче компонентів для введення даних представлена панель керування (рис. 2.19) яка розділена на дві частини.

a) Ліва частина — відповідає на налагодження зовнішнього представлення результату порівняння;

– merged — результат відображається в одному спільному результуючому тексті (рис. 2.20);

– separate — результат відображається окремо (рис. 2.22).

b) Права частина — відповідає за процес порівняння окремо для кожного вхідного тексту.

– кнопка «Submit» — запустити процес порівняння;

– кнопка «Reset» — очистити вхідні дані;

– кнопка «Test» — представити приклад зображення результату.

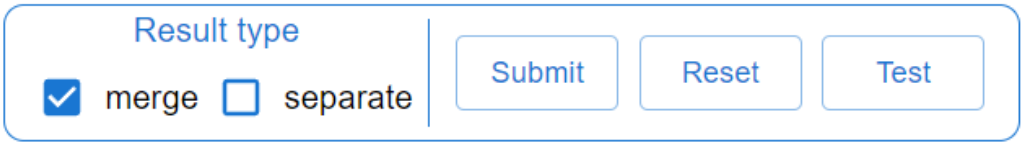

Рис. 2.19. Панель управління

Опис інтерфейсу ведення вихідних даних

Як було зазначено раніше, вихідні дані порівняння можуть бути представлені у вигляді результуючого тексту.

Компонент з вихідними даними не відображається з початку відвідування сторінки, а з’являється в результаті виконання наступних дій:

1. Ввести два тексти в спеціально зазначені компоненти (рис. 2.13).
2. Натиснути кнопку «Submit» на панелі керування (рис. 2.19).

Після виконання вище зазначених дій має з’явитися зображений результат порівняння (рис. 2.20), який розташований нижче панелі керування.

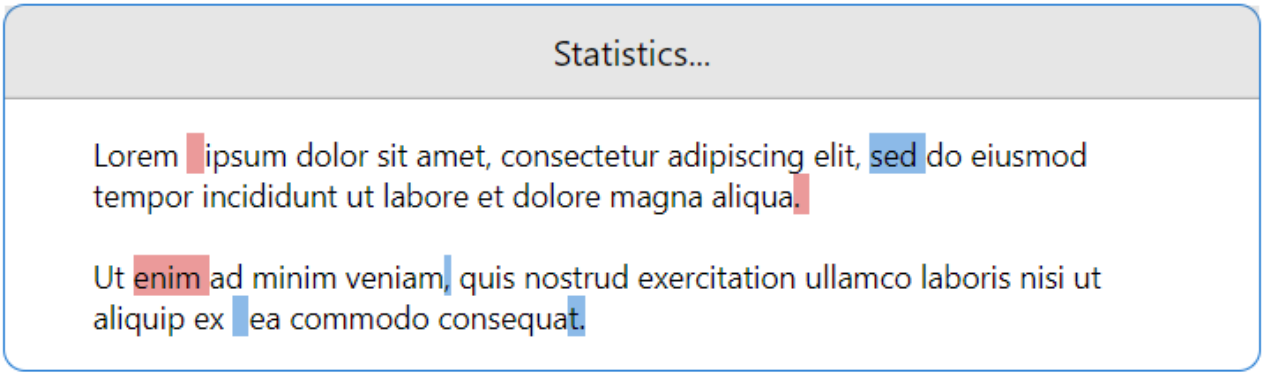

Рис. 2.20. З’єднаний результат порівняння

Слід зазначити що для досягнення результату (рис. 2.18) було обрано кольори за замовчуванням (перший вхідний текст — червоний, другий — блакитний) та тип відображення також за замовчуванням (merged).

Як представлено на рисунку 2.20 — представлений з’єднаний результат порівняння, де візуально відображаються відмінності в залежності від типу вхідних текстів.

У верхній частині компоненту з вихідними даними представлено статистика відносно порівняння (рис. 2.21), а саме:

– загальна кількість змін;
– кількість змін, яка належить до першого вхідного тексту;
– кількість змін, яка належить до другого вхідного тексту.

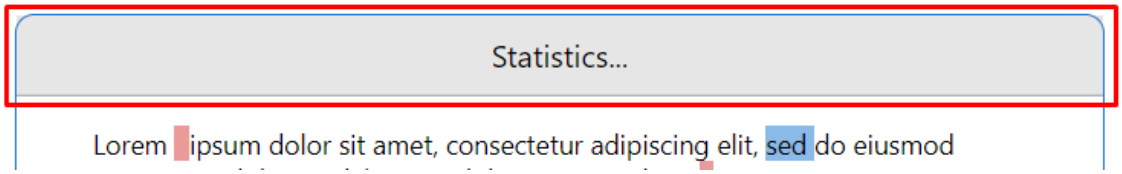

Рис. 2.21. Панель статистики порівняння

Як було зазначено раніше результуючий текст можна представити ще у розділеному вигляді (рис. 2.22), для цього слід обрати тип представлення результату — «separate». Слід зазначити що у такому випадку з’єднаний результат відображатися не буде.

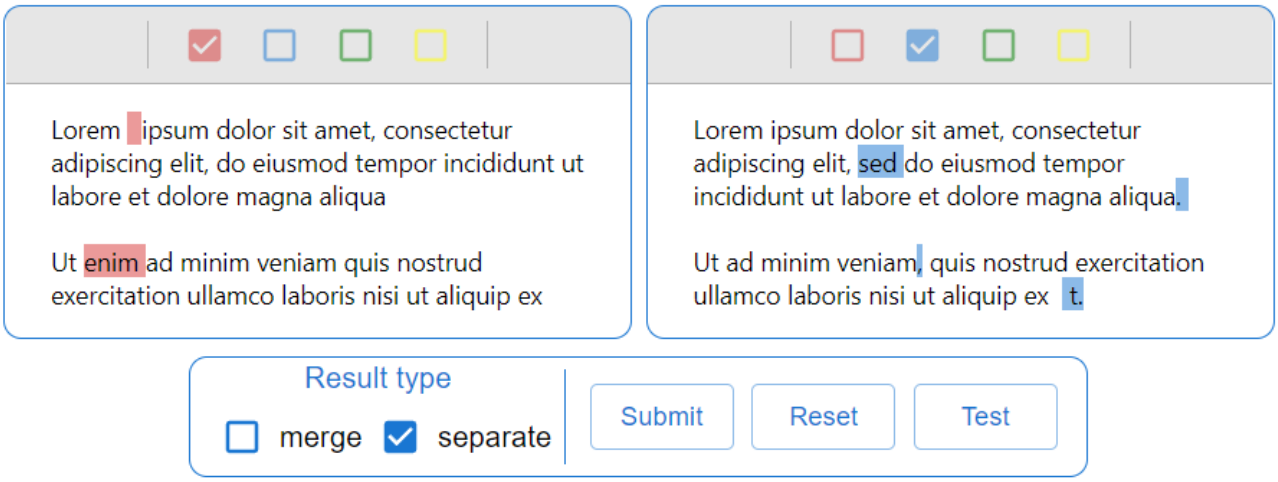

Рис. 2.22. Розділений результат порівняння

Як представлено на рисунку 2.22, результат поділено в залежності з типом вхідних текстів та, відповідно, з відображенням лише притаманних відмінностей.

До вихідних даних також належить окремий список відмінностей (рис. 2.24), з яким також можна взаємодіяти. Для перегляду даного списку необхідно натиснути на кнопку у лівій частині екрана (рис. 2.23) або кнопку «Result list» в панелі «заголовок» у верхній частині (рис. 2.33).

Рис. 2.23. Кнопка розгортання списку відмінностей

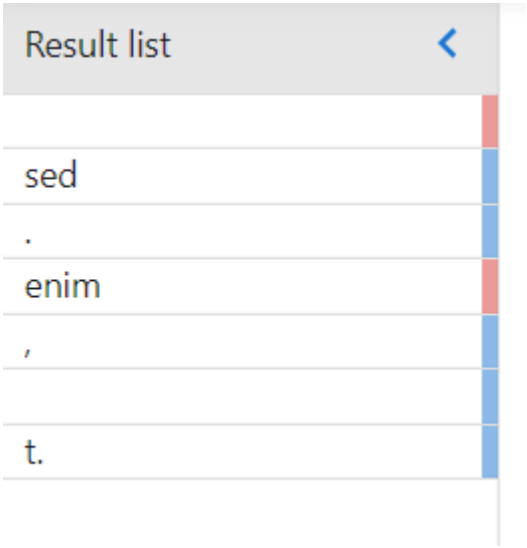

Рис. 2.24. Список відмінностей

У вище зазначеному сипку знаходиться зміст відмінностей. У правій частині кожного елементу списку знаходиться кольоровий маркер, який відображає належність до вхідних даних. Колір маркерів відповідає обраним в панелі налаштування відмінностей (рис. 2.18) та може динамічно змінюватися у процесі використання (2.25).

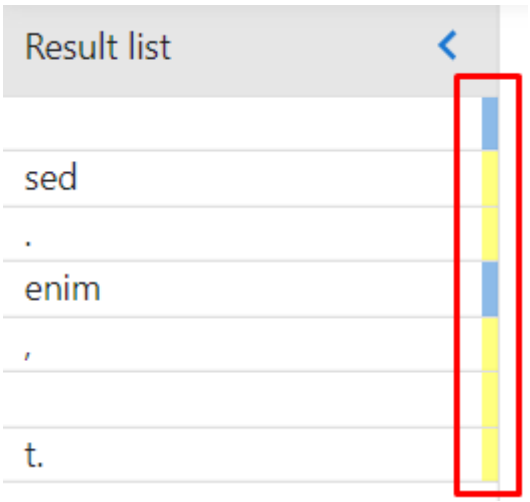

Рис. 2.25. Інші варіанти кольорів відмінностей

Кожний елемент зі списку можна обрати, після чого відповідна відмінність додатково підкреслиться у результуючому тексті (рис. 2.26).

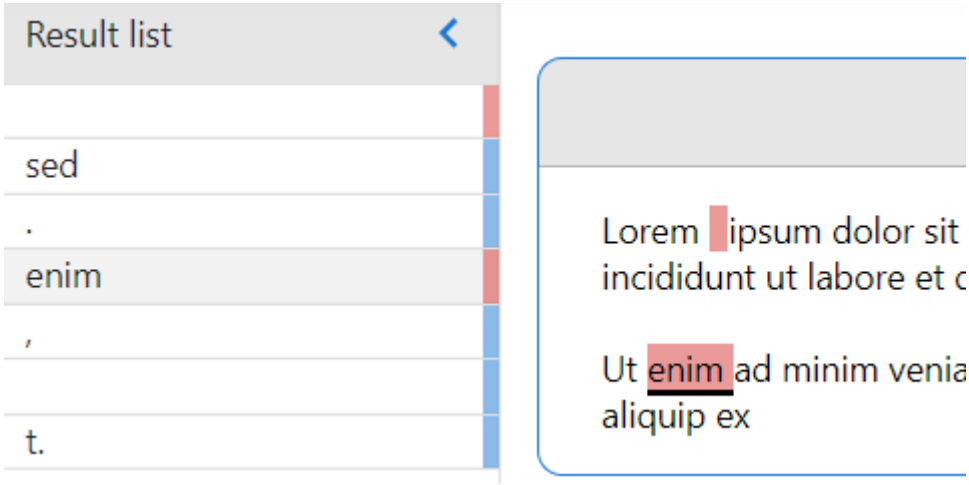

Рис. 2.26. Підкреслення обраної відмінності

Щоб закрити список відмінностей слід натиснути на кнопку у верхній частині списку (рис. 2.27) або відповідну кнопку на панелі «заголовок» (рис. 2.33).

Рис. 2.27. Кнопка згортання списку відмінностей

Також до вихідних даних також належить список збережених результатів (рис. 2.28), з яким також можна взаємодіяти. Для перегляду даного списку необхідно виконати відповідні дії, як для списку відмінностей, але кнопка розташована у правій частині екрана (рис. 2.23) або кнопку «Saved Result list» на панелі «заголовок» (рис. 2.33).

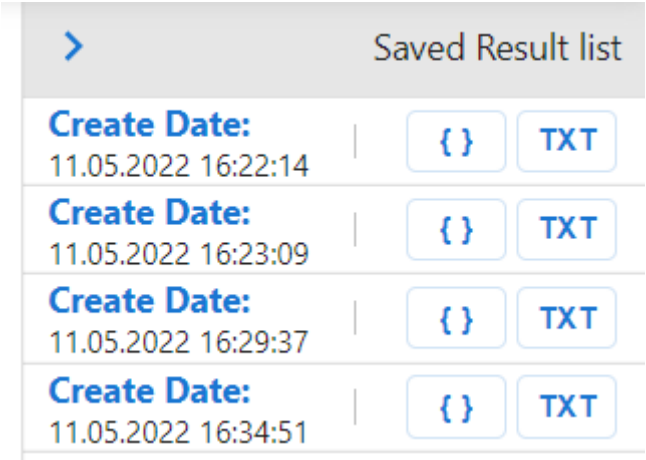

Рис. 2.28. Список збережених результатів порівняння

Даний список доступний тільки для зареєстрованих користувачів. У кожному елементі списку знаходиться:

– дата та час створення результату;

– кнопка відображення структури результату;

– кнопка завантаження результату у форматі «txt» документа.

Слід зазначити що результат зберігається автоматично після натискання кнопки «Submit» на панелі налаштування (2.19).

По натисканню кнопки з символами: «{ }» (рис. 2.29) — має завантажитися результуючий текст (рис. 2.20) та, відповідно, список відмінностей (рис. 2.24). Тобто мають з’явитися вихідні дані, які описані раніше, як після процесу порівняння з якими також можна взаємодіяти.

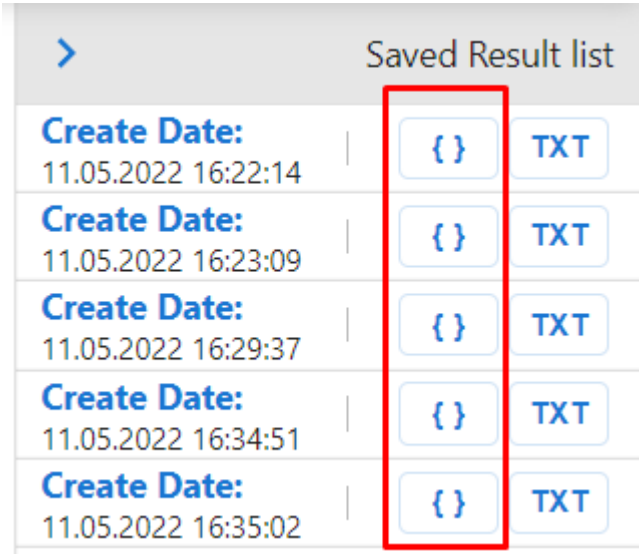

Рис. 2.29. Кнопка відображення структури результату

По натисканню кнопки з текстом: «TXT» (рис. 2.30) — має завантаження на пристрій користувача результат у форматі «txt» документа (рис. 2.31).

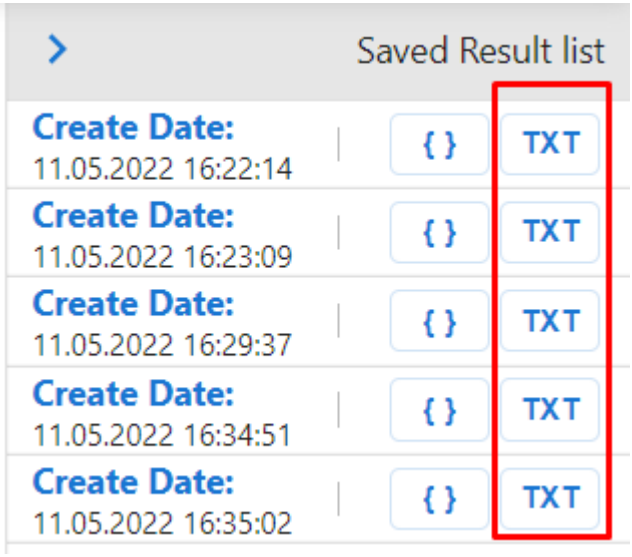

Рис. 2.30. Кнопка завантаження результату у форматі «txt» документа

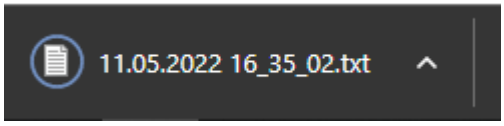

Рис. 2.31. Завантаження результату у форматі «txt» документа

Згортання списку відбувається відповідно списку відмінностей: через кнопку у верхній частині (рис. 2.32) або кнопку на панелі «заголовок» (рис. 2.33).

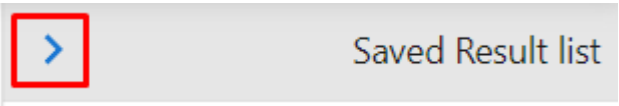

Рис. 2.32. Кнопка згортання списку збережених результатів

Панель «заголовок»

Панель яка постійно знаходиться на фіксованому місці – у верхній частині сторінки (рис. 2.33) та включає в себе:

– логотип;

– панель навігації;

– панель авторизації.

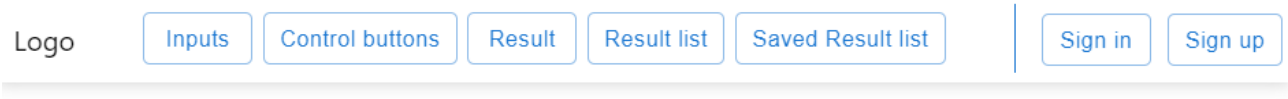

Рис. 2.33. Панель «заголовок»

Панель навігації відповідає за переміщення фокусу на компоненти сторінки які не поміщається на екран користувача, таким чином поліпшується зручність та швидкість навігації по усій сторінці, за рахунок того, що дана панель завжди наявна та фіксується у верхній частині екрану.

Панель авторизації відповідає за процес створення облікового запису та його використання, що дозволяє взаємодіяти зі списком збережених результатів (рис. 2.28).

Авторизація

Авторизація включає в себе три процеси.

– реєстрація — створення нового облікового запису;

– автентифікація — входження в обліковий запис;

– заміна пароля.

Процес реєстрації складається з наступних кроків:

1. На панелі «заголовок» (рис. 2.33) натиснути кнопку «Sign up».

2. Ввести дані у відповідну форму з кроком — 1 (рис. 2.34) про користувача: пошта та пароль.

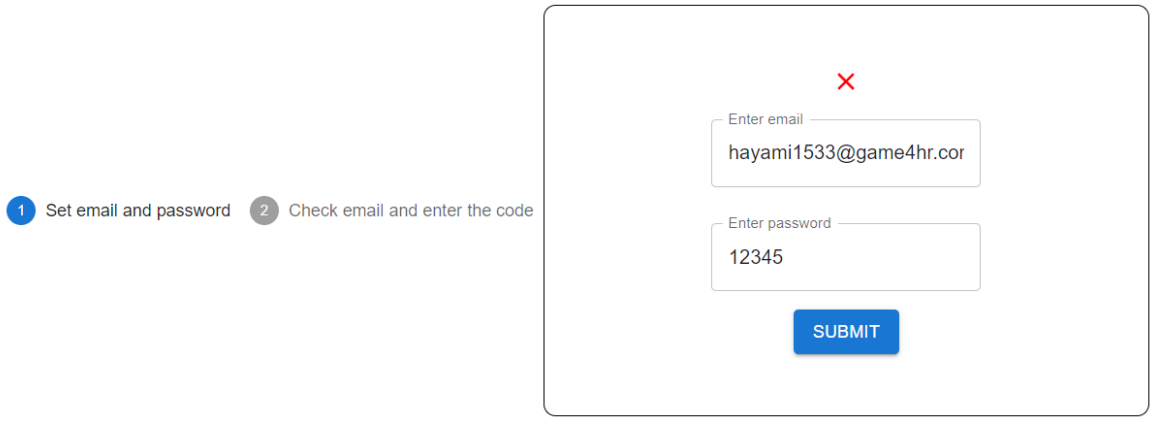

Рис. 2.34. Перший крок реєстрації

3. На вказану пошту має прийти код підтвердження особистості користувача (рис. 2.35).

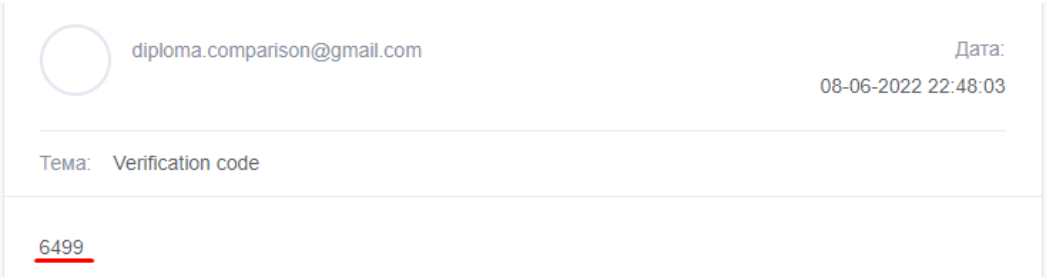

Рис. 2.35. Повідомлення на пошту

4. Ввести код підтвердження у відповідну форму з кроком — 2 (рис. 2.36).

Після виконання останнього кроку, якщо код введено вірно обліковий запис буде створено.

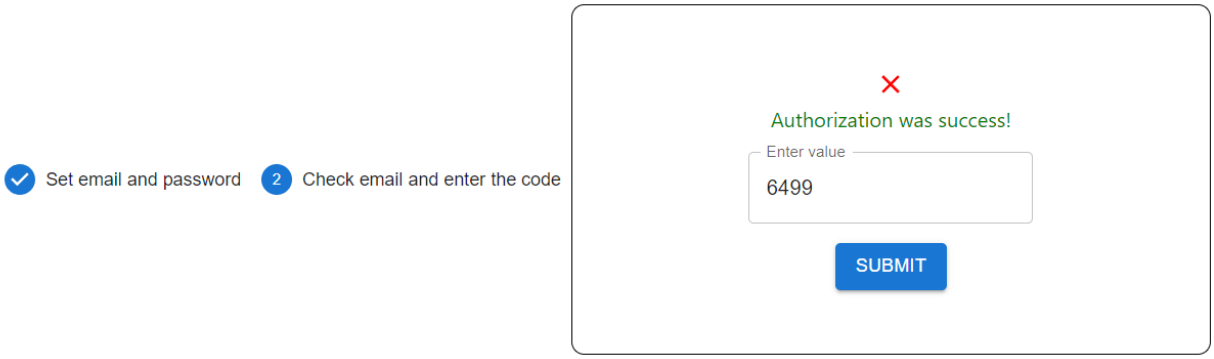

Рис. 2.36. Другий крок реєстрації

Слід зазначити що кожен крок має додатковий опис та стан проходження процесу реєстрації (рис. 2.37).

Рис. 2.37. Панель стану проходження реєстрації

Процес автентифікації складається з наступних кроків:

1. На панелі «заголовок» (рис. 2.33) натиснути кнопку «Sign in».

2. Ввести дані у відповідну форму (рис. 2.38) про обліковий запис: логін (пошта) та пароль.

Якщо введені користувачем дані вірні, то після введення має з’явитися відповідне повідомлення про успішну автентифікацію користувача.

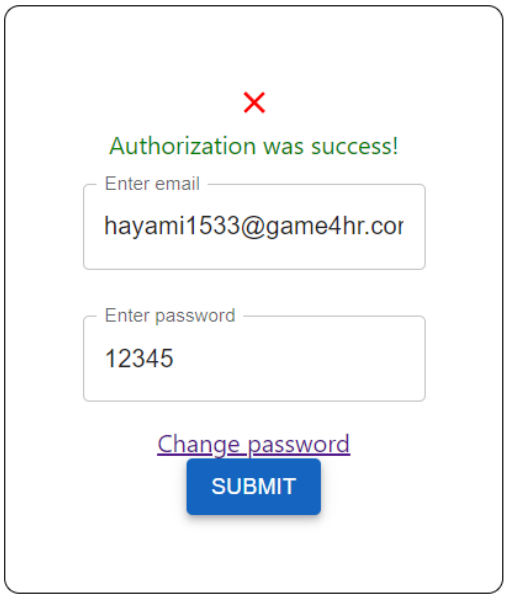

Рис. 2.38. Автентифікація

Процес зміни паролю складається з наступних кроків:

1. На панелі «заголовок» (рис. 2.33) натиснути кнопку «Sign in».
2. На формі автентифікація натиснути кнопку «Change password» (рис. 2.39).

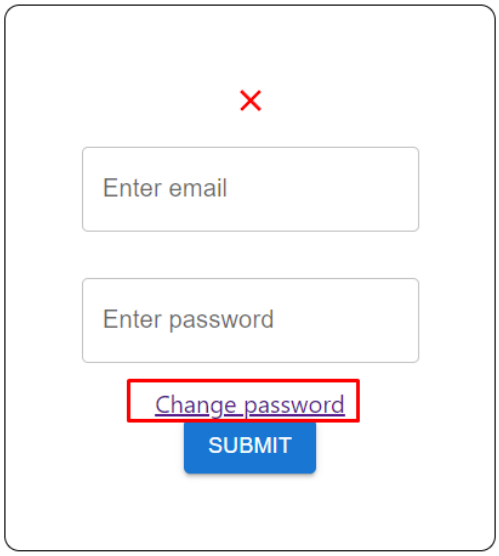

Рис. 2.39. Зміна паролю

3. Ввести логін (пошту) від облікового запису у відповідну форму з кроком — 1 (рис. 2.40).

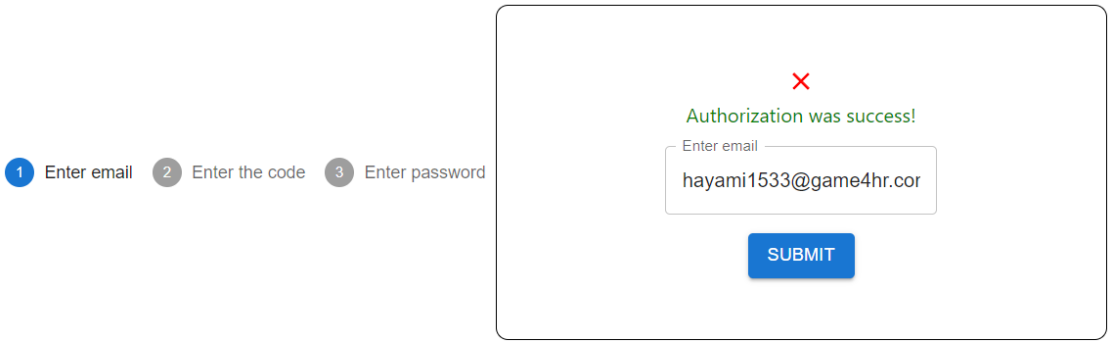

Рис. 2.40. Перший крок зміни паролю

4. Відповідно процесу реєстрації на вказану пошту має прийти код підтвердження особистості користувача (рис. 2.35).

5. Ввести код підтвердження у відповідну форму з кроком — 2 (рис. 2.41).

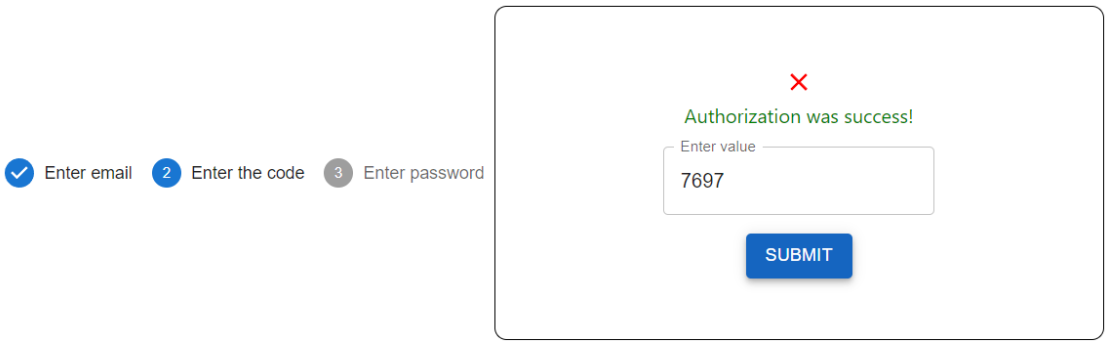

Рис. 2.41. Другий крок зміни паролю

6. Ввести новий пароль від облікового запису у відповідну форму з кроком — 3 (рис. 2.42).

Якщо всі кроки були пройдені успішно, то пароль від облікового запису зміниться та додатково користувач автоматично буде авторизований.

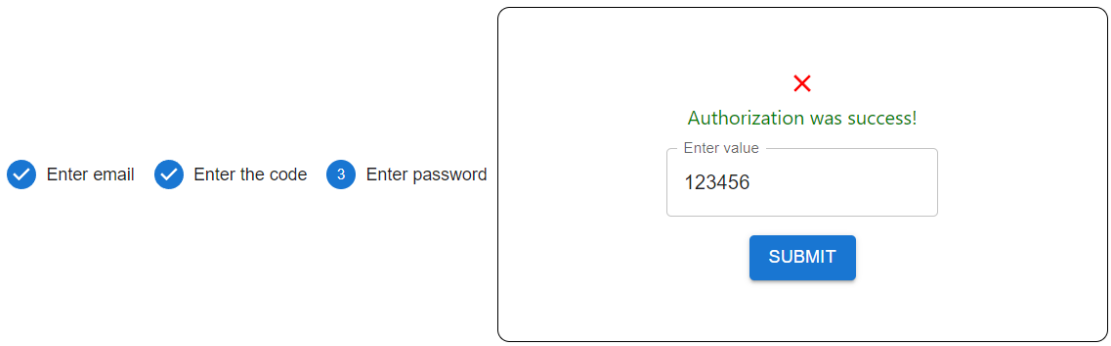

Рис. 2.42. Третій крок зміни паролю

Послідовність використання графічного інтерфейсу для порівняння

Слід зазначити, що для тестової демонстрації процесу порівняння передбачено кнопка на панелі керування «Test» (рис. 2.19), за допомогою якої можна пропустити етап введення даних та отримати тестовий результат порівняння з можливістю взаємодії.

Нище наведено приклад послідовного виконання кроків для виконання процесу порівняння двох текстів:

– перший текст (source): «Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua Ut enim ad minim veniam quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequa»;

– другий текст (target): «Lorem ipsum dolor sit amet, consectetur adipiscing elit, do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.».

Послідовність використання графічного інтерфейсу:

1. Введення вхідних текстів у відповідні компоненти (рис. 2.43).

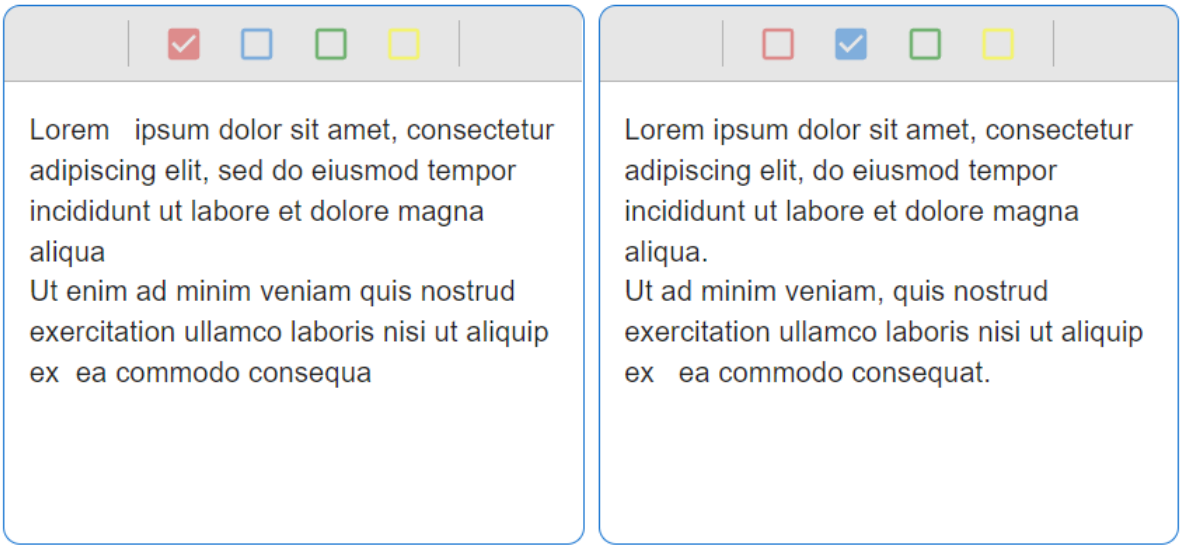

Рис. 2.43. Приклад введення вхідних текстів

2. За бажанням можна обрати колір відображення відмінностей (рис. 2.18-19) та тип результату порівняння (за замовченням колір відмінностей: перший вхідний текст — червоний, другий — блакитний; тип результату — merged).

3. Натиснути кнопку «Submit» на панелі керування (рис. 2.19).

4. Має з'явитися результівний текст порівняння, де відмінності помічаються відповідними кольорами (рис. 2.44).

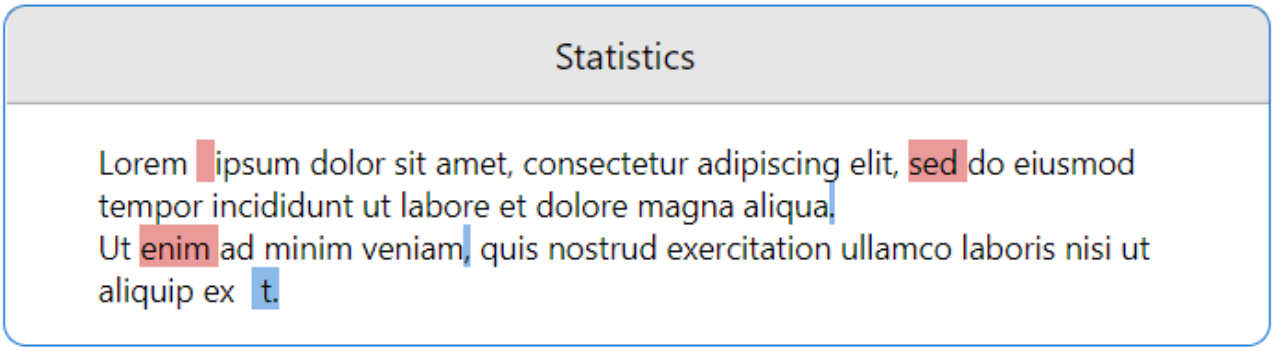

Рис. 2.44. Результат порівняння

5. Також для додаткової взаємодії з результатом можна розгорнути список відмінностей (рис. 2.45) за допомогою кнопки у лівій частині екрана (рис. 2.23) або кнопки «Result List» на панелі «заголовок» (рис. 2.33).

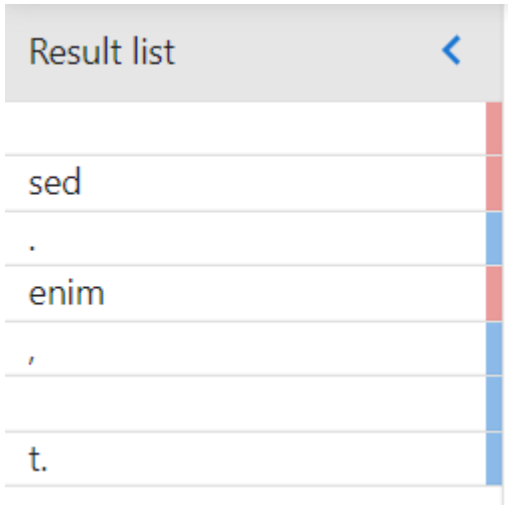

Рис. 2.45. Список відмінностей

6. Кожний елемент зі списку можна обрати, після чого відповідна відмінність додатково підкреслиться у результуючому тексті (рис. 2.46).

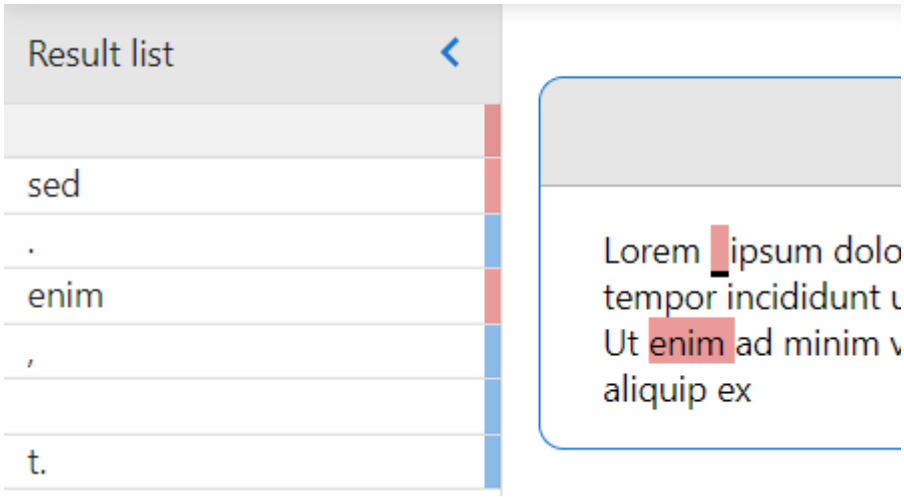

Рис. 2.46. Підкреслення обраної відмінності

Таким чином виконується базовий процес порівняння, в результаті якого користувач отримую наглядовий результат порівняння у декількох представлених та можливість взаємодіяти з вихідними даними.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок вхідних даних

1. q – передбачуване число операторів — 2374.
2. C – коефіцієнт складності — 1,85.
3. p – коефіцієнт корекції програми у ході її розробки — 0,072.
4. B – коефіцієнт збільшення витрат — 1,33.
5. k – коефіцієнт кваліфікації програміста — 1,17.

Обґрунтовування значення:

– професійний досвід — майже два роки;

– досвід набутий в процесі навчання у коледжі та університеті загалом складає — 7 років.

Ґрунтуючись на цих даних було виставлено коефіцієнт – 1,17.

6. $C_{ПР}$ — середня годинна заробітна плата програміста, грн/год. Згідно з середнього значення погодинної оплати $C_{ПР}$ = 98,2 грн/год.

Обґрунтовування значення — розрахунок було здійснено на основі моніторингу статистики[25], де використовувались наступні значення:

– рівень кваліфікації програміста: Junior Software Engineer;

– технології: C#/.NET та JavaScript;

– досвід: від 1 до 2 років.

– рівень знання англійської мови: «Elementary»-«Pre-Intermediate».

Слід зазначити, що значення для розрахунку брались відповідно до рівня складності та потреб кваліфікаційної роботи.

7. B_k — число виконавців — 1.

8. F_p — місячний фонд робочого часу (при 40 годинному робочому тижні F_p = 340 годин).

9. C_{MB} — вартість машино-години ЕОМ, грн/год. Згідно з середнім значенням споживаної потужності обчислювальної машини та вартості електроенергії C_{MB} = 0,66 грн/год.

Обґрунтовування значення – розрахунок вхідних значень:

a) d — загальне середнє значення споживчої потужності = 395 Вт/год або 0,395 кВт/год:

- обчислювальна машина — 340 Вт/год;
- монітор (у кількості двох одиниць) — 52 Вт/год;
- wifi роутер – 3 Вт/год;

b) e — вартість електроенергії = 1,68 грн за 1 кВт [26];

c) i — вартість інтернету = 0,62 грн/год;

d) m — амортизація обчислювальної машини = 3,2 грн/год.

Формула для розрахунку $C_{ПР}$ (3.1).

$$C_{MB} = d\ (\text{кВт/год}) \cdot e + i + m \qquad (3.1)$$

Використовуючи формулу (3.1) було розраховано $C_{ПР}$:

$$C_{MB} = 0{,}395 \cdot 1{,}68 + 0{,}62 + 3{,}2 = 4{,}48 \text{ грн/год.}$$

3.2. Визначення трудомісткості розробки програмного забезпечення

Опис процесу розрахунку

Розрахунок трудомісткості розробки програмного забезпечення здійснюється за допомогою формул:

Трудомісткість розробки ПЗ, формула (3.2):

$$t = t_o + t_u + t_a + t_n + t_{\text{отл}} + t_{\text{д}} \text{ , людино-годин,} \qquad (3.2)$$

де t_o — витрати праці на підготовку й опис поставленої задачі,

t_u — витрати праці на дослідження алгоритму рішення задачі,

t_a — витрати праці на розробку блок-схеми алгоритму,

t_n — витрати праці на програмування по готовій блок-схемі,

$t_{\text{отл}}$ — витрати праці на налагодження програми на ЕОМ,

$t_{\text{д}}$ — витрати праці на підготовку документації.

Умовне число операторів (підпрограм), формула (3.3):

$$Q = q \cdot C \cdot (1 + p), \tag{3.3}$$

де q — передбачуване число операторів,

C — коефіцієнт складності програми,

p — коефіцієнт кореляції програми в ході її розробки.

Витрати праці на вивчення опису задачі t_u, формула (3.4).

$$t_u = \frac{Q \cdot B}{(75...85) \cdot k}, \text{ людино-годин.} \tag{3.4}$$

де В — коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі,

k — коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.

Витрати праці на розробку алгоритму рішення задачі t_a та складання програми по готовій блок-схемі t_n, формула (3.5):

$$t_{a,n} = \frac{Q}{(20...25) \cdot k}, \text{ людино-годин.} \tag{3.5}$$

Витрати праці на налагодження програми на ЕОМ, формули (3.6-3.7).

За умови автономного налагодження одного завдання:

$$t_{\text{отл}} = \frac{Q}{(4...5) \cdot k}, \text{ людино-годин,} \tag{3.6}$$

За умови комплексного налагодження завдання:

$$t_{\text{отл}}^{k} = 1{,}5 \cdot t_{\text{отл}}, \text{ людино-годин,} \tag{3.7}$$

Витрати праці на підготовку документації, формули (3.8-3.10):

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}}, \text{ людино-годин,} \tag{3.8}$$

де $t_{\text{др}}$ — трудомісткість підготовки матеріалів і рукопису, формула (3.9).

$$t_{\text{др}} = \frac{Q}{(20...25) \cdot k}, \text{ людино-годин.} \quad (3.9)$$

де $t_{\text{др}}$ — трудомісткість підготовки матеріалів і рукопису, формула (3.10).

$$t_{\text{до}} = 0{,}75 + t_{\text{др}}, \text{ людино-годин,} \quad (3.10)$$

Розрахунок

1. Розраховано умовне число операторів (підпрограм), за формулою (3.3).

$$Q = 2374 \cdot 1{,}85 \cdot (1 + 0{,}072) = 4\,708$$

2. Розраховано витрати праці на вивчення опису задачі t_u, за формулою (3.4).

$$t_u = \frac{4\,708 \cdot 1{,}33}{84 \cdot 1{,}17} = 63{,}7$$

3. Розраховано витрати праці на розробку алгоритму рішення задачі, за формулою (3.5).

$$t_a = \frac{4\,708}{23{,}6 \cdot 1{,}17} = 170{,}5$$

4. Розраховано витрати на складання програми по готовій блок-схемі, за формулою (3.5).

$$t_n = \frac{4\,708}{24{,}7 \cdot 1{,}17} = 162{,}9$$

5. Розраховано витрати праці на комплексного налагодження програми на ЕОМ, за формулою (3.6-3.7).

$$t_{\text{отл}} = \frac{4\,708}{4.1 \cdot 1{,}17} = 981{,}4$$

$$t_{\text{отл}}^{k} = 1{,}5 \cdot 1\,556{,}2 = 1\,472{,}2$$

6. Розраховано витрати праці на підготовку документації, за формулами (3.8-3.10).

$$t_{\text{др}} = \frac{4\ 708}{22{,}3 \cdot 1{,}17} = 180{,}4$$

$$t_{\text{до}} = 0{,}75 + 180{,}4 = 181{,}2$$

$$t_{\text{д}} = 180{,}4 + 181{,}2 = 361{,}6$$

7. Розраховано трудомісткість розробки ПЗ за формулою (3.2).

$t = 50 + 63{,}7 + 170{,}5 + 162{,}9 + 1\ 472{,}2 + 361{,}6 = 2\ 280{,}9$ людино-годин

3.3. Витрати на створення програмного забезпечення

Опис процесу розрахунку

Розрахунок витрат на створення програмного забезпечення здійснюється за допомогою формул:

Витрати на створення ПЗ K_{no} включають витрати на заробітну плату виконавця програми $З_{з/п}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ, формула (3.11):

$$K_{\text{ПО}} = З_{\text{ЗП}} + З_{\text{МВ}}\text{ , грн.} \qquad (3.11)$$

Заробітна плата виконавців, формула (3.12):

$$З_{\text{ЗП}} = t \cdot C_{\text{ПР}}\text{ , грн.} \qquad (3.12)$$

де t — загальна трудомісткість, людино-годин,

$C_{\text{ПР}}$ — середня годинна заробітна плата програміста, грн/година.

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, формула (3.13):

$$З_{\text{мв}} = t_{\text{отл}} \cdot C_{\text{МВ}}\text{, грн,} \qquad (3.13)$$

де $t_{\text{отл}}$ — трудомісткість налагодження програми на ЕОМ, год.,

$C_{МВ}$ — вартість машино-години ЕОМ, грн/год.

Очікуваний період створення ПЗ, формула (3.14):

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс}, \tag{3.14}$$

де B_k — число виконавців,

F_p — місячний фонд робочого часу (при 40 годинному робочому тижні Fp=176 годин).

Розрахунок

1. Розраховано заробітна плата виконавців, за формулою (3.12).

$$З_{ЗП} = 2\,280{,}9 \cdot 98{,}2 = 223\,984{,}38 \text{ грн}$$

2. Розраховано вартість машинного часу, за формулою (3.13).

$$З_{мв} = 1\,472{,}2 \cdot 4{,}48 = 6\,597{,}7 \text{ грн}$$

3. Розраховано витрати на створення ПЗ виконавців, за формулою (3.11).

$$K_{ПО} = 223\,984{,}38 + 6\,597{,}7 = 230\,582{,}08 \text{ грн}$$

4. Розраховано очікуваний період створення ПЗ, за формулою (3.14).

$$T = \frac{2\,280{,}9}{1 \cdot 340} = 6{,}7 \text{ міс}$$

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено веб-додаток з чітко розмежованими частинами:

– графічний інтерфейс користувача;

– серверна частина для обробки даних, процесу порівняння, авторизації, взаємодії з базою даних та іншими сервісами.

Що у поєднанні надає користувачу можливість введення початкових даних у декількох форматах:

вручну ввести текст;

завантажити документ текстового формату.

Виконувати процес порівняння та отримувати наглядову результуючу інформацію у наступних представленнях:

– результат порівняння у форматі тексту;

– список відмінностей;

– список збережених результатів.

З можливістю налаштувати тип відображення та додатково взаємодіяти з вихідних даними.

Та надає замовнику, після вкладання зазначеної в економічній частині суми грошей та часу, самостійно працездатний застосунок, у якому ведеться інформація про користувачів, їх потреби у використанні, активність та зацікавленість предметною галуззю, що у майбутньому дозволить використовувати розроблені частини у більш великих розробках або надалі, зарекомендувавши себе на ринку, розширити даний застосунок новим функціоналом з комерційними елементами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. CodeTools & Tips, HTTP: Протокол, який повинен розуміти кожний веб-розробник (Частина 1) [Електронний ресурс] / Паван Поділа, 8 квітня 2013 р. (Оновлено 30 січня 2022 р.) — Режим доступу до ресурсу: https://code.tutsplus.com/uk/tutorials/http-the-protocol-every-web-developer-must-know-part-1--net-31177.

2. What Is a Web Application? How It Works, Benefits and Examples [Електронний ресурс] / Indeed Editorial Team, November 10, 2021 — Режим доступу до ресурсу: https://www.indeed.com/career-advice/career-development/what-is-web-application.

3. JavaScript, Standard ECMA-262 [Електронний ресурс] — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JavaScript.

4. Node.js [Електронний ресурс] / Офіційне джерело розробників з поточною версією: v18 від 2022-04-19 — Режим доступу до ресурсу: nodejs.org/uk/about/.

5. Proposal for making AJAX crawlable [Електронний ресурс]/ Google. October 7, 2009 / Retrieved July 13, 2011 — Режим доступу до ресурсу: developers.google.com/search/blog/2009/10/proposal-for-making-ajax-crawlable.

6. Server Side Rendering for Meteor". Archived from the original on March 20, 2015. Retrieved January 31, 2015.

7. React — JavaScript-бібліотека для створення користувацьких інтерфейсів [Електронний ресурс] / Офіційне джерело розробників з поточною стабільною версією: 16.13.1 (March 19, 2020) — Режим доступу до ресурсу: https://uk.reactjs.org/.

8. State of frontend 2022 [Електронний ресурс] / Автор статті: Aleksandra Dobrowska — Режим доступу до ресурсу: https://tsh.io/state-of-frontend/#frameworks

9. Popular Web Frameworks for Web App Development [Електронний ресурс] / Автор статті: Jeel Patel — Режим доступу до ресурсу: https://www.monocubed.com/blog/most-popular-web-frameworks/.

10. Redux — Контейнер передбачуваного стану для JS Apps [Електронний ресурс] / Офіційне джерело розробників з поточною версією: 4.2.0 — Режим доступу до ресурсу: https://redux.js.org/.

11. Material UI — бібліотека компонентів інтерфейсу React, реалізує Material Design від Google [Електронний ресурс] / Офіційне джерело розробників з поточною версією: 5.8.3 — Режим доступу до ресурсу: https://redux.js.org/. https://mui.com.

12. Single Page Application: Dispelling SEO Myths | Hacker Noon [Електронний ресурс] / Retrieved 2021-12-15 — Режим доступу до ресурсу: https://hackernoon.com/single-page-application-dispelling-seo-myths

13. Чеглаков А. Л. Композиція web-сервісів на основі архітектури rest [Електронний ресурс] / А. Л. Чеглаков., 2016. — Режим доступу до ресурсу: https://cyberleninka.ru/article/n/kompozitsiya-web-servisov-na-osnove-arhitektury-rest/viewer.

14. C# (See Sharp) мова програмування, .NET [Електронний ресурс] / Стаття видана 18.03.2022 — Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp.

15. Price M. C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development: Build applications with C#, .NET Core, Entity Framework Core, ASP.NET Core, and ML.NET using Visual Studio Code, 4th Edition / Mark Price., 2021. – 816 с. – (4).

16. Алекс Макки. Введение в .NET 4.0 и Visual Studio 2010 для профессионалов / Introducing .NET 4.0: with Visual Studio 2010. — М.: «Вильямс».

17. ASP.NET Core — Modern Cross-Platform Development: Build applications with C# [Електронний ресурс] Деніел Рот, Рік Андерсон та Шон Латтін, 06/04/2022 — Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core.

18. Ріхтер Д. CLR via C#. Програмування на платформі Microsoft .NET Framework 4.0 на мові C# / Джеффрі Ріхтер. – Пітер, 2013. – 928 с.

19. WEB API [Електронний ресурс] / Рік Андерсон та Кірк Ларкін, 06/04/2022 — Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-6.0&tabs=visual-studio.

20. Microsoft SQL Server 2019 Express, SQL, Database [Електронний ресурс] — Режим доступу до ресурсу: https://www.microsoft.com/en-us/Download/details.aspx?id=101064.

21. API [Електронний ресурс] — Режим доступу до ресурсу: https://avada-media.ua/ua/api/.

22. Price M. C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development: Build applications with C#, .NET Core, Entity Framework Core, ASP.NET Core, and ML.NET using Visual Studio Code, 4th Edition / Mark Price., 2021. – 816 c. – (4).

23. Пауэрс Л., Снелл М. Microsoft Visual Studio 2008 / Microsoft Visual Studio 2008 Unleashed by Lars Powers and Mike Snell. — С.: «БХВ-Петербург», 2008. — 1200с.

24. Відстань Левенштейна, функція Левенштейна, алгоритм Левенштейна, відстань редагування [Електронний ресурс] — Режим доступу до ресурсу: uk.wikipedia.org/wiki/Відстань_Левенштейна.

25. DOU / jobs, salaries, Junior Software Engineer, C#/.NET, JavaScript [Електронний ресурс] — Режим доступу до ресурсу: https://jobs.dou.ua/salaries/?period=2021-12&position=Middle%20SE.

26. МІНІСТЕРСТВО ЕНЕРГЕТИКИ УКРАЇНИ [Електронний ресурс] / Стаття видана 30.09.2021 | 10:35 — Режим доступу до ресурсу: http://mpe.kmu.gov.ua/minugol/control/publish/article?art_id=245583544.

ДОДАТОК А

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Таблиця А.1

Опис файлів

Ім’я файлу	**Опис**
Пояснювальні документи	
Кваліфікаційна_робота_Гордієнко_121-19ск-1.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна_робота_Гордієнко_121-19ск-1.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
Програма	
Program.rar	Архів. Містить коди програми.
Презентація	
Презентація_Гордієнко_121-19ск-1.pptx	Презентація кваліфікаційної роботи.

ДОДАТОК Б

ВИХІДНИЙ КОД ПРОГРАМИ

Лістинг Б.1

Компонент графічного інтерфейсу – «ComparisonForm»

```
import  React from "react";
import { connect } from "react-redux";
import { Grid } from "@mui/material";

import { resultType } from "../../interfaces";
import * as actionsComparison from "../../actions/Comparison";
import useForm from "./useForm";

import Form from "./Form";
import ResultBox from "../ResultBox/ResultBox";

import "./ComparisonForm.scss";

const ComparisonForm = (props) => {
   const {
      activeChange,
      colors,
      handleChangeColors,
      styleProps,
   } = props;

   const {
      inputs,
      resultSettings,
      handleForm,
      handleResult
   } = useForm(
      props.resultList,
      props.resultString,
      props.getStaticResultArray,
      props.getResultByString,
      props.getResultByFiles
   )

   return (
      <Grid container className="box">
         <Grid item xs={10} className="grid-form">
            <Form
               resultList={props.resultList}
               resultString={props.resultString}

               inputs={inputs}
               colors={colors}
               resultSettings={resultSettings}
               resultType={resultType}
               activeChange={activeChange}
               styleProps={styleProps}

               handleForm={handleForm}
               handleResult={handleResult}
               handleChangeColors={handleChangeColors}
            />
            <ResultBox
```

```
                resultList={props.resultList}

                activeChange={activeChange}
                resultSettings={resultSettings}

                resultType={resultType}
                styleProps={styleProps}
              />
            </Grid>
          </Grid>
        )
}

const mapStateToProps = (state) => ({
    resultList: state.ComparisonReducer.result,
    resultString: state.ComparisonReducer.resultString,
})
const mapActionToProps = {
    getStaticResultArray: actionsComparison.getStaticResultArray,
    getResultByString: actionsComparison.getResultByString,
    getResultByFiles: actionsComparison.getResultByFiles,
}

export default connect(mapStateToProps, mapActionToProps)(ComparisonForm);

// ComparisonForm.scss file
.box {
    width: 100%;
    height: 100%;
    display: flex;
    justify-content: center;

    padding-top: 80px;

    .grid-form {
        display: flex;
        flex-direction: column;
        align-items: center;
    }
}

// Form.js file
import { Button, Grid } from "@mui/material";

import Input from "../DragDropFile/Input";
import ColorChange from "../changeColor/ChangeColor";
import SelectionResult from "../SelectionResult/SelectionResult";
import ResultInputBox from "../ResultInputBox/ResultInputBox";

import "./Form.scss";

const Form = (props) => {
    const {
        resultList,
        resultString,
        colors,
        resultSettings,
        resultType,
        activeChange,
        styleProps,
        inputs,
        handleForm,
        handleResult,
```

```
    handleChangeColors
} = props;

return (
    <form autoComplete="off" noValidate onSubmit={handleForm.handleSubmit} className="form">
        <Grid container className="inputsPanel">
            <Grid item xs={5.8} className="inputWrapper">
                <ColorChange
                    name="source"
                    selectedValue={colors.source}
                    handleChangeColors={handleChangeColors}
                    className="colorPanel"
                />
                { (resultSettings.resultType === resultType.separate && resultSettings.isDisplay) ? (
                    <ResultInputBox
                        resultList={resultList}
                        activeChange={activeChange}
                        type="source"
                        styleProps={styleProps}
                    />
                ) : (
                    <Input
                        name="source"
                        label="Source"
                        value={inputs.source}
                        handleChange={handleForm.handleInputChange}
                        saveFiels={handleForm.handleSaveFiels}
                    />
                )}
            </Grid>
            <Grid item xs={5.8} className="inputWrapper">
                <ColorChange
                    name="target"
                    selectedValue={colors.target}
                    handleChangeColors={handleChangeColors}
                    className="colorPanel"
                />
                { (resultSettings.resultType === resultType.separate && resultSettings.isDisplay) ? (
                    <ResultInputBox
                        resultList={resultList}
                        activeChange={activeChange}
                        type="target"
                        styleProps={styleProps}
                    />
                ) : (
                    <Input
                        name="target"
                        label="Target"
                        value={inputs.target}
                        handleChange={handleForm.handleInputChange}
                        saveFiels={handleForm.handleSaveFiels}
                    />
                )}
            </Grid>
        </Grid>
        <div className="formControlPanelWrapper">
            <Grid container className="formControlPanel">
                <Grid item xs={5} className="resultTypePanelWrapper">
                    <SelectionResult
                        name={"resultType"}
                        value={resultSettings.resultType}
                        handleResultType={handleResult.handleResultSettings}
                    />
```

```
                </Grid>
                <div className="splitLine"></div>
                <Grid item xs={6.9} className="controlPanel">
                  <Button
                    variant="outlined"
                    color="primary"
                    type="submit"
                  >
                    Submit
                  </Button>
                  <Button
                    variant="outlined"
                  >
                    Reset
                  </Button>
                  <Button
                    variant="outlined"
                    onClick={ () => { handleResult.handleStaticResult() }}
                  >
                    Test
                  </Button>
                </Grid>
              </Grid>
            </div>
          </form>
    )
};

export default Form;

// Form.scss file
.form {
    width: 100%;

    .inputsPanel {
        display: flex;
        justify-content: center;
        align-items: stretch;

        & > div {
            box-sizing: border-box;
        }

        .inputWrapper {
            max-width: 550px;
            box-sizing: border-box;
            overflow: hidden;
            margin: 0 4px;
            border: 1px solid #1976d2;
            border-radius: 10px;

            .colorPanel {
                display: flex;
                justify-content: center;

                border-bottom: 0.5px solid rgba(128, 128, 128, 0.5);
                background-color: rgba(128, 128, 128, 0.2);

                &:before {
                    content: "";
                    margin: 8px 10px;
                    border-left: 1px solid rgba(128, 128, 128, 0.5);
                }
```

```
            &:after {
                content: "";
                margin: 8px 10px;
                border-right: 1px solid rgba(128, 128, 128, 0.5);
            }
        }

        &:hover {
            border-color: #1976d2;
        }

        &:focus-within {
            border-color:rgb(0, 0, 0)
        }
    }
  }

  .formControlPanelWrapper {
    width: 100%;
    display: flex;
    justify-content: center;

    .formControlPanel {
        width: 500px;
        margin: 10px 0;
        border: 1px solid #1976d2;
        border-radius: 10px;
        box-sizing: border-box;

        .resultTypePanelWrapper {
            width: 100%;
            display: flex;
            justify-content: center;
        }

        .splitLine {
            width: 1px;
            margin: 6px 0;
            background-color: #1976d2;
        }

        .controlPanel {
            display: flex;
            justify-content: center;
            align-items: center;


            & > button {
                width: 80px;
                margin: 0 5px;
                text-transform: none;
            }
        }
    }
  }
}


// useForm.js file
import  React, { useState, useEffect } from "react";
import { inputValues, fileInputs, inputsType, resultType, displayResult } from "../../interfaces";
```

```
const useForm = (
    resultList,
    resultString,
    getStaticResultArray,
    getResultByString,
    getResultByFiles,
    getSavedResults,
    getSavedResultJson
) => {

    const [inputs, setInputs] = useState(inputValues)
    const [files, setFiles] = useState(fileInputs)

    const [resultSettings , setResultSettings] = useState(displayResult)

    useEffect(() => {
        if (!files.source && !files.target) {
            if (resultSettings.currentResult === inputsType.file) {
                setResultSettings({
                    ...resultSettings,
                    ...{
                            isDisplay: false,
                            canChangeType: true
                        }
                });
            }
            else {
                setResultSettings({
                    ...resultSettings,
                    ...{canChangeType: true}
                });
            }
        }
    }, [files])

    const handleForm = {
        handleSubmit: (event) => {
            event.preventDefault()

            const formData = new FormData()

            formData.append("FormFiles", files.source)
            formData.append("FormFiles", files.target)

            // console.log("files.source:", files.source)
            // console.log("files.target:", files.target)

            if (files.source && files.target) {
                const onSuccess = () => {
                    setResultSettings({
                        ...resultSettings,
                        ...{
                                isDisplay: true,
                                currentResult: inputsType.file
                            }
                    });

                    getSavedResults()
                }

                getResultByFiles(formData, onSuccess);
            }
            else {
```

```
            const onSuccess = () => {
                setResultSettings({
                    ...resultSettings,
                    ...{
                        isDisplay: true,
                        currentResult: inputsType.text
                    }
                });

                getSavedResults()
            }

            getResultByString(inputs, onSuccess);
        }
    },

    handleInputChange: (event) => {
        const { name, value } = event.target
        const fieldValue = { [name]: value }

        setInputs({
            ...inputs,
            ...fieldValue
        });

        setResultSettings({
            ...resultSettings,
            ...{isDisplay: false}
        });
    },

    handleSaveFiels: (nameValue, file) => {
        const name = nameValue
        const value = file
        const fieldValue = { [name]: value }

        setFiles({
            ...files,
            ...fieldValue
        });

        setResultSettings({
            ...resultSettings,
            ...{
                    resultType: resultType.merge,
                    canChangeType: false,
                }
        });
    },
}

const handleResult = {
    handleResultSettings: (event) => {
        if (resultSettings.canChangeType) {
            const { name, value } = event.target
            const setting = { [name]: value }

            setResultSettings({
                ...resultSettings,
                ...setting
            });
        }
    },
```

```
        handleStaticResult: () => {
            const onSuccess = () => {
                setResultSettings({
                    ...resultSettings,
                        ...{
                            isDisplay: true,
                            currentResult: inputsType.text
                        }
                });

                getSavedResults()
            }

            getStaticResultArray(onSuccess)
        },
    }

    return {
        inputs,
        files,
        resultSettings,
        handleForm,
        handleResult
    };
};

export default useForm;
```

Лістинг Б.2

Компонент графічного інтерфейсу – «actions»

```
// api.js file
import axios from "axios";

import { setTokenToHeader } from "../utils/handleToken";

const baseUrl = "https://localhost:44335/"

export default {
    Comparison(url = baseUrl + "Comparison/") {
        return {
            getStaticResultArray: () => axios.get(url, {
                headers: setTokenToHeader()
            }),
            gerComparisonByString: (inputs) => axios.post(url + "ComparisonByString", inputs, {
                headers: setTokenToHeader()
            }),
            gerComparisonByFiles: (files) => axios.post(url + "ComparisonByFiles", files, {
                headers: setTokenToHeader(),
            }),
            getSavedResults: () => axios.get(url + "GetSavedResults", {
                headers: setTokenToHeader()
            }),
            getSavedResultJson: (id) => axios.get(url + "GetResultList", {
                headers: setTokenToHeader(),
                params: { id: id }
            }),
            getSavedResultTxt: (id) => axios.get(url + "GetResultString", {
                headers: setTokenToHeader(),
                params: { id: id }
            }),
```

```
        }
    },

    User(url = baseUrl) {
        return {
            signin: (inputs) => axios.get(url + "api/Users/Authentication", {
                params: {
                    email: inputs.email,
                    password: inputs.password
                }
            }),
            interimSignup: (inputs) => axios.post(url + `api/Users/InterimSignup`, null, {
                params: {
                    email: inputs.email,
                    password: inputs.password
                }
            }),
            signup: (inputs) => axios.post(url + "api/Users/Signup", null, {
                params: {
                    email: inputs.email,
                    verCode: inputs.verCode
                }
            }),
            email: (email) => axios.get(url + "api/Users/SendCodeToEmail", { params: { email } }),
            verCode: (inputs) => axios.get(url + "api/Users/CheckVerCode", {
                params: {
                    email: inputs.email,
                    verCode: inputs.verCode
                }
            }),
            changePassword: (inputs) => axios.put(url + "api/Users/ChangePassword", null, {
                params: {
                    email: inputs.email,
                    newPassword: inputs.password
                }
            }),
        }
    }
}

// store.js file
import { createStore, applyMiddleware, compose} from "redux"
import thunk from "redux-thunk"
import { redusers } from "../reducers"

export const store = createStore(
    redusers,
    compose(
        applyMiddleware(thunk),
    )

// Comparison.js file
import api from "./api"

export const ACTIONS_TYPES = {
    getStaticResultArray: "GET-STATIC-RESULT-ARRAY",
    getResultByString: "GET-RESULT-BY-STRING",
    getResultByFiles: "GET-RESULT-BY-FILES",
    getResultString: "GET-RESULT-STRING",
    getSavedResults: "GET-SAVED-RESULTS",
}

//action creater
```

```
export const getStaticResultArray = (onSuccess) => dispatch => {
   api.Comparison().getStaticResultArray()
   .then(response => {
      console.log(response)
      dispatch({
         type: ACTIONS_TYPES.getStaticResultArray,
         payload: response.data
      })
      if (onSuccess) onSuccess()
   })
   .catch(error => console.log(error))
}

export const getResultByString = (inputs, onSuccess) => dispatch => {
   api.Comparison().gerComparisonByString(inputs)
   .then(response => {
      console.log(response)
      dispatch({
         type: ACTIONS_TYPES.getResultByString,
         payload: response.data
      })
      if (onSuccess) onSuccess()
   })
   .catch(error => console.log(error))
}

export const getResultByFiles = (files, onSuccess) => dispatch => {
   api.Comparison().gerComparisonByFiles(files)
   .then(response => {
      console.log(response)
      dispatch({
         type: ACTIONS_TYPES.getResultByFiles,
         payload: response.data
      })
      if (onSuccess) onSuccess()
   })
   .catch(error => console.log(error))
}

export const getSavedResults = (onSuccess) => dispatch => {
   api.Comparison().getSavedResults()
   .then(response => {
      console.log(response)
      dispatch({
         type: ACTIONS_TYPES.getSavedResults,
         payload: response.data
      })
      if (onSuccess) onSuccess()
   })
   .catch(error => console.log(error))
}

export const getSavedResultJson = (id, onSuccess) => dispatch => {
   api.Comparison().getSavedResultJson(id)
   .then(response => {
      console.log(response)
      dispatch({
         type: ACTIONS_TYPES.getResultByString,
         payload: response.data
      })
      if (onSuccess) onSuccess(response.data)
   })
   .catch(error => console.log(error))
```

```
}

export const getSavedResultTxt = (id, onSuccess) => {
    api.Comparison().getSavedResultTxt(id)
    .then(response => {
        console.log(response)
        if (onSuccess) onSuccess(response.data)
    })
    .catch(error => console.log(error))
}

// User.js file
import api from "./api"

export const ACTIONS_TYPES = {
    initAuth: "INIT-AUTH",
    signin: "SIGN-IN",
    signup: "SIGN-UP",
}

export const initAuth = (isValidToken) => dispatch => {
    dispatch({
        type: ACTIONS_TYPES.signin,
        payload: { isUserAuth: isValidToken }
    })
}

export const signin = (inputs, onSuccess, onError) => dispatch => {
    api.User().signin(inputs)
    .then(response => {
        console.log(response)

        sessionStorage.setItem("token", response.headers["token"]);
        sessionStorage.setItem("token-expire-time", response.headers["token-expire-time"]);

        dispatch({
            type: ACTIONS_TYPES.signin,
            payload: { isUserAuth: true }
        })

        if (onSuccess) onSuccess()
    })
    .catch(error => {
        console.log(error.response)

        if (onError) onError(error.response.status, error.response.data)
    })
}

export const interimSignup = (inputs, onSuccess, onError) =>  {
    api.User().interimSignup(inputs)
    .then(response => {
        console.log(response)

        if (onSuccess) onSuccess()
    })
    .catch(error => {
        console.log(error.response)

        if (onError) onError(error.response.status, error.response.data)
    })
}
```

```
export const signup = (inputs, onSuccess, onError) => dispatch => {
    api.User().signup(inputs)
    .then(response => {
        console.log(response)

        dispatch({
            type: ACTIONS_TYPES.signup,
            payload: { isUserAuth: true }
        })

        sessionStorage.setItem("token", response.headers["token"]);
        sessionStorage.setItem("token-expire-time", response.headers["token-expire-time"]);

        if (onSuccess) onSuccess()
    })
    .catch(error => {
        console.log(error.response)

        if (onError) onError(error.response.status, error.response.data)
    })
}

export const email = (email, onSuccess, onError) =>  {
    api.User().email(email)
    .then(response => {
        console.log(response)

        if (onSuccess) onSuccess()
    })
    .catch(error => {
        console.log(error.response)

        if (onError) onError(error.response.status, error.response.data)
    })
}

export const verCode = (inputs, onSuccess, onError) => {
    api.User().verCode(inputs)
    .then(response => {
        console.log(response)

        if (onSuccess) onSuccess()
    })
    .catch(error => {
        console.log(error.response)

        if (onError) onError(error.response.status, error.response.data)
    })
}

export const changePassword = (inputs, onSuccess, onError) => {
    api.User().changePassword(inputs)
    .then(response => {
        console.log(response)

        if (onSuccess) onSuccess()
    })
    .catch(error => {
        console.log(error.response)

        if (onError) onError(error.response.status, error.response.data)
    })
}
```

Лістинг Б.3

Компонент графічного інтерфейсу – «reducers»

```
//index.js file
import { combineReducers } from "redux"
import { ComparisonReducer } from "./Comparison"
import { UserReducer } from "./User"

export const redusers = combineReducers({
   ComparisonReducer,
   UserReducer
})

import { ACTIONS_TYPES } from "../actions/Comparison"

const initialState = {
   result: [],
   resultString: "",
   savedResultsList: [],
}

//Comparison.js file
export const ComparisonReducer = (state = initialState, action) => {
   switch(action.type) {
      case ACTIONS_TYPES.getStaticResultArray:
         return {
            ...state,
            result: [...action.payload],
         }
      case ACTIONS_TYPES.getResultByString:
         return {
            ...state,
            result: [...action.payload],
         }
      case ACTIONS_TYPES.getResultByFiles:
         return {
            ...state,
            result: [...action.payload],
         }
      case ACTIONS_TYPES.getResultString:
         return {
            ...state,
            resultString: action.payload,
         }
      case ACTIONS_TYPES.getSavedResults:
         return {
            ...state,
            savedResultsList: [...action.payload],
         }
      default:
         return state;
   }
}

//User.js file
import { ACTIONS_TYPES } from "../actions/User"

const initialState = {
   isUserAuth: false,
}

export const UserReducer = (state = initialState, action) => {
```

```
    switch(action.type) {
        case ACTIONS_TYPES.initAuth:
            return {
                ...state,
                ...action.payload,
            }
        case ACTIONS_TYPES.signin:
            return {
                ...state,
                ...action.payload,
            }
        case ACTIONS_TYPES.signup:
        return {
            ...state,
            ...action.payload,
        }
        default:
            return state;
    }
}
```

Лістинг Б.4

Клас серверної частини – «ComparisonController»

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Http;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

using back_end.Services;
using back_end.Models;
using back_end.Core;
using back_end.Models.Database;
using back_end.Models.Database.Helper;

namespace back_end.Controllers
{
        [ApiController]
        [Route("[controller]")]
        public class ComparisonController : Controller
        {
          ComparisonService _comparisonService;
          private readonly DiplomaDBContext _context;

          public ComparisonController(ComparisonService comparisonService, DiplomaDBContext context)
          {
                  _comparisonService = comparisonService;
                  _context = context;
          }

          [HttpGet]
          public async Task<IEnumerable<ResultItem>> GetStaticResult()
          {
                  List<ResultItem> resultList = _comparisonService.GetResultList();

                  if (Request.Headers.ContainsKey("Authorization"))
                  {
                          HandleQueries handleQueries = new HandleQueries(_context);
                          Models.Database.User user = handleQueries.GetUser(Request.Headers);
```

```
                if (user != null && Helper.HandleAuth.CheckValidToken(_context, user))
                {
                        string newToken = Helper.HandleAuth.UpdateToken(_context, user);

                        if (newToken != Helper.HandleAuth.TryGetToken(Request.Headers))
                        {
                                Response.Headers.Add("token", newToken);
                                Response.Headers.Add("token-expire-time",
DateTime.Now.AddMinutes(Models.TokenOptions.LIFETIME).ToString());
                        }

                        handleQueries.SetResult(user, _comparisonService.GetResultString(), resultList);
                }

        }

        return resultList;
  }

  [HttpPost("ComparisonByString")]
  public IEnumerable<ResultItem> GetComparisonByString(InputString input)
  {
        //if (input.source == null || input.target == null) return BadRequest();

        List<ResultItem> resultList = _comparisonService.GetResultList(input.source, input.target);

        if (Request.Headers.ContainsKey("Authorization"))
        {
                HandleQueries handleQueries = new HandleQueries(_context);
                Models.Database.User user = handleQueries.GetUser(Request.Headers);

                if (user != null && Helper.HandleAuth.CheckValidToken(_context, user))
                {
                        string newToken = Helper.HandleAuth.UpdateToken(_context, user);

                        if (newToken != Helper.HandleAuth.TryGetToken(Request.Headers))
                        {
                                Response.Headers.Add("token", newToken);
                                Response.Headers.Add("token-expire-time",
DateTime.Now.AddMinutes(Models.TokenOptions.LIFETIME).ToString());
                        }

                        handleQueries.SetResult(user, _comparisonService.GetResultString(), resultList);
                }
        }

        return resultList;
  }

        [HttpPost("ComparisonByFiles")]
        public IEnumerable<ResultItem> GetComparisonByFiles([FromForm] InputFiles files)
        {
                string source = HandleFile.FormFileToString(files.FormFiles[0]);
                string target = HandleFile.FormFileToString(files.FormFiles[1]);

                List<ResultItem> resultList = _comparisonService.GetResultList(source, target);

                if (Request.Headers.ContainsKey("Authorization"))
                {
                        HandleQueries handleQueries = new HandleQueries(_context);
                        Models.Database.User user = handleQueries.GetUser(Request.Headers);
```

```
                        if (user != null && Helper.HandleAuth.CheckValidToken(_context, user))
                        {
                                string newToken = Helper.HandleAuth.UpdateToken(_context, user);

                                if (newToken != Helper.HandleAuth.TryGetToken(Request.Headers))
                                {
                                        Response.Headers.Add("token", newToken);
                                        Response.Headers.Add("token-expire-time",
DateTime.Now.AddMinutes(Models.TokenOptions.LIFETIME).ToString());
                                }

                                handleQueries.SetResult(user, _comparisonService.GetResultString(), resultList);
                        }
                }

                return resultList;
        }

        // Get info about user from header with token
        [HttpGet("GetSavedResults")]
        public IEnumerable<SavedResult> GetSavedResults()
        {
                if (Request.Headers.ContainsKey("Authorization"))
                {
                        HandleQueries handleQueries = new HandleQueries(_context);
                        Models.Database.User user = handleQueries.GetUser(Request.Headers);

                        if (user != null && Helper.HandleAuth.CheckValidToken(_context, user))
                        {
                                List<Models.Database.Result>                    results                    =
_context.Result.FromSqlRaw(DBQueries.GetResults(user)).ToList();
                                return Models.Helper.Maping.DatabaseResultToSavedResult(results);
                        }
                }

                return null;
        }

        [HttpGet("GetResultList")]
        public IEnumerable<ResultItem> GetResult(int id)
        {
                Models.Database.Result result = _context.Result.Find(id);

                if (result == null) return null;

                return Core.HandleFile.filepathToList(result.ResultList);
        }

        [HttpGet("GetResultString")]
        public string GetResultString(int id)
        {
                Models.Database.Result result = _context.Result.Find(id);

                if (result == null) return null;

                return Core.HandleFile.filepathToString(result.ResultFile);
        }
    }
}
```

Лістинг Б.5

Клас серверної частини – «UsersController»

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

using System.Security.Claims;
using System.IdentityModel.Tokens.Jwt;
using Microsoft.IdentityModel.Tokens;

using back_end.Models.Database;
using back_end.Models.Database.Helper;
using back_end.Controllers.Helper;
using back_end.Models;

namespace back_end.Controllers
{
   [Route("api/[controller]")]
   [ApiController]
   public class UsersController : ControllerBase
   {
      private readonly DiplomaDBContext _context;

      public UsersController(DiplomaDBContext context)
      {
         _context = context;
      }

      // GET: api/Users
      [HttpGet]
      public async Task<ActionResult<IEnumerable<User>>> GetUser()
      {
         return await _context.User.ToListAsync();
      }

      // GET: api/Users/5
      [HttpGet("{id}")]
      public async Task<ActionResult<User>> GetUser(int id)
      {
         var user = await _context.User.FindAsync(id);

         if (user == null)
         {
            return NotFound();
         }

         return user;
      }

      [HttpGet("Authentication")]
      public async Task<ActionResult<User>> Authentication(string email, string password)
      {
         List<User> user = await _context.User.FromSqlRaw(DBQueries.AuthenticationQuery(email,
password)).ToListAsync();

         if (user != null && user.Count != 0)
```

```
        {
            string encodedJwt = HandleAuth.CreateToken(user[0]);

            HandleQueries handleQueries = new HandleQueries(_context);
            handleQueries.SetToken(user[0].UserId, encodedJwt);

            Response.Headers.Add("token", encodedJwt);
            Response.Headers.Add("token-expire-time",
DateTime.Now.AddMinutes(Models.TokenOptions.LIFETIME).ToString());

            return Ok(new {
                username = user[0].Email
            });
        }
        else
        {
            return NotFound("Invalid username or password.");
        }
    }

    [HttpPost("Signup")]
    public async Task<ActionResult<User>> Signup(string email, int verCode)
    {
        bool findUser = _context.User.Any(u => u.Email == email);

        if (!_context.InterimSignup.Any(u => u.VerCode == verCode)) return BadRequest("Code is not correct");
        if (!_context.InterimSignup.Any(u => u.VerCode == verCode && u.ExpireTime > DateTime.Now)) return
BadRequest("Code is expire");

        if (!findUser)
                    {
            HandleQueries handleQueries = new HandleQueries(_context);
            InterimSignup interimSignup = _context.InterimSignup.SingleOrDefault(u => u.Email == email &&
                                                    u.VerCode == verCode &&
                                                    u.ExpireTime > DateTime.Now);
            if (interimSignup != null)
            {
                User newUser = new User
                {
                    Email = interimSignup.Email,
                    Password = interimSignup.Password,
                    Type = "user",
                };

                _context.User.Add(newUser);
                await _context.SaveChangesAsync();

                handleQueries.DeleteInterimSignupItems(newUser.Email);

                string encodedJwt = HandleAuth.CreateToken(newUser);
                var user = await _context.User.FromSqlRaw(DBQueries.AuthenticationQuery(newUser.Email,
newUser.Password)).ToListAsync();
                handleQueries.SetToken(user[0].UserId, encodedJwt);

                Response.Headers.Add("token", encodedJwt);
                Response.Headers.Add("token-expire-time",
DateTime.Now.AddMinutes(Models.TokenOptions.LIFETIME).ToString());

                return Ok(new
                {
                    username = user[0].Email
                });
            }
```

```
            return BadRequest();
        }

        return BadRequest($"User with email: '{email}' already exist");
    }

[HttpPost("InterimSignup")]
    public async Task<ActionResult<User>> InterimSignup(string email, string password)
    {
        HandleQueries handleQueries = new HandleQueries(_context);

        if (handleQueries.ExistEmail(email)) return BadRequest($"User with email: '{email}' already exist");
        else
        {
            InterimSignup interimSignup = new InterimSignup
            {
                VerCode = HandleEmail.CreateVerCode(),
                Email = email,
                Password = password,
                ExpireTime = DateTime.Now.AddMinutes(10)
            };

            if (HandleEmail.SendEmail(email, interimSignup.VerCode.ToString()))
            {
                _context.InterimSignup.Add(interimSignup);
                await _context.SaveChangesAsync();
            }
            else return BadRequest("Send email failed");

            return Ok();
        }
    }

    [HttpGet("SendCodeToEmail")]
    public async Task<ActionResult<User>> SendCodeToEmail(string email)
    {
        int verCode = HandleEmail.CreateVerCode();
        if (HandleEmail.SendEmail(email, verCode.ToString()))
        {
            if (_context.User.Any(u => u.Email == email))
                                {
                _context.InterimSignup.Add(new InterimSignup {
                    Email = email,
                    Password = "-",
                    VerCode = verCode,
                    ExpireTime = DateTime.Now.AddMinutes(5),
                });

                _context.SaveChanges();
            }

            return Ok();
        }
        else return BadRequest("Send email failed");
    }

    [HttpGet("CheckVerCode")]
    public async Task<ActionResult<User>> CheckVerCode(string email, int verCode)
    {

        if (_context.InterimSignup.Any(u => u.Email == email && u.VerCode == verCode && u.ExpireTime >
DateTime.Now))
```

```
    {
        HandleQueries handleQueries = new HandleQueries(_context);
        handleQueries.DeleteInterimSignupItems(email);

        return Ok();
    }

    return BadRequest("Verification code expired");
}

[HttpPut("ChangePassword")]
public async Task<ActionResult<User>> ChangePassword(string email, string newPassword)
{
    User user = _context.User.SingleOrDefault(u => u.Email == email);

    if (user != null)
    {
        user.Password = newPassword;
        _context.SaveChanges();

        return Ok();
    }

    return NotFound();
}

// PUT: api/Users/5
// To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
[HttpPut("{id}")]
public async Task<IActionResult> PutUser(int id, User user)
{
    if (id != user.UserId)
    {
        return BadRequest();
    }

    _context.Entry(user).State = EntityState.Modified;

    try
    {
        await _context.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!UserExists(id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }

    return NoContent();
}

// POST: api/Users
// To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
[HttpPost]
public async Task<ActionResult<User>> PostUser(User user)
{
    _context.User.Add(user);
```

```
            await _context.SaveChangesAsync();

            return CreatedAtAction("GetUser", new { id = user.UserId }, user);
        }

        // DELETE: api/Users/5
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteUser(int id)
        {
            var user = await _context.User.FindAsync(id);
            if (user == null)
            {
                return NotFound();
            }

            _context.User.Remove(user);
            await _context.SaveChangesAsync();

            return NoContent();
        }

        private bool UserExists(int id)
        {
            return _context.User.Any(e => e.UserId == id);
        }
    }
}
```

Лістинг Б.6

Клас серверної частини – «Comparison»

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;

using back_end.Models;
using back_end.Core;

namespace back_end.Core
{
        public class Comparison
        {
                private Result result;
        public ComparisonItem source { get; set; }
        public ComparisonItem target { get; set; }

        public int startMatrixIndex { get; set; }
        public int currentMatrixIndex { get; set; }

        public Comparison()
                {
            source = new ComparisonItem(HandleFile.filepathToString("./defaultFiles/source.txt"), NodeType.delete, new string[] { "[", "]" });
            target = new ComparisonItem(HandleFile.filepathToString("./defaultFiles/target.txt"), NodeType.insert, new string[] { "(", ")" });
            result = new Result();
        }

        public Comparison(string source, string target)
                {
```

```
        this.source = new ComparisonItem(source, NodeType.delete, new string[] { "[", "]" });
        this.target = new ComparisonItem(target, NodeType.insert, new string[] { "(", ")" });
        result = new Result();
    }

    public Result Compare()
                {
        startMatrixIndex = createMatrix(source.src, target.src);

        while (source.checkRange() && target.checkRange()) // виконується поки що обидва рядки не пройдені
        {
            if (source.src[source.position] == target.src[target.position])
            {
                result.SetResult(source.src[source.position]);

                source.position++;
                target.position++;
            }
            else
            {
                startMatrixIndex = createMatrix(source.src.Substring(source.position),
target.src.Substring(target.position)); // обчислення поточного стану матриць ставлення (з огляду на поточні
позиції у рядках)
                alignMatrix(source, target);
                alignMatrix(target, source);
            }
        }

        // якщо виявилося, що один рядок коротший за інший спрацюють дані перевірки
        alignMatrix(source);
        alignMatrix(target);

        return result;
    }

    private int createMatrix(string source, string target)
    {
        int row = source.Length + 1;
        int col = target.Length + 1;
        int[,] matrix = new int[row, col];

        for (int i = 0; i < row; i++) matrix[i, 0] = 0;
        for (int j = 0; j < col; j++) matrix[0, j] = 0;

        for (int i = 1; i < row; i++)
        {
            for (int j = 1; j < col; j++)
            {
                if (source[i - 1] == target[j - 1]) matrix[i, j] = matrix[i - 1, j - 1] + 1;
                else
                {
                    if (matrix[i, j - 1] > matrix[i - 1, j]) matrix[i, j] = matrix[i, j - 1];
                    else matrix[i, j] = matrix[i - 1, j];
                }
            }
        }

        return matrix[row - 1, col - 1];
    }

    private void alignMatrix(ComparisonItem itemLeft, ComparisonItem itemRight)
    {
        currentMatrixIndex = createMatrix(source.src.Substring(source.position), target.src.Substring(target.position));
```

```
        if (itemLeft.position < itemLeft.src.Length - 1
            && startMatrixIndex == createMatrix(itemLeft.src.Substring(itemLeft.position + 1),
itemRight.src.Substring(itemRight.position))) // обчислення нової матриці ставлення з кроком  - positionSource + 1
        {
            // якщо в результаті кроку кількість однакових символів не змінилася - це означає, що дія обрана правильно
            string substring = String.Empty;

            while (startMatrixIndex == currentMatrixIndex && itemLeft.position < itemLeft.src.Length) // якщо потрібна дія виконується поспіль
            {
                substring += itemLeft.src[itemLeft.position++];

                if (itemLeft.position + 1 < itemLeft.src.Length)
                    currentMatrixIndex = createMatrix(itemLeft.src.Substring(itemLeft.position + 1),
itemRight.src.Substring(itemRight.position)); // якщо під кінець рядка кожен рядок має відмінні (унікальні) елементи, то спрацює дана ситуація, коли може порушиться діапазон рядка
            }

            result.SetResult(itemLeft, substring, true);
        }
    }

    private void alignMatrix(ComparisonItem item)
            {
        if (item.position < item.src.Length - 1)
            result.SetResult(item, item.src.Substring(item.position));
    }
  }
}
```