**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** *Бескоморного Андрія Ігоровича*
(ПІБ)

**академічної групи** *121-19з-1*
(шифр)

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

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

**на тему:** *Розробка веб-застосунку для тестування школярів з використанням фреймворків .NET і ReactJS*

| **Керівники** | **Прізвище, ініціали** | **Оцінка за шкалою** | | **Підпис** |
|---|---|---|---|---|
| | | **рейтинговою** | **інституційною** | |
| кваліфікаційної роботи | *доц. Кабак Л.В.* | | | |
| **розділів:** | | | | |
| спеціальний | *доц. Кабак Л.В.* | | | |
| економічний | *доц. Касьяненко Л.В.* | | | |
| | | | | |
| | | | | |
| **Рецензент** | *доц. Каштан В.Ю.* | | | |
| **Нормоконтролер** | *ст.викл. Мартиненко А.А.* | | | |

**Дніпро**
**2023**

**Міністерство освіти і науки України**
**НТУ «Дніпровська політехніка»**

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

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2023 року

# ЗАВДАННЯ
## на кваліфікаційну роботу

*бакалавра*
(назва освітньо-кваліфікаційного рівня)

**студента** *121-19з-1* (група) *Бескоморного Андрія Ігоровича* (прізвище та ініціали)

**тема кваліфікаційної роботи** *Розробка веб-застосунку для тестування школярів з використанням фреймворків .NET і ReactJS*

затверджена наказом ректора НТУ «ДП» від *11.04.2023* № *256-с*

| Розділ | Зміст виконання | Термін виконання |
|---|---|---|
| ***Спеціальний*** | ***На основі матеріалів проєктно-технологічної та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми*** | ***13.05.2023 р.*** |
| ***Економічний*** | ***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки*** | ***27.05.2023 р.*** |

Завдання видав _____ *доц. Кабак Л.В.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Бескоморний А.І.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2023 р.*

Термін подання кваліфікаційної роботи до ЕК: *12.06.2023 р.*

# РЕФЕРАТ

Пояснювальна записка: 88 с., 17 рис., 3 дод., 17 джерел.

Об'єкт розробки: веб платформа для розміщення і проходження онлайн тестів для школярів.

Мета кваліфікаційної роботи: забезпечити вчителів можливістю створювати онлайн тести, які можна проходити учнями в зручний час.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проведено аналіз предметної області, визначено актуальність завдання та призначення розробки, розроблена постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі виконано аналіз існуючих рішень, обрано вибір платформи для розробки, виконано проектування і розробка програми, наведено опис алгоритму і структури функціонування системи, визначені вхідні і вихідні дані, наведені характеристики складу параметрів технічних засобів, описаний виклик та завантаження застосунку, описана робота програми.

В економічному розділі визначено трудомісткість розробленої інформаційної підсистеми, проведений підрахунок вартості роботи по створенню застосунку та розраховано час на його створення.

Актуальність даного програмного забезпечення визначається необхідністю шкіл мати онлайн інструмент для створення і проходження тестів.

Список ключових слів: API, UI, JWT, CRUD, NoSQL, REST, ACID.

## ABSTRACT

Explanatory note: 88 pages, 17 pics, 3 apps, 17 sources.

Object of development: a web platform for posting and taking online tests for schoolchildren.

The purpose of the qualification work: to provide teachers with the ability to create online tests that can be taken by students at a convenient time.

The introduction discusses the analysis and current state of the problem, specifies the purpose of the qualification work and its scope, provides a justification for the relevance of the topic, and clarifies the task statement.

The first chapter analyzes the subject area, determines the relevance of the task and the purpose of the development, develops the task statement, and sets the requirements for software implementation, technologies and software tools.

The second section analyzes existing solutions, selects a platform for development, designs and develops the program, describes the algorithm and structure of the system, defines input and output data, describes the composition of the parameters of technical means, describes the call and download of the application, and describes the operation of the program.

The economic section defines the labor intensity of the developed information subsystem, calculates the cost of creating the application, and estimates the time for its creation.

The relevance of this software is determined by the need for schools to have an online tool for creating and passing tests.

List of keywords: API, UI, JWT, CRUD, NoSQL, REST, ACID.

**ЗМІСТ**

## СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

API — Application Programming Interface (інтерфейс програмування додатків).

UI — User Interface (інтерфейс користувача).

ПЗ — програмне забезпечення.

JWT — JSON Web Token (JSON веб токен).

CRUD — Create-Read-Update-Delete (створити-прочитати-оновити-видалити).

CLI — Command Line Interface (інтерфейс командного рядка).

SQL — Structured query language (мова структурованих запитів).

NoSQL — Not Only SQL (не тільки SQL).

HTTP — HyperText Transfer Protocol (протокол передачі гіпертексту).

REST — Representational State Transfer (передача репрезентативного стану).

JSON — JavaScript Object Notation (запис об’єктів JavaScript).

BSON — Binary JavaScript Object Notation (бінарний запис об’єктів JavaScript).

XML — Extensible Markup Language (розширювана мова розмітки).

ACID — Atomicity-Consistency-Isolation-Durability (атомарність-узгодженість-ізольованість-довговічність).

# ВСТУП

Онлайн-тести - це форма оцінювання, яка проводиться за допомогою комп'ютера або мобільного пристрою та підключення до Інтернету. Ці тести використовуються в електронному навчанні для оцінки знань і розуміння різних предметів учнями. Онлайн-тести можуть бути у формі запитань з декількома варіантами відповідей, запитань на правильну чи неправильну відповідь, запитань з короткою відповіддю або запитань на відповідність. Вони часто використовуються школами та університетами для оцінки прогресу навчання своїх студентів, а також для підтвердження їхніх навичок і знань. Онлайн-іспити проводяться за допомогою різноманітного програмного забезпечення та платформ, які дозволяють створювати, доставляти та аналізувати тести.

Онлайн-іспити та офлайн-іспити мають свої переваги та недоліки. Онлайн-іспити пропонують більш гнучкий спосіб проведення іспитів, оскільки їх можна складати з будь-якого місця, де є доступ до Інтернету, і вони можуть оцінюватися автоматично, що економить час. На противагу цьому, недоліками онлайн-іспитів є більша ймовірність списування, можливі технічні проблеми у студентів.

Сьогодні онлайн-іспити стали поширеною формою оцінювання в електронному навчанні, пропонуючи більшу гнучкість і негайні результати як для студентів, так і для викладачів. Хоча онлайн-іспити мають певні обмеження, вони зарекомендували себе як цінний інструмент для оцінювання знань студентів у наш час.

# РОЗДІЛ 1
# АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

## 1.1. Загальні відомості з предметної галузі

Онлайн-платформи для складання іспитів останнім часом набули величезної популярності завдяки своїй зручності та економічній ефективності.
З початком пандемії COVID-19 офлайн-іспити та заняття стали неможливими, що призвело до збільшення кількості людей, які складають іспити онлайн.

Система онлайн-іспитів має низку переваг, серед яких легке створення іспитів, сумісність з мобільними пристроями та зручний статистичний аналіз даних іспитів. Такі платформи також допомагають автоматизувати всі процеси, необхідні для проведення іспитів, що полегшує навчальним закладам управління екзаменаційним процесом. Загалом, система онлайн-іспитів забезпечує надійний, ефективний та зручний метод проведення іспитів, що робить її популярним вибором у сфері освіти.

Найбільш популярними платформами для проведення тестування онлайн є Moodle, Google Forms і Microsoft Forms.

**Moodle** - це популярна система управління навчанням (рис. 1.1), яку можна використовувати для проведення онлайн-іспитів. Вона має низку переваг та недоліків, які наведені нижче:
Переваги:

- Moodle дозволяє легко створювати іспити та керувати ними.
- Платформа сумісна з мобільними пристроями, що дозволяє студентам легко складати іспити з будь-якого місця.
- Можливий статистичний аналіз даних іспитів, що може допомогти у визначенні сфер, які потребують покращення
- Функції боротьби з шахрайством можуть бути включені для забезпечення чесності іспитів

– Система автоматизує завдання, що полегшує навчальним закладам та роботодавцям управління іспитами.
– Безкоштовність використання, відкритий код
– Недоліки:
– Moodle може бути не оптимізована під серверне залізо конкретного хостера, що може призвести до низької швидкості роботи з великою кількістю користувачів.
– Кастомізація вимагає знань з програмування.
– Можуть виникати технічні проблеми, які можуть вплинути на процес складання іспитів.
– Це селф-хостед платформа, тобто потрібно самому встановлювати і налаштовувати все на своєму обладнанні

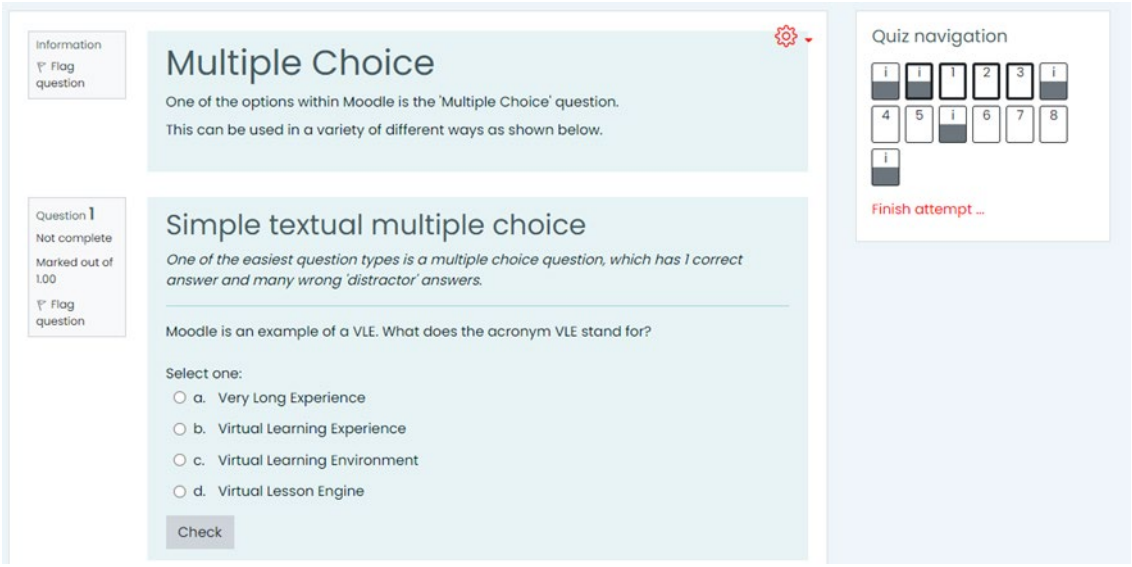


Рис. 1.1. Скріншот платформи Moodle

Загалом, Moodle може бути надійною та ефективною платформою для проведення онлайн-іспитів, але важливо враховувати її обмеження та придатність для конкретного формату іспиту.

**Google Forms** - це безкоштовний онлайн-інструмент, який пропонує різноманітні функції для створення форм, опитувань та вікторин, які можна використовувати для проведення онлайн-іспитів у школах (рис. 1.2). Інструмент надає заздалегідь розроблені шаблони для користувачів і підтримує різні типи запитань, такі як вибір однієї відповіді, вибір декількох відповідей, відповідь текстом і функція завантаження файлів. Дані збираються в електронній таблиці Google, яку можна аналізувати, а для запитань із закритими відповідями пропонується візуальне представлення. Гугл форми також прості у використанні та доступні. Однак існує кілька обмежень для проведення онлайн-іспитів за допомогою гугл форм, таких як відсутність часових обмежень.

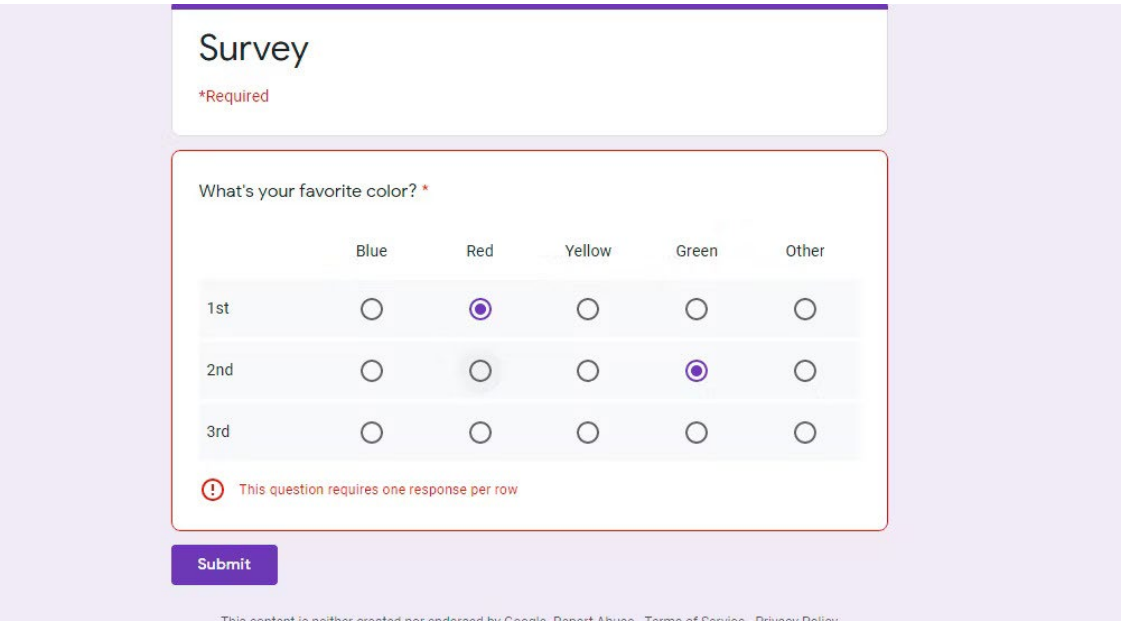


Рис. 1.2. Інтерфейс Google Forms

## 1.2. Призначення розробки та область застосування.

В якості мови програмування обрані C# та Typescript. Головною метою роботи є створення веб-платформи для створення і проходження онлайн тестів.

Головними критеріями розроблюваної системи є:

- Зручність в використанні
- Адаптивний UI
- Відмовостійка серверна частина

Ця веб-платформа призначена для застосування школами і може бути інтегрована з будь-якою системою е-навчання.

## 1.3. Підстава для розробки

Підставами для розробки (виконання кваліфікаційної роботи) є:

- Освітня програма за спеціальністю 121 «Інженерія програмного забезпечення».
- Графік навчального процесу та навчальний план.
- Наказ ректора Національного технічного університету «Дніпровська політехніка» № 256-с від 11.04.2023 р.
- Завдання на кваліфікаційну роботу на тему «Розробка веб-застосунку для тестування школярів з використанням фреймворків .NET і ReactJS».

## 1.4. Постановка завдання

Метою кваліфікаційної роботи є розробка веб додатку для створення і проходження онлайн тестів. Додаток призначений для використання в школах і його легко інтегрувати в будь-яку систему онлайн навчання.

Для виконання проекту потрібно:

- Проаналізувати існуючу рішення.
- Спроектувати архітектуру.
- Розробити дизайн.
- Реалізувати спроектований додаток.

## 1.5. Вимоги до програми або програмного виробу.

### 1.5.1. Вимоги до функціональних характеристик.

Розроблена платформа повинна мати наступні функції:

- Вхід і реєстрація за допомогою Firebase

- Створення, редагування, видалення тестів
- Перегляд доступних для проходження тестів
- Проходження учнями тестів
- Перегляд і редагування оцінок

### 1.5.2. Вимоги до інформаційної безпеки.

Для безпечної роботи платформи потрібно реалізувати:

- Аутентифікацію та реєстрацію за допомогою Firebase
- Перевірку JWT токенів на бекенді
- Перевірку ролей для доступу до ресурсів
- Перевіряти чи робота була здана в дозволеному проміжку часу та чи не використаний весь час сесії
- Збереження JWT токенів на фронтенді в sessionStorage

### 1.5.3. Вимоги до складу та параметрів технічних засобів.

Для роботи з клієнтом потрібне лише постійне підключення до Інтернету та сучасний браузер, який підтримує сучасний JavaScript.

Незважаючи на те, що UI цього проекту повністю адаптивний, потрібно, щоб розмір екрану був 370×420px або більше.

Для підтримки стабільної роботи серверної частини потрібен кластер Kubernetes з декількох віртуальних серверів.

### 1.5.4. Вимоги до інформаційної та програмної сумісності.

Для запуску серверної частини потрібно наступне ПЗ:

- PostgreSQL 14 або вище
- .NET 7

Для запуску клієнтської частини потрібен будь-який сучасний браузер, роботу застосунку перевірено в наступних браузерах:

- Chrome 114
- Firefox 114
- Safari 16.4.1

# РОЗДІЛ 2
# ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

## 2.1. Функціональне призначення системи

Під час виконання кваліфікаційної роботи було розроблено веб додаток, головна мета якого забезпечити школи можливістю проведення онлайн тестування.

## 2.2. Опис застосованих математичних методів

У серверній частині було застосовано алгоритми вихор Мерсенна та відстань Дамерау-Левенштейна. Вихор Мерсенна використовується для генерування випадкового числа, а відстань Дамерау-Левенштейна - для перевірки, чи збігається відповідь із правильним варіантом у разі орфографічних помилок.

**Mersenne Twister (Вихор Мерсенна)** - високоефективний генератор псевдовипадкових чисел, розроблений 1997 року японськими вченими Макото Мацумото і Такудзі Нісімурою. До математика Марену Мерсенна назва має відношення тому, що період генерації дорівнює числу 219937-1, яке, своєю чергою, є числом Мерсенна. Незважаючи на те, що Mersenne Twister є одним із найретельніше протестованих генераторів чисел з тих, що існують зараз, а послідовності, які він видає, проходять статистичні тести, використання Mersenne Twister у криптографії не рекомендується без додаткового шифрування.

Цей генератор вважається одним із найкращих на цей час і має дуже великий період, однак для генерації одного елемента з потоку потрібне виконання складного послідовного алгоритму. Також генератор вимагає багато пам'яті для зберігання свого стану, що може сильно вплинути на продуктивність,

тому що стан доводиться зберігати в глобальній пам'яті. Однак, у ситуаціях, коли точність важливіша за швидкість, краще використовувати цей генератор.

Генератор дещо складний у реалізації, оскільки він орієнтований на word, а не на байти. Оскільки генератор є word-орієнтованим, це має два наслідки для реалізації. По-перше, результат роботи GenerateWord32 має відповідати результату виклику GenerateBlock з масивами з 1, 2, 3 і 4 байт на машинах як з big, так і з little endian. Наприклад, якщо GenerateWord32 повертає 0xD091BB5C, то GenerateBlock має повернути 0xD0 0x91 0xBB 0x5C для 1, 2, 3 і 4-байтових масивів. По-друге, Discard округляє до кратного розміру слова, а потім відкидає необхідну кількість слів (а не байт).

Вихор Мерсенна також потребує параметризації деяких магічних констант. Магічні константи - це K, що використовується у прокручуванні; M, що використовується як параметр періоду; N, що використовується як розмір стану; F, що використовується як множник; і S, що використовується як початкове значення за замовчуванням.

Вихор Мерсенна алгоритмічно реалізується двома основними частинами: рекурсивною і загартування. Рекурсивна частина являє собою регістр зсуву з лінійним зворотним зв'язком, у якому всі біти в його слові визначаються рекурсивно; потік вихідних бітів визначаються також рекурсивно функцією бітів стану.

Регістр зсуву складається з 624 елементів, і, загалом, з 19937 клітин. Кожен елемент має довжину 32 біти за винятком першого елемента, який має тільки 1 біт за рахунок відкидання біта.

Процес генерації починається з логічного множення на бітову маску, що відкидає 31 біт (крім найбільш значущих).

Наступним кроком виконується ініціалізація ($x_0$, $x_1$,..., $x_{623}$) будь-якими беззнаковими 32-розрядними цілими числами. Наступні кроки містять об'єднання та перехідні стани.

Простір станів має 19937 біт (624-32 - 31). Наступний стан генерується зсувом одного слова вертикально вгору і вставкою нового слова в кінець. Нове

слово обчислюється гамуванням середньої частини з виключеною. Вихідна послідовність починається з $x_{624}$, $x_{625}$,....

**Відстань Дамерау-Левенштейна** - це варіант відстані Левенштейна, який є різновидом відстані редагування. Відстань редагування - це великий клас метрик відстані, що вимірює відмінність між двома рядками шляхом обчислення мінімальної кількості операцій (з набору операцій), які використовуються для перетворення одного рядка в інший. Її можна розглядати як спосіб попарного вирівнювання рядків.

Дамерау стверджував, що чотири операції у відстані Дамерау-Левенштейна відповідають більш ніж 80% усіх людських помилок

Відстань Дамерау-Левенштейна відрізняється від класичної відстані Левенштейна тим, що включає операцію транспозиції серед допустимих операцій на додаток до трьох операцій редагування окремих символів (вставки, видалення та заміни).

Відстань Дамерау-Левенштейна має широкий спектр застосувань в обчислювальній лінгвістиці, інформатиці, обробці природної мови, біоінформатиці та багатьох інших галузях.

Для двох рядків S1 і S2, відстань редагування - це мінімальна вартість операцій, пов'язаних з перетворенням рядка S1 в S2. 3 Операції, доступні у відстані Левенштейна, такі:

- Вставка: Вставити символ у певну позицію (вартість: 1)
- Видалення: Видалити символ у будь-якій позиції (вартість: 1)
- Замінити: Замінити символ у будь-якій позиції на інший символ (вартість: 1)
- Додамо ще одну операцію: транспозиція
- Транспозиція: поміняти місцями два сусідні символи (вартість: 1)

Для операції транспозиції ми будемо дивитися на один символ назад, щоб застосувати цю операцію.

Наприклад**:**

*S1: ag qwe*

*S2: a wqe*

Операції на дистанції Левенштейна**:**

*Операція 1: видалити g на позиції 1*

*Операція 2: видалити q на позиції 4*

*Операція 3: вставити q на позиції 5*

*Відстань Левенштейна = 3*

Операції на відстані Левенштейна з операцією транспозиції**:**

*Операція 1: видалення g на позиції 1*

*Операція 2: транспозиція: поміняти місцями символи на позиціях 4 і 5*

*Відстань Левенштейна з операцією транспозиції = 2*

Однією з проблем цього методу є те, що він припускає, що між символами, які переставляються, не додається і не видаляється жоден символ.

Отже, у відстані Дамерау-Левенштейна для операції транспонування ми будемо дивитися за межі одного символу, а отже, ми можемо міняти місцями символи між будь-якими індексами.

Вартість транспозиції обчислюється наступним чином:

(вартість до транспозиції) + (відстань між рядками) + (відстань між стовпчиками) + 1

Обчислюється вартість всіх чотирьох операцій і вибирається операція з мінімальною вартістю.

Відстань Левенштейна з невід'ємною вартістю задовольняє аксіомам метрики, що породжує метричний простір рядків, коли виконуються наступні умови:

- Кожна операція редагування має додатну вартість
- Для кожної операції існує обернена операція з рівною вартістю
    - Властивості відстаней Левенштейна з одиничною вартістю:
- LCS відстань обмежена зверху сумою довжин пари рядків
- LCS відстань є верхньою межею відстані Левенштейна

– Для строчок однакової довжини відстань Хеммінга є верхньою межею відстані Левенштейна

Незалежно від вартості/ваги, для всіх відстаней редагування виконується наступна властивість:

– Коли a та b мають спільний префікс, цей префікс не впливає на відстань

### 2.3. Опис використаних технологій та мов програмування

**Event Sourcing** - це техніка моделювання, за допомогою якої ви не тільки моделюєте стан вашого бізнесу, але й переходи між станами. Тоді, замість того, щоб зберігати поточний стан, сховище даних зберігає ці переходи.

Відмінність від персистентності, заснованої на сховищах даних Create-Read-Update-Delete (CRUD), полягає в тому, що нам не потрібно заздалегідь зіставляти наші уявні сутності з моделлю бази даних. Замість цього ми моделюємо сутності та події, які впливають на їх стан.

Фундаментальна ідея Event Sourcing полягає в тому, що кожна зміна стану програми фіксується в об'єкті події, а самі об'єкти подій зберігаються в тій послідовності, в якій вони були застосовані, протягом того ж часу, що і стан самої програми (рис. 2.1.).

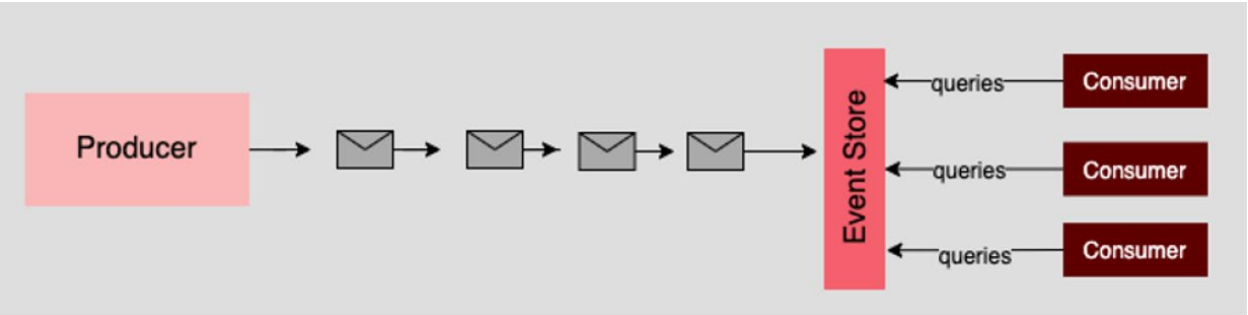


Рис. 2.1. Приклад реалізації Event Sourcing

Розглянемо простий приклад зі сповіщеннями про доставку (рис. 2.2.). У цьому прикладі у нас є багато кораблів у відкритому морі, і нам потрібно знати, де вони знаходяться. Простий спосіб зробити це - мати додаток для відстеження

з методами, які дозволять нам дізнатися, коли корабель прибуває або відпливає з порту.

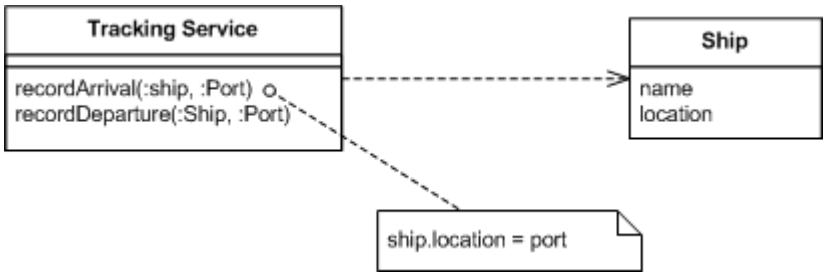


Рис. 2.2. Інтерфейс сервісу і моделі до впровадження Event Sourcing

У цьому випадку, коли сервіс викликається, він знаходить відповідне судно і оновлює його місцезнаходження. Корабельні об'єкти фіксують поточний відомий стан кораблів.

Впровадження Event Sourcing додає ще один крок до цього процесу. Тепер сервіс створює об'єкт події для запису змін і обробляє його, щоб оновити корабель (рис. 2.3.).

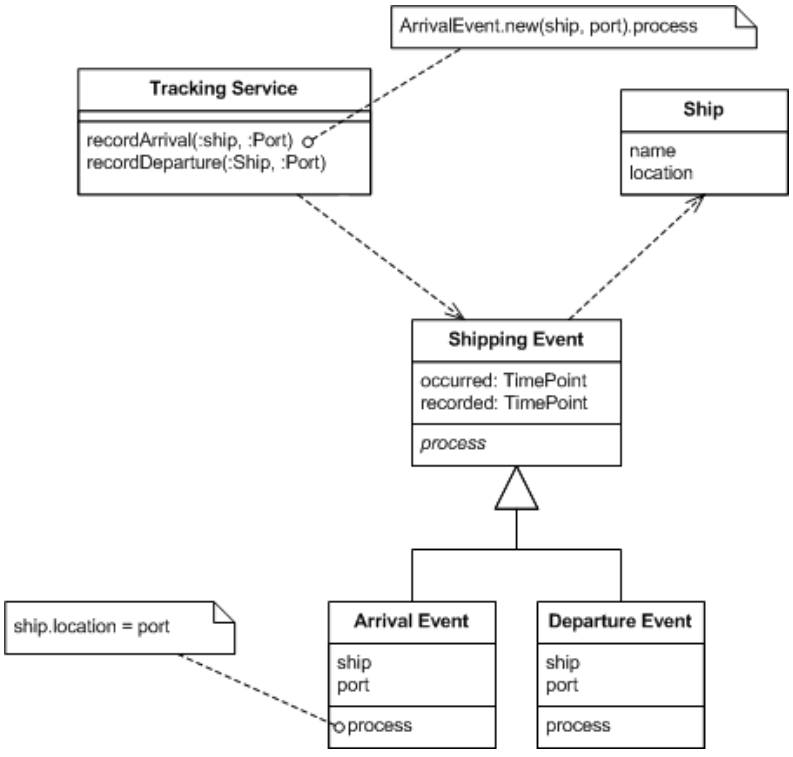


Рис. 2.3. Архітектура коду з Event Sourcing

Якщо розглядати лише обробку, то це просто непотрібний рівень опосередкованості. Цікава різниця виникає, коли ми дивимося на те, що залишається в додатку після кількох змін. Уявімо собі кілька простих змін:

– Корабель "Король Рой" відпливає з Сан-Франциско
– Корабель "Принц Тревор" прибуває до Лос-Анджелеса
– Корабель "Король Рой" прибуває до Гонконгу

За допомогою базового сервісу ми бачимо лише кінцевий стан, зафіксований об'єктами корабля (рис. 2.4.).


Рис. 2.4. Кінцевий стан моделей

За допомогою Event Sourcing ми також фіксуємо кожну подію. Якщо ми використовуємо персистентне сховище, події будуть зберігатися так само, як і об'єкти корабля. В цьому прикладі ми зберігаємо дві різні речі: стан програми та журнал подій (рис. 2.5.).

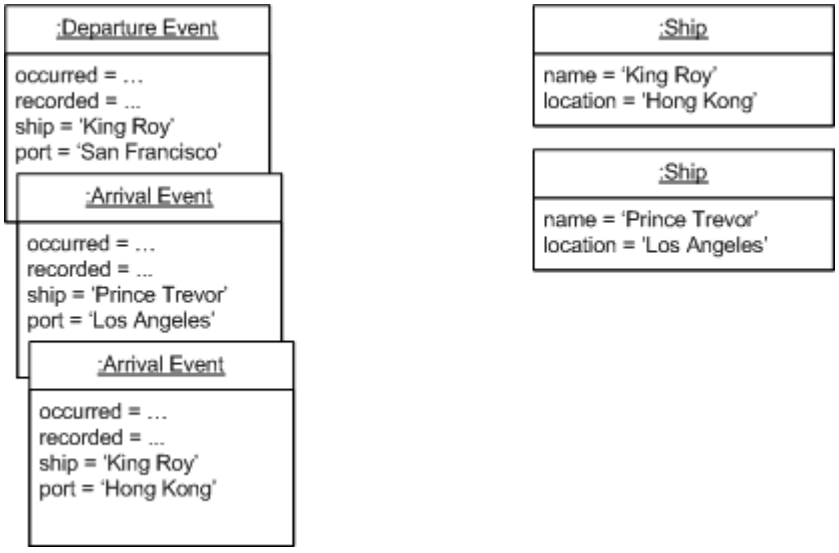


Рис. 2.5. Приклад моделей подій і стану кораблів

Найочевидніша річ, яку ми отримали від використання Event Sourcing - це те, що тепер у нас є журнал усіх змін. Ми не просто бачимо, де знаходиться

кожен корабель, ми бачимо, де він побував. Однак це невеликий виграш. Ми також можемо зробити це, зберігаючи історію минулих портів в об'єкті корабля, або записуючи в лог-файл кожного разу, коли корабель переміщується.

- Ключовим моментом у джерелі подій є те, що ми гарантуємо, що всі зміни в об'єктах домену ініціюються об'єктами подій. Це призводить до того, що над журналом подій можна побудувати низку об'єктів:
- Повна перебудова: Ми можемо повністю відкинути стан програми і відновити його, повторно запустивши події з журналу подій на порожньому додатку.
- Тимчасовий запит: Ми можемо визначити стан програми в будь-який момент часу. Умовно ми робимо це, починаючи з порожнього стану і повторно запускаючи події до певного часу або події. Ми можемо піти далі, розглядаючи декілька часових ліній (аналогічно до розгалуження в системі контролю версій).
- Відтворення подій: Якщо ми виявимо, що минула подія була неправильною, ми можемо обчислити наслідки, скасувавши її та пізніші події, а потім повторивши нову подію та пізніші події. (Або навіть відкинувши стан програми і повторно відтворивши всі події з правильною подією по черзі). Ця ж техніка може обробляти події, отримані у неправильній послідовності, що є поширеною проблемою у системах, які взаємодіють за допомогою асинхронного обміну повідомленнями.

Поширеним прикладом програми, яка використовує Event Sourcing, є система контролю версій. Така система досить часто використовує тимчасові запити. Subversion використовує повні перезбірки щоразу, коли ви використовуєте дамп і відновлення для переміщення даних між файлами сховища.

**NoSQL**, також відомий як "не тільки SQL", "не-SQL", - це підхід до проектування баз даних, який дозволяє зберігати і запитувати дані поза

традиційними структурами, що використовуються в реляційних базах даних. Хоча він все ще може зберігати дані, що містяться в реляційних системах управління базами даних, він просто зберігає їх інакше порівняно з ними. Рішення про використання реляційної бази даних у порівнянні з нереляційною базою даних значною мірою є контекстним і залежить від конкретного випадку використання.

Замість типової табличної структури реляційної бази даних, бази даних NoSQL зберігають дані в одній структурі даних, наприклад, у вигляді документа JSON. Оскільки такий дизайн нереляційної бази даних не вимагає схеми, він пропонує швидку масштабованість для управління великими і, як правило, неструктурованими наборами даних.

NoSQL також є типом розподіленої бази даних, що означає, що інформація копіюється і зберігається на різних серверах, які можуть бути віддаленими або локальними. Це забезпечує доступність і надійність даних. Якщо частина даних виходить з ладу, решта бази даних може продовжувати працювати.

Сьогодні компаніям необхідно управляти великими обсягами даних на високих швидкостях з можливістю швидкого масштабування для запуску сучасних веб-додатків майже в кожній галузі. В епоху розвитку хмарних технологій, великих даних, мобільних і веб-додатків, бази даних NoSQL забезпечують таку швидкість і масштабованість, що робить їх популярним вибором завдяки своїй продуктивності і простоті використання.

Мова структурованих запитів (SQL) часто згадується у зв'язку з NoSQL. Щоб краще зрозуміти різницю між NoSQL та SQL, варто звернутися до історії SQL - мови програмування, яка використовується для отримання певної інформації з бази даних.

До появи реляційних баз даних компанії використовували ієрархічні системи баз даних з деревоподібною структурою таблиць даних. Ці ранні системи управління базами даних (СУБД) дозволяли користувачам організовувати великі обсяги даних. Однак вони були складними, часто призначалися для конкретних додатків і обмежували можливості пошуку

інформації в даних. Ці обмеження врешті-решт призвели до розробки реляційних систем управління базами даних, які впорядковували дані в таблицях. SQL забезпечив інтерфейс для взаємодії з реляційними даними, що дозволило аналітикам з'єднувати таблиці шляхом об'єднання на основі спільних полів.

З часом вимоги до швидшого і більш розрізненого використання великих наборів даних ставали все більш важливими для нових технологій, таких як додатки для електронної комерції. Програмістам потрібно було щось більш гнучке, ніж бази даних SQL (тобто реляційні бази даних). Такою альтернативою стала NoSQL.

Хоча NoSQL є альтернативою SQL, ця технологія жодним чином не замінила бази даних SQL. Наприклад, уявімо, що ви керуєте роздрібними замовленнями в компанії. У реляційній моделі окремі таблиці будуть керувати даними про клієнтів, замовленнями і товарами окремо, і вони будуть об'єднані за допомогою унікального спільного ключа, наприклад, ідентифікатора клієнта або ідентифікатора замовлення. Хоча це чудовий спосіб зберігання та швидкого пошуку даних, він вимагає значного обсягу пам'яті. Коли ви хочете додати більше пам'яті, бази даних SQL можуть масштабуватися лише вертикально, а не горизонтально, а це означає, що ваша можливість додати більше пам'яті обмежена наявним у вас обладнанням. В результаті вертикальне масштабування в кінцевому підсумку обмежує можливості зберігання та пошуку даних вашої компанії.

Для порівняння, бази даних NoSQL є нереляційними, що усуває необхідність з'єднання таблиць. Вбудована функція шардеризації та висока доступність полегшують горизонтальне масштабування. Якщо одного сервера бази даних недостатньо для зберігання всіх даних або обробки всіх запитів, робоче навантаження можна розподілити між двома або більше серверами, що дозволяє компаніям горизонтально масштабувати свої дані.

Хоча кожен тип бази даних має свої переваги, компанії зазвичай використовують як NoSQL, так і реляційні бази даних в одному додатку. Сучасні хмарні провайдери можуть підтримувати бази даних SQL або NoSQL.

NoSQL надає інші можливості для організації даних різними способами. Пропонуючи різноманітні структури даних, NoSQL можна застосовувати в аналітиці даних, управлінні великими даними, соціальних мережах і розробці мобільних додатків.

База даних NoSQL управляє інформацією, використовуючи будь-яку з цих первинних моделей даних:

**Сховище пар ключ-значення**

Зазвичай це вважається найпростішою формою баз даних NoSQL. Ця модель даних без схеми організована у вигляді словника пар ключ-значення, де кожен елемент має ключ і значення. Ключем може бути щось подібне до бази даних SQL, наприклад, ідентифікатор кошика для покупок, а значенням - масив даних, наприклад, кожен окремий товар у кошику цього користувача. Він зазвичай використовується для кешування та зберігання інформації про сеанси користувача, наприклад, про кошики. Однак він не є ідеальним, коли вам потрібно витягти кілька записів одночасно. Redis і Memcached є прикладами баз даних ключ-значення з відкритим вихідним кодом.

**Сховище документів**

Як випливає з назви, бази даних документів зберігають дані у вигляді документів. Вони можуть бути корисними для управління напівструктурованими даними, які зазвичай зберігаються у форматах JSON, XML або BSON. Це зберігає дані разом, коли вони використовуються в додатках, зменшуючи кількість перекладу, необхідного для використання даних. Розробники також отримують більшу гнучкість, оскільки схеми даних не повинні збігатися в різних документах (наприклад, ім'я та прізвище). Однак це може бути проблематично для складних транзакцій, що призводить до пошкодження даних. Популярні випадки використання баз даних документів включають системи управління контентом та профілі користувачів.

**Ширококолонне сховище**

Ці бази даних зберігають інформацію в стовпцях, що дозволяє користувачам отримувати доступ лише до конкретних стовпців, які їм потрібні,

не виділяючи додаткову пам'ять для несуттєвих даних. Ця база даних намагається усунути недоліки сховищ ключ-значення і сховищ документів, але оскільки вона може бути складнішою в управлінні, її не рекомендується використовувати для нових команд і проектів. Apache HBase та Apache Cassandra є прикладами баз даних з відкритим вихідним кодом та широкими стовпцями. Apache HBase побудована на основі розподіленої файлової системи Hadoop, яка забезпечує спосіб зберігання розріджених наборів даних, що широко використовується в багатьох додатках для роботи з великими даними. Apache Cassandra, з іншого боку, був розроблений для управління великими обсягами даних на декількох серверах і кластерах, які охоплюють кілька центрів обробки даних. Він використовується для різноманітних випадків використання, таких як соціальні мережі та аналітика даних у реальному часі.

**Сховище графів**

Цей тип бази даних зазвичай містить дані з графа знань. Елементи даних зберігаються у вигляді вузлів, ребер і властивостей. Вузлом може бути будь-який об'єкт, місце або людина. Ребро визначає зв'язок між вузлами. Наприклад, вузлом може бути клієнт, наприклад, IBM, і агентство, наприклад, Ogilvy. Ребро буде класифікувати відносини як клієнтські відносини між IBM та Ogilvy.

Бази даних графів використовуються для зберігання та управління мережею зв'язків між елементами всередині графа. Neo4j (посилання знаходиться за межами IBM), сервіс графових баз даних на основі Java з відкритим вихідним кодом, де користувачі можуть придбати ліцензії на резервне копіювання в Інтернеті та розширення високої доступності, або попередньо запаковану ліцензійну версію з резервним копіюванням та розширеннями, включеними в комплект.

**Приклади баз даних NoSQL**

Багато компаній увійшли в середовище NoSQL. На додаток до згаданих вище, ось кілька популярних баз даних NoSQL:

- **Apache CouchDB**, база даних з відкритим вихідним кодом на основі документів JSON, яка використовує JavaScript як мову запитів.

- **Elasticsearch**, база даних на основі документів, яка включає повнотекстову пошукову систему.
- **Couchbase**, база даних ключ-значення та документів, яка дозволяє розробникам створювати гнучкі та адаптивні додатки для хмарних, мобільних та периферійних обчислень.

Кожен тип баз даних NoSQL має сильні сторони, які роблять його кращим для конкретних випадків використання. Однак всі вони мають наступні переваги для розробників і створюють основу для надання кращого сервісу, в тому числі і клієнтам:

- **Економічна ефективність**: Підтримка високоякісних комерційних баз даних коштує дорого. Вони вимагають придбання ліцензій, кваліфікованих менеджерів баз даних та потужного обладнання для вертикального масштабування. NoSQL бази даних дозволяють швидко масштабуватися горизонтально, краще розподіляючи ресурси для мінімізації витрат.
- **Гнучкість**: Горизонтальне масштабування та гнучка модель даних також означає, що бази даних NoSQL можуть обробляти великі обсяги даних, що швидко змінюються, що робить їх ідеальними для гнучкої розробки, швидких ітерацій та частих змін коду.
- **Реплікація**: Функція реплікації NoSQL копіює та зберігає дані на декількох серверах. Така реплікація забезпечує надійність даних, гарантуючи доступ до них під час простою та захищаючи від втрати даних, якщо сервери виходять з ладу.
- **Швидкість**: NoSQL забезпечує більш швидке та гнучке зберігання та обробку даних для всіх користувачів, від розробників до відділів продажу та клієнтів. Швидкість також робить бази даних NoSQL більш придатними для сучасних, складних веб-додатків, сайтів електронної комерції або мобільних додатків.

Коротше кажучи, бази даних NoSQL забезпечують високу продуктивність, доступність і масштабованість.

Структура і тип бази даних NoSQL, яку ви оберете, залежатиме від того, як ваша організація планує її використовувати. Ось кілька конкретних застосувань для різних типів баз даних NoSQL.

- **Керування зв'язками даних**: Управління складною сукупністю даних і взаємозв'язками між ними зазвичай здійснюється за допомогою графової бази даних NoSQL. Сюди відносяться системи рекомендацій, графи знань, програми для виявлення шахрайства та соціальні мережі, де зв'язки між людьми встановлюються за допомогою різних типів даних.
- **Продуктивність з низькою затримкою**: Ігри, додатки для домашнього фітнесу та рекламні технології вимагають високої пропускної здатності для управління даними в режимі реального часу. Ця інфраструктура забезпечує найбільшу цінність для споживача, чи то оновлення ринкових торгів, чи показ найрелевантніших оголошень. Веб-додатки потребують баз даних NoSQL в пам'яті, щоб забезпечити швидкий час відгуку і керувати піками використання без затримок, які можуть виникати при використанні дискового сховища.
- **Масштабування і великі обсяги даних**: Електронна комерція вимагає здатності керувати величезними сплесками використання, будь то одноденний розпродаж або сезон святкових покупок. Бази даних ключових значень часто використовуються в додатках електронної комерції, оскільки їхня проста структура легко масштабується під час великого трафіку. Ця гнучкість є цінною для ігор, реклами та додатків Інтернету речей (IoT).

Безумовно, рішення, що використовують лише NoSQL, мають місце в центрі обробки даних, щоб задовольнити потреби конкретних додатків. Але Postgres має можливості для підтримки більшості додатків, які розгортають сучасні підприємства і які потребують напівструктурованих або неструктурованих даних. Завдяки JSON для баз даних документів і типу даних HSTORE для пар ключ/значення, Postgres підтримує неструктуровані і напівструктуровані дані поряд з реляційними даними.

Postgres спочатку був спроектований як об'єктно-реляційна база даних, розроблена спеціально для забезпечення розширюваності. Вона підтримує об'єкти, класи, користувацькі типи даних і методи. У перші роки проекту Postgres це було проблематично, оскільки сповільнювало цикли розробки, оскільки новий код потрібно було повністю інтегрувати, щоб все працювало з усім іншим. Однак, оскільки за останні 15 років Postgres став більш багатофункціональним, ця первісна перешкода перетворилася на унікальну перевагу. Той факт, що Postgres є об'єктно-реляційною базою даних, означає, що нові можливості можна розробляти і підключати до бази даних по мірі розвитку потреб.

Використовуючи цей рівень розширюваності, розробники Postgres розширили базу даних, включивши в неї нові функції і можливості в міру появи нових робочих навантажень, що вимагають більшої гнучкості в моделі даних. Найбільш релевантними прикладами в обговоренні NoSQL є JSON і HSTORE. Завдяки JSON і HSTORE, Postgres може підтримувати додатки, які вимагають великої гнучкості в моделі даних.

Однією з ключових переваг Postgres є легка інтеграція звичайних операторів SQL для таблиць і записів ANSI SQL з посиланнями JSON і HSTORE, що вказують на документи і пари ключ-значення. Оскільки JSON і HSTORE є розширенням базової моделі Postgres, запити використовують той самий синтаксис, виконуються в тому самому транзакційному середовищі ACID і покладаються на ті самі технології планування, оптимізації та індексування запитів, що й звичайні SQL-запити.

Postgres надає ряд функцій для зв'язку між JSON і ANSI SQL. Це важлива можливість, коли додатки і моделі даних стають зрілими, і розробники починають розпізнавати нові структури даних і зв'язки. Postgres може створити міст між SQL і JSON, наприклад, перетворивши таблицю SQL на набір даних у форматі JSON. Ця можливість дозволяє розробникам і DBA починати роботу з неструктурованим набором даних, а по мірі виконання

проекту регулювати баланс між структурованими і неструктурованими даними.

Здатність Postgres підтримувати сховища ключових значень і документів в одній базі даних дозволяє користувачам задовольняти зростаючі потреби, використовуючи перевірені, найкращі у своєму класі технології з відкритим вихідним кодом. Використання можливостей NoSQL в Postgres для вирішення більшої кількості проблем з даними замість того, щоб відразу звертатися до рішення NoSQL, в кінцевому підсумку означає менші витрати, менший ризик і меншу складність, забезпечуючи при цьому відповідність робочих навантажень корпоративного класу стандарту ACID і довгострокову життєздатність корпоративних даних.

## 2.4. Опис структури системи та алгоритмів її функціонування

Розроблений додаток працює як односторінковий додаток, тобто фронтенд сам рендерить сторінки, використовуючи JSON дані отримані по REST API з бекенду.

Структура папок клієнтської частини виглядає наступним чином (рис. 2.6.):

- Папка assets – іконки в форматі svg
- Components – React компоненти
- Contexts – контексти для отримування об’єктів без прокидування їх через пропси
- Firebase – код для роботи з Firebase
- Hoc – допоміжні HOC функції
- Pages – компоненти сторінок

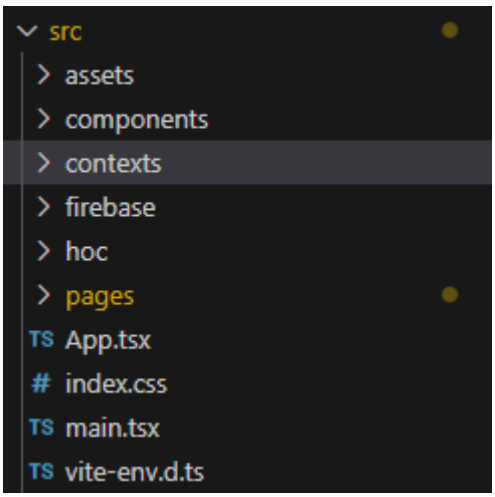


Рис 2.6. Структура папки з кодом клієнтської частини

Серверна частина працює як REST API і складається з наступних контролерів:

- TestsController – CRUD для тестів
- SessionController – контролер сесії проходження тесту, який використовує event sourcing
- QuestionsController – CRUD для питань

## 2.5. Обгрунтування та організація вхідних та вихідних даних програми

Клієнтська частина застосунку спілкується із серверною за допомогою REST API. Вхідні дані серіалізуються на клієнті у формат JSON і надсилаються по HTTP на сервер. Вихідні дані клієнт отримує з сервера теж у форматі JSON.

## 2.6. Опис роботи розробленої системи

### 2.6.1. Використані технічні засоби

Під час розробки використовувались контейнери на віртуальній машині WSL.

### 2.6.2. Використані програмні засоби

Під час розробки використовувались наступні програмні засоби:

- Браузер Chrome 114
- Docker
- .NET 7
- PostgreSQL
- VS Code
- Node.js 16
- Vite
- NPM

### 2.6.3. Виклик та завантаження програми

Для завантаження клієнтського застосунку користувачеві потрібно відкрити сторінку платформи в браузері.

Для завантаження серверної частини потрібно встановити .NET CLI і Postgres, у конфігураційному файлі замінити рядок підключення з логіном і паролем до бази даних, перейти в папку з кодом у терміналі та запустити сервер командою dotnet run.

### 2.6.4. Опис інтерфейсу користувача

На головній сторінці платформи (рис. 2.7.) є вкладки, за допомогою яких можна переміщатися між головною сторінкою та сторінкою з результатами пройдених тестів (рис. 2.8.). На головній сторінці виводяться на екран картки доступних для проходження тестів.

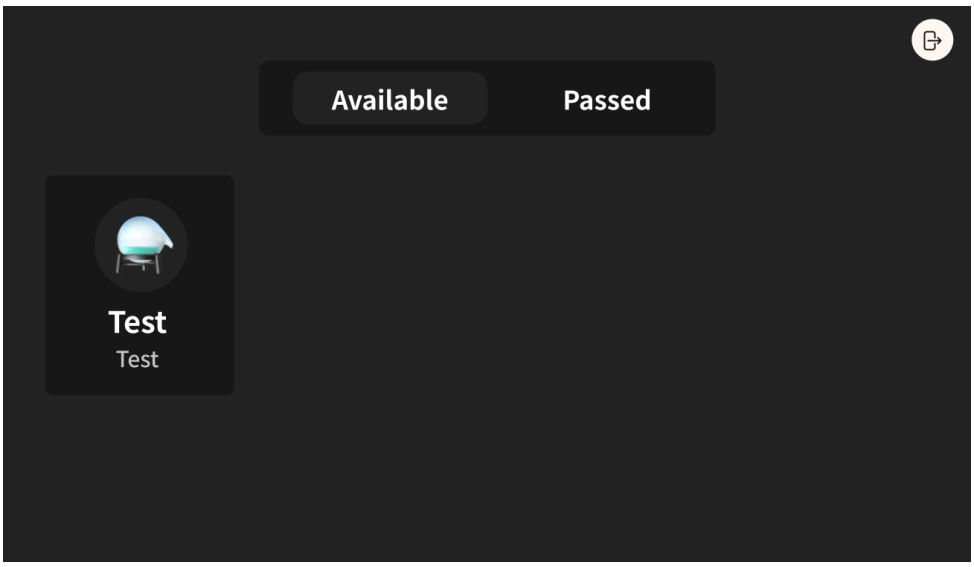


Рис 2.7. Головна сторінка

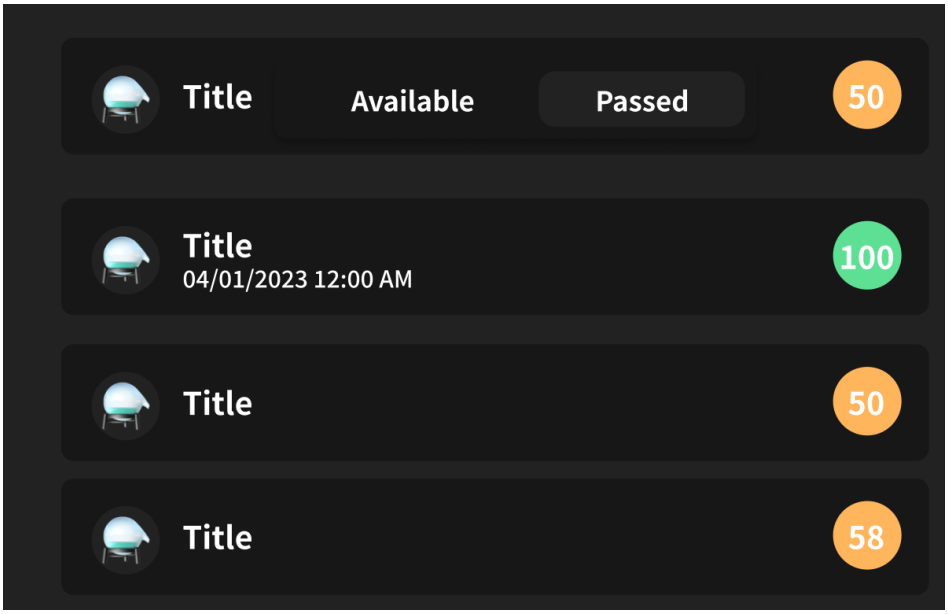


Рис. 2.8. Результати тестів

Якщо натиснути на картку на екрані з пройденими тестами, користувач зможе побачити оцінку і свої результати (рис. 2.9.).

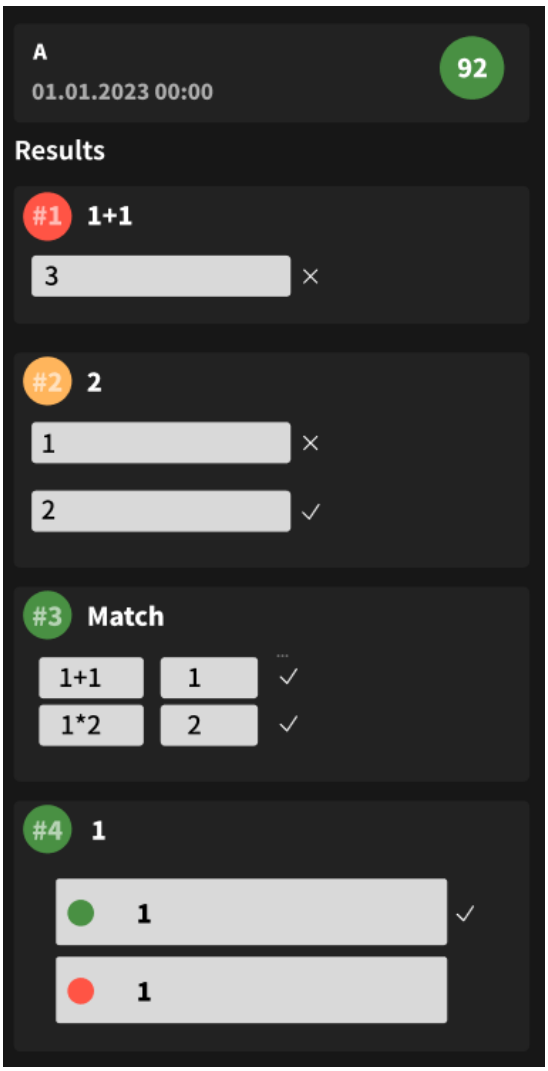


Рис. 2.9. Результати тесту

Після переходу по посиланню на тест (рис. 2.10.), користувач зможе побачити як тест називається, з якого він предмету, коли його можна пройти і скільки часу дається на виконання тесту.

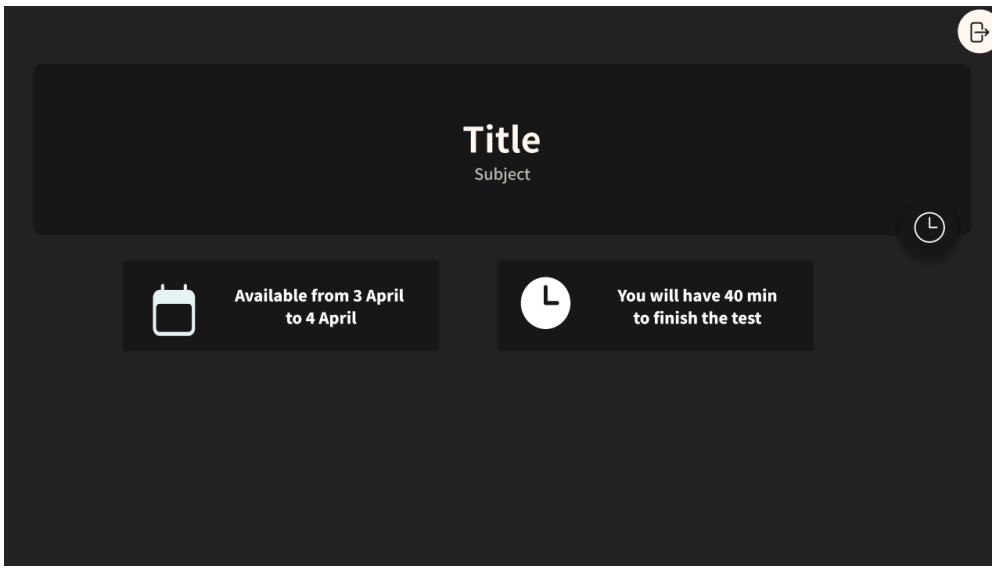


Рис. 2.10. Сторінка тесту

Коли тест буде активним, учні зможуть його розпочати, натиснувши кнопку. На екрані тесту (рис. 2.11.) користувач може відповідати на запитання. Можна переходити між запитаннями, якщо до них є доступ, і змінювати відповіді, поки не вийшов час.

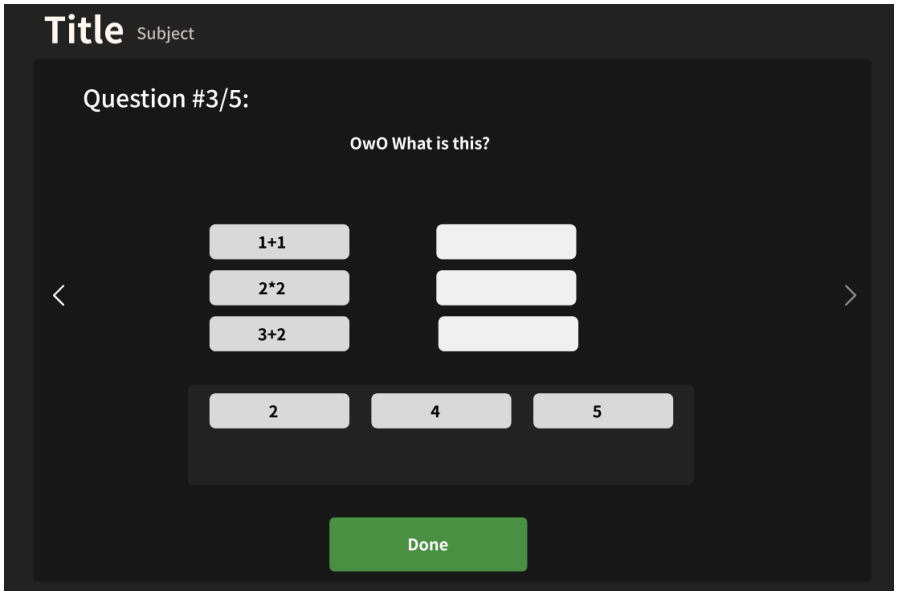


Рис. 2.11. Приклад завдання

Якщо користувач користується вчительським обліковим записом, то головна сторінка буде показувати список створених ним тестів. Якщо натиснути на тест, то можна буде відредагувати його (рис. 2.12.), але тільки якщо він не почався.

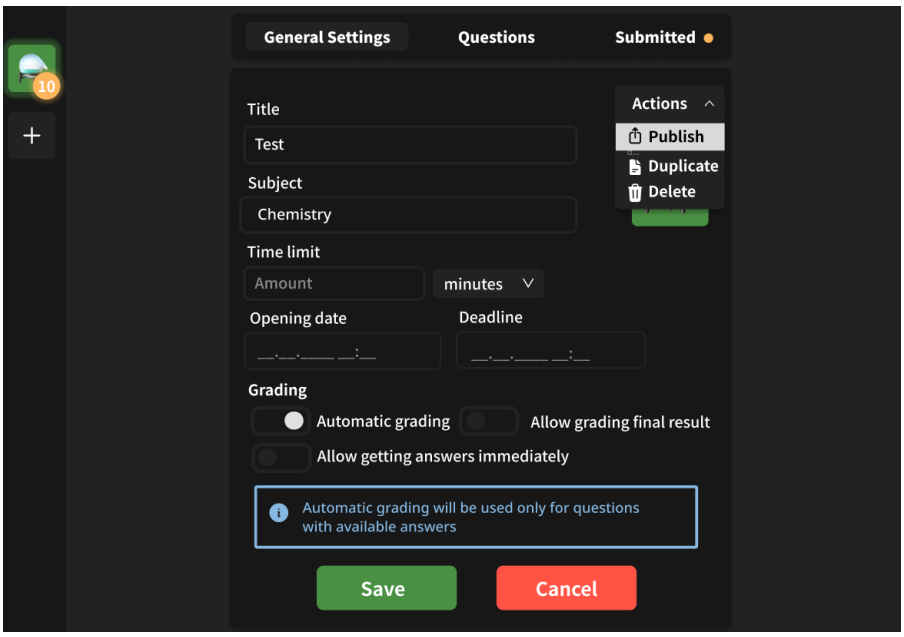


Рис. 2.12. Редактор тесту

На вкладці з питаннями (рис. 2.13.) можна переміщувати, редагувати і додавати нові питання.

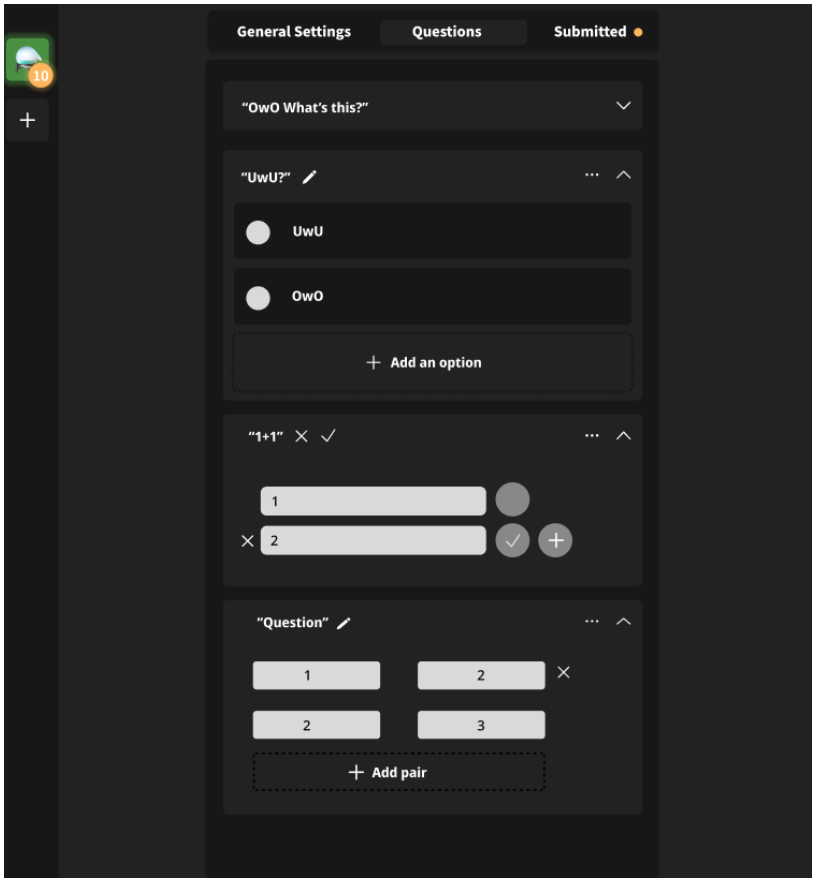


Рис. 2.13. Редактор питань

Якщо учні здали нові роботи, то на заголовку вкладки “Submitted” загориться жовтий індикатор. Якщо перейти на цю вкладку (рис. 2.14.), на екран будуть виведені всі здані роботи, які можна відфільтрувати і відсортувати.

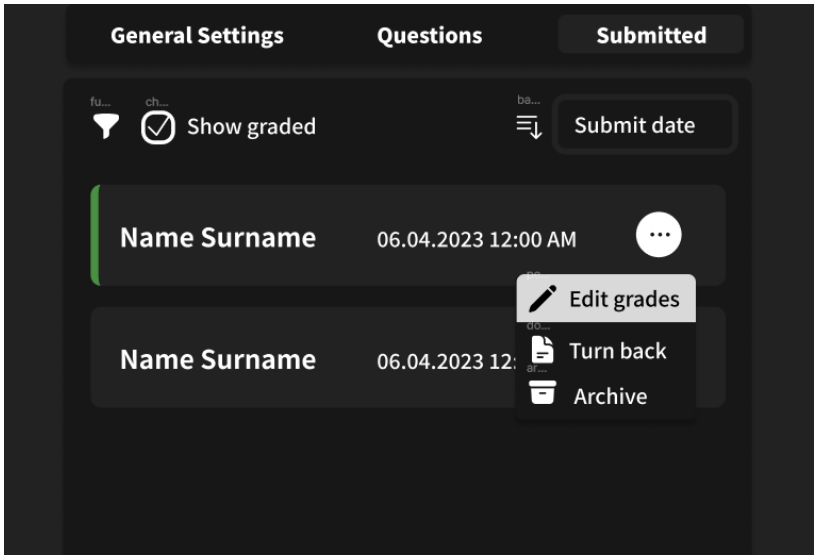


Рис. 2.14. Вкладка «Submitted»

Екран перегляду результату (рис. 2.15.) виглядає майже так само як і для звичайного користувача, бо використовує ті ж компоненти.

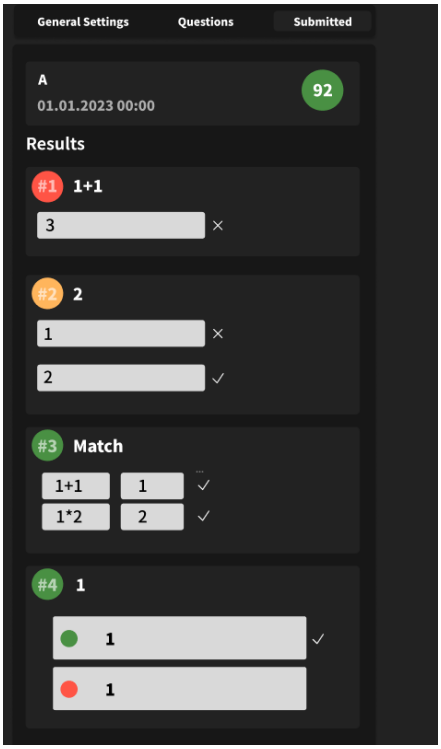


Рис. 2.15. Екран перегляду результату

# РОЗДІЛ 3
# ЕКОНОМІЧНА ЧАСТИНА

Під час розробки програмного забезпечення важливими етапами є визначення трудомісткості розробки і розрахунок витрат на створення програмного продукту.

## 3.1. Визначення трудомісткості розробки програмного забезпечення

Задані дані:

1. передбачуване число операторів (підпрограм) – 600;
2. коефіцієнт складності програми – 1,4;
3. коефіцієнт корекції програми в ході її розробки – 0,2;
4. годинна заробітна плата програміста [1], грн/год – 150;
5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,3;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,0;
7. вартість машино-години ЕОМ, грн/год – 21 (розрахунок нижче).

Собівартість машино-години ЕОМ, грн/год:

$$M = \frac{A + S_1 + S_2 + S_3}{H}, \text{грн/год}, \tag{3.1}$$

де $A$ – річна сума амортизації (оновлення, обслуговування, ремонтування та ін.), грн;

$S_1$ – річні витрати на електроенергію, грн;

$S_2$ – річні витрати на ПЗ, грн;

$S_3$ – річні накладні розходи, грн;

$H$ – дійсний годовий фонд часу роботи, годин.

$$M = \frac{40000 + 5000 + 10000 + 2000}{2710} = 21, \text{грн/год},$$

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

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

де $t_o$ - витрати праці на підготовку й опис поставленої задачі (приймається 50);

$t_u$ - 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;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q * C * (1 + p), \quad (3.3)$$

де $q$ - передбачуване число операторів;

$C$ - коефіцієнт складності програми;

$p$ - коефіцієнт корекції програми в ході її розробки.

$$Q = 600 * 1,4 * (1 + 0,2) = 1008.$$

Витрати праці на вивчення опису задачі $t_u$ визначається з урахуванням уточнення опису і кваліфікації програміста:

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

де $B$ - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

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

$$t_u = \frac{1008 * 1,3}{80 * 1} = \frac{1310,4}{80} = 16,38 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25) * k}, \text{ людино-годин,} \quad (3.5)$$

$$t_a = \frac{1008}{20 * 1} = 50,4 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25) * k}, \text{ людино-годин,} \quad (3.6)$$

$$t_n = \frac{1008}{25 * 1} = 40,3 \text{ людино-годин,}$$

Витрати праці на налагодження програми на ЕОМ:
– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5) * k}, \text{ людино-годин,} \quad (3.7)$$

$$t_{отл} = \frac{1008}{4 * 1} = 252 \text{ людино-годин;}$$

– за умови комплексного налагодження завдання:

$$t^k_{отл} = 1,5 * t_{отл}, \text{ людино-годин,} \quad (3.8)$$

$$t^k_{отл} = 1,5 * 252 = 378 \text{ людино-годин.}$$

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до}, \text{ людино-годин,} \quad (3.9)$$

де $t_{др}$ - трудомісткість підготовки матеріалів і рукопису.

$$t_{др} = \frac{Q}{15..20 * k}, \text{ людино-годин,} \quad (3.10)$$

$$t_{др} = \frac{1008}{20 * 1} = 50,4 \text{ людино-годин.}$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0,75 * t_{др}, \text{ людино-годин,} \quad (3.11)$$

$$t_{до} = 0,75 * 50,4 = 37,8 \text{ людино-година,}$$

$$t_д = 50,4+37,8 = 88,19 \text{ людино-годин.}$$

Тепер розрахуємо трудомісткість ПЗ:

$t = 37,8 + 16,38 + 50,4 + 40,3 + 252 + 88,19 = 485,07$ людино-годин.

### 3.2. Витрати на створення програмного забезпечення

Витрати на створення ПЗ *Кпо* включають витрати на заробітну плату виконавця програми *Зз/п* і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{по} = З_{зп} + З_{мв}, \text{ грн.} \quad (3.12)$$

Заробітна плата виконавців визначається за формулою:

$$З_{зп} = t * С_{пр}, \text{ грн,} \quad (3.13)$$

де $t$ - загальна трудомісткість, людино-годин;
$С_{пр}$ - середня годинна заробітна плата програміста, грн/година.

$$З_{зп} = 485,07 * 150 = 72760,5 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми:

$$З_{мв} = t_{отл} * С_{мч}, \text{ грн,} \quad (3.14)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год,
$С_{мч}$ - вартість машино-години ЕОМ, грн/год,
$С_{мч}$ = 21, грн/год.

$$З_{мв} = 378 * 21 = 7938, \text{ грн.}$$

Визначені в такий спосіб витрати на створення програмного забезпечення є частиною одноразових капітальних витрат на створення ПЗ:

$$К_{по} = 72760,5 + 7938 = 80698,5 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k * F_p}, \text{ міс.,} \quad (3.15)$$

де $B_k$ - число виконавців (приймається 1),
$F_p$ - місячний фонд робочого часу (40 годин на тиждень $F_p$ = 176 годин).

$$T = \frac{485,07}{1 * 176} = 2,7 міс.$$

Вартість даного продукту становить 80 тис. грн. і не вимагає додаткових витрат. Очікуваний час розробки становить 2,7 місяців. Цей термін пов'язаний зі значним числом операторів, і включає час на дослідження і розробку алгоритму вирішення поставленого завдання, програмування по готовому алгоритму, налагодження програми і підготовку документації.

# ВИСНОВКИ

В рамках кваліфікаційної роботи було розроблено веб додаток для онлайн тестування школярів.

Це програмне забезпечення призначено для використання в школах. Застосунок зручний для користування, має гарний інтерфейс, не має багів, його легко інтегрувати в існуючу систему онлайн навчання.

Під час виконання кваліфікаційної роботи були виконані наступні задачі:

- Проаналізовано предметну область задачі.
- Розроблено дизайн додатку.
- Спроектована внутрішня архітектура.
- Визначено трудомісткість розробленого додатку.
- Підрахована вартість розробки.

Розроблений додаток дозволяє:

- Створювати тести.
- Проходити тести і отримувати оцінки.
- Дивитись відповіді і виставляти оцінки.

Додаток реалізований за допомогою фреймворків .NET і ReactJS.

Час створення платформи становить 2,7 місяців, а його оціночна вартість становить 80 тис. грн.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Freeman, A., Pro ASP.NET Core MVC 3, Eighth Edition. Apress. Berkeley. – 2020. - 157 p.

2. Dagraça M. Learning C# 7 By Developing Games with Unity 2017: Learn C# Programming by building fun and interactive games with Unity. Packt Publishing Ltd. - 2017. – 150 p.

3. Al-Bastami B. G., Naser S. S. Abu. Design and Development of an Intelligent Tutoring System for C# Language. European academic research, 2017,

4. Chan J. C#: Learn C# in One Day and Learn It Well. LCF Publishing. - 2017. - 161 p.

5. Greene J. Head First C#: A Learner's Guide to Real-World Programming with C#, XAML, and .NET. O'Reilly UK Ltd. - 2021. – 745 p.

6. Albahari J. C# 8.0 Pocket Reference: Instant Help for C# 8.0 Programmers. O'Reilly Media, Inc, USA. – 2019. – 241 p.

7. Boehm A. Murach's C# 2015. MIKE MURACH & ASSOC INC. – 2015. – 126 p.

8. Skeet J. C# in Depth. Manning. – 2019. – 528 p.

9. The C# Player's Guide. RB Whitaker. – 2017. – 406 p.

10. Ferrone H. Learning C# by Developing Games with Unity. 2019. – 2019. – 324 p.

11. Troelsen A. Pro C# 7: With .NET and .NET Core. Apress. 2019. – 2017. – 1437 p.

12. Robert C. Martin. Agile Principles, Patterns, and Practices in C#. Prentice Hall. – 2006. – 732 p.

13. Sharp J. Microsoft Visual C# Step by Step. Microsoft Press. – 2018. – 791 p.

14. Albahari J. C# 7.0 in a Nutshell: The Definitive Reference. O'Reilly UK Ltd. – 2017. – 1070 p.

15. S. Cleary. Concurrency in C# Cookbook: Asynchronous, Parallel, and Multithreaded Programming. O'Reilly UK Ltd. – 2019. – 234 p.

16. Wagner B. Effective C# (Covers C# 6.0), (includes Content Update Program). Sams Publishing. – 2016. – 288 p.

17. W. Piekarski, B. Thomas. An Object Oriented Software Architecture for 3D Mixed Reality Applications. – 2016. – 45 p.

18. B. Bashhar. Design and Development of an ITS for C# Language. European Academic Research. – 2017. – 410 p.

**ДОДАТОК А**

## КОД ПРОГРАМИ

### HMACSigner.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;

namespace TestsAPI
{
    public class HMACSigner : ISigner
    {
        private HMACSHA256 hmacsha256;
        public HMACSigner(string key)
        {
            this.hmacsha256 = new HMACSHA256(Encoding.UTF8.GetBytes(key));
        }
        public byte[] Sign(byte[] data)
        {
            return hmacsha256.ComputeHash(data);
```

```
        }

        public bool Verify(byte[] data, byte[] signature)
        {
            byte[] generated = Sign(data);
            return generated.SequenceEqual(signature);
        }
    }
}
```

**IEncoder.cs**

```
namespace TestsAPI
{
    public interface IEncoder
    {
        string Encode(byte[] value);
        byte[] Decode(string value);
    }
}
```

**ISigner.cs**

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

namespace TestsAPI
{
   public interface ISigner
   {
      byte[] Sign(byte[] data);
      bool Verify(byte[] data, byte[] signature);
   }
}
```

**RandomNumberGenerator.cs**

```
/// <summary>
/// Random Number Generator based on Mersenne-Twister algorithm
///
/// Usage :
///    RandomNumberGenerator.Instance.Generate());
```

```
        ///    RandomNumberGenerator.Instance.Generate(1.1,2.2);
        ///    RandomNumberGenerator.Instance.Generate(1,100)
        ///
        /// inspired from : http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/VERSIONS/C-LANG/980409/mt19937-2.c
        /// </summary>
        namespace TestsAPI
        {

           public class RandomNumberGenerator
           {
              #region constants
              /// <summary>
              /// N
              /// </summary>
              private static readonly int N = 624;

              /// <summary>
              /// M
              /// </summary>
              private static readonly int M = 397;

              /// <summary>
              /// Constant vector a
```

```
/// </summary>
private readonly UInt32 MATRIX_A = 0x9908b0df;

/// <summary>
/// most significant w-r bits
/// </summary>
private readonly UInt32 UPPER_MASK = 0x80000000;

/// <summary>
/// least significant r bits
/// </summary>
private readonly UInt32 LOWER_MASK = 0x7fffffff;

/// <summary>
/// Tempering mask B
/// </summary>
private readonly UInt32 TEMPERING_MASK_B = 0x9d2c5680;

/// <summary>
/// Tempering mask C
/// </summary>
private readonly UInt32 TEMPERING_MASK_C = 0xefc60000;

/// <summary>
```

```
        /// Last constant used for generation
        /// </summary>
        private readonly double FINAL_CONSTANT = 2.3283064365386963e-
10;
        #endregion

        public RandomNumberGenerator(ulong seed)
        {
            //init
            this.sgenrand(seed);
        }

        #region helpers methods
        private ulong TEMPERING_SHIFT_U(ulong y)
        {
            return y >> 11;
        }

        private ulong TEMPERING_SHIFT_S(ulong y)
        {
            return y << 7;
        }

        private ulong TEMPERING_SHIFT_T(ulong y)
```

```
{
    return y << 15;
}

private ulong TEMPERING_SHIFT_L(ulong y)
{
    return y >> 18;
}
#endregion

#region properties

/// <summary>
/// the array for the state vector
/// </summary>
private readonly ulong[] mt = new ulong[625];

/// <summary>
/// mti==N+1 means mt[N] is not initialized
/// </summary>
private int mti = N + 1;
#endregion

#region engine
```

```
        /// <summary>
        /// setting initial seeds to mt[N] using
        /// the generator Line 25 of Table 1 in
        /// [KNUTH 1981, The Art of Computer Programming Vol. 2 (2nd Ed.),
pp102]
        /// </summary>
        /// <param name="seed"></param>
        private void sgenrand(ulong seed)
        {
            mt[0] = seed & 0xffffffff;
            for (mti = 1; mti < N; mti++)
                mt[mti] = (69069 * mt[mti - 1]) & 0xffffffff;
        }

        private double genrand()
        {
            ulong y;
            ulong[] mag01 = new ulong[2] { 0x0, MATRIX_A };
            /* mag01[x] = x * MATRIX_A  for x=0,1 */

            if (mti >= N)
            { /* generate N words at one time */
                int kk;
```

```
    if (mti == N + 1)   /* if sgenrand() has not been called, */
        sgenrand(4357); /* a default initial seed is used   */

    for (kk = 0; kk < N - M; kk++)
    {
        y = (mt[kk] & UPPER_MASK) | (mt[kk + 1] & LOWER_MASK);
        mt[kk] = mt[kk + M] ^ (y >> 1) ^ mag01[y & 0x1];
    }
    for (; kk < N - 1; kk++)
    {
        y = (mt[kk] & UPPER_MASK) | (mt[kk + 1] & LOWER_MASK);
        mt[kk] = mt[kk + (M - N)] ^ (y >> 1) ^ mag01[y & 0x1];
    }
    y = (mt[N - 1] & UPPER_MASK) | (mt[0] & LOWER_MASK);
    mt[N - 1] = mt[M - 1] ^ (y >> 1) ^ mag01[y & 0x1];

    mti = 0;
  }

  y = mt[mti++];
  y ^= TEMPERING_SHIFT_U(y);
  y ^= TEMPERING_SHIFT_S(y) & TEMPERING_MASK_B;
  y ^= TEMPERING_SHIFT_T(y) & TEMPERING_MASK_C;
```

```
            y ^= TEMPERING_SHIFT_L(y);

            //reals: (0,1)-interval
            //return y; for integer generation
            return ((double)y * FINAL_CONSTANT);
        }
        #endregion

        #region public methods
        /// <summary>
        /// Generate a random number between 0 and 1
        /// </summary>
        /// <returns></returns>
        public double Generate()
        {
            return this.genrand();
        }

        /// <summary>
        /// Generate an int between two bounds
        /// </summary>
        /// <param name="lowerBound">The lower bound (inclusive)</param>
        /// <param name="higherBound">The higher bound (inclusive)</param>
```

```
        /// <returns></returns>
        public double Generate(int lowerBound, int higherBound)
        {
            if (higherBound < lowerBound)
            {
                return double.NaN;
            }
            return Convert.ToInt32(Math.Floor(this.Generate(lowerBound * 1.0d, higherBound * 1.0d)));
        }

        /// <summary>
        /// Generate a double between two bounds
        /// </summary>
        /// <param name="lowerBound">The lower bound (inclusive)</param>
        /// <param name="higherBound">The higher bound (inclusive)</param>
        /// <returns>The random num or NaN if higherbound is lower than lowerbound</returns>
        public double Generate(double lowerBound, double higherBound)
        {
            if (higherBound < lowerBound)
            {
                return double.NaN;
            }
```

```
            return (this.Generate() * (higherBound - lowerBound + 1)) +
lowerBound;
        }
        #endregion
    }
}
```

**URLSafeBase64Encoder.cs**

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

namespace TestsAPI
{
    public class URLSafeBase64Encoder : IEncoder
    {
        static readonly char[] padding = { '=' };

        public string Encode(byte[] value)
        {
            return System.Convert.ToBase64String(value)
```

```
                .TrimEnd(padding).Replace('+', '-').Replace('/', '_');
        }

        public byte[] Decode(string returnValue)
        {
            string incoming = returnValue
                .Replace('_', '/').Replace('-', '+');
            switch (returnValue.Length % 4)
            {
                case 2: incoming += "=="; break;
                case 3: incoming += "="; break;
            }
            byte[] bytes = Convert.FromBase64String(incoming);
            return bytes;
        }
    }
}
```

**TestsController.cs**

```
using Marten;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
```

```
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;

namespace TestsAPI.Controllers
{
    public class CreateTestDTO
    {
        public String Title { get; set; }
        public String Subject { get; set; }
        public String Emoji { get; set; }
    }

    public class EditTestDTO
    {
        public string? Title { get; set; }
        public string? Subject { get; set; }
        public string? Emoji { get; set; }
        public double? StartAt { get; set; }
        public double? EndAt { get; set; }
        public string? TimeAvailable { get; set; }
    }


    [ApiController]
```

```
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
[Route("api/[controller]")]
public class TestsController : ControllerBase
{
    private readonly IQuerySession _session;

    public TestsController(IQuerySession session)
    {
        _session = session;
    }

    [HttpGet("")]
    public async Task<IActionResult> Get([FromQuery(Name = "filter")] string? filter)
    {
        if (filter == "CreatedByMe")
        {
            string userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
            var list = await _session.Query<Test>().Where(x => x.Author == userId).ToListAsync();
            return Ok(list.Select(test =>
            {
                return new
```

```
                {
                    Id = test.Id,
                    Title = test.Title,
                    Emoji = test.Emoji,
                };
            }).ToList());
        }

        return Ok(await _session.Query<Test>().Where(x => x.StartAt != null
&& x.EndAt != null && x.EndAt > DateTime.Now).ToListAsync());
    }

    [HttpPost("")]
    public async Task<IActionResult> Post([FromBody] CreateTestDTO
dto)
    {
        if (dto.Subject == "" || dto.Title == "")
            return BadRequest();
        if (dto.Emoji == "") dto.Emoji = ":smiley_cat:";
        string userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;

        Test t = new Test
        {
            Title = dto.Title,
            Subject = dto.Subject,
```

```
            Author = userId,
            Emoji = dto.Emoji,
            EndAt = null,
            StartAt = null,
            TimeAvailable = "",
            Questions = new List<Question>()
        };
        await using var session = _session.DocumentStore.LightweightSession();
        session.Store(t);

        await session.SaveChangesAsync();
        return Ok(t);
    }

    [HttpDelete("{id}")]
    public async Task<IActionResult> Delete(string id)
    {
        var test = await _session.LoadAsync<Test>(new Guid(id));
        await using var session = _session.DocumentStore.LightweightSession();
        session.Delete(test);
        await session.SaveChangesAsync();
        return NoContent();
    }
```

```
[HttpGet("{id}")]
public async Task<IActionResult> GetTest(string id)
{
   var test = await _session.LoadAsync<Test>(new Guid(id));
   return Ok(new
   {
      Id = test.Id,
      Title = test.Title,
      Author = test.Author,
      Subject = test.Subject,
      EndAt = test.EndAt,
      StartAt = test.StartAt,
      TimeAvailable = test.TimeAvailable
   });
}

[HttpPut("{id}")]
public async Task<IActionResult> Put(string id, [FromBody] EditTestDTO dto)
{
   var test = await _session.LoadAsync<Test>(new Guid(id));
   if (dto.Title != null && dto.Title != test.Title)
   {
```

```
            test.Title = dto.Title;
        }
        if (dto.Subject != null && dto.Subject != test.Subject)
        {
            test.Subject = dto.Subject;
        }
        if (dto.Emoji != null && dto.Emoji != test.Emoji)
        {
            test.Emoji = dto.Emoji;
        }
        if (dto.TimeAvailable != null && dto.TimeAvailable !=
test.TimeAvailable)
        {
            test.TimeAvailable = dto.TimeAvailable;
        }
        if (dto.StartAt != null)
        {
            DateTime dateTime = new DateTime(1970, 1, 1, 0, 0, 0, 0,
DateTimeKind.Utc);
            dateTime                                                =
dateTime.AddMilliseconds(dto.StartAt.Value).ToLocalTime();

            test.StartAt = dateTime;
        }
        if (dto.EndAt != null)
```

```
            {
                DateTime dateTime = new DateTime(1970, 1, 1, 0, 0, 0, 0, DateTimeKind.Utc);
                dateTime = dateTime.AddMilliseconds(dto.EndAt.Value).ToLocalTime();

                test.EndAt = dateTime;
            }
            await using var session = _session.DocumentStore.LightweightSession();
            session.Store(test);

            await session.SaveChangesAsync();
            return Ok(test);
        }
    }
}
```

**SessionController.cs**

```
using EventStore.Client;
using Marten;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using System.Text.Json;
```

```
namespace TestsAPI.Controllers
{

    public class MultipleChoices
    {
        public List<string> Choices { get; set; }
    }

    public class SingularChoice
    {
        public string Choice { get; set; }
    }

    public class Match
    {
        public List<string> Keys { get; set; }
        public List<string> Values { get; set; }
    }

    public class StudentQuestion
    {
        public MultipleChoices? MultipleChoicesQuestion { get; set; }
        public SingularChoice? SingularChoiceQuestion { get; set; }
```

```
    public Match? MatchQuestion { get; set; }

    public string Title { get; set; }
}

public class WriteAnswerDTO
{
    public MultipleChoices? MultipleChoicesQuestion { get; set; }
    public SingularChoice? SingularChoiceQuestion { get; set; }
    public Match? MatchQuestion { get; set; }
    public string? Input { get; set; }
    public string UUID { get; set; }
}


public class OpenQuestion
{
    public string Title { get; set; }
    public string UUID { get; set; }

    public override string ToString()
    {
        return $"Question \"{Title}\" ({UUID}) opened";
    }
```

```
}

public class AnswerQuestion
{
    public MultipleChoices? MultipleChoicesQuestion { get; set; }
    public SingularChoice? SingularChoiceQuestion { get; set; }
    public Match? MatchQuestion { get; set; }
    public string? Input { get; set; }
    public string UUID { get; set; }

    public override string ToString()
    {
        return $"Question {UUID} answered";
    }
}

public class CountQuestions
{
    private List<Guid> IDs = new List<Guid>();
    public int Count { get; set; }

    public Guid Id { get; set; }

    public void Apply(OpenQuestion q) {
```

```
        var id = new Guid(q.UUID);
        if (IDs.Contains(id)) return;
        IDs.Add(id);
        Count = IDs.Count;
    }

    public override string ToString()
    {
        return Id.ToString();
    }
}


public class QuestionResult
{
    public MultipleChoices? MultipleChoicesQuestion { get; set; }
    public SingularChoice? SingularChoiceQuestion { get; set; }
    public Match? MatchQuestion { get; set; }
    public string? Input { get; set; }
    public string Title { get; set; }
    public string UUID { get; set; }
}

public class Answers
```

```
{
    public List<QuestionResult> Results = new List<QuestionResult>();
    public int Grade = 0;

    public Guid Id { get; set; }

    public void Apply(OpenQuestion q)
    {
        var qr = new QuestionResult();
        Results.Add(qr);
    }

    public override string ToString()
    {
        return Id.ToString();
    }
}

public class CreateSessionDTO
{
    public string Test { get; set; }
}

[Route("api/[controller]")]
```

```
[ApiController]
public class SessionController : ControllerBase
{
    private readonly IQuerySession _session;

    public SessionController(IQuerySession session)
    {
        _session = session;
    }

    [HttpGet("{token}")]
    public async Task<IActionResult> Get(String token)
    {
        var data = new URLSafeBase64Encoder().Decode(token);
        var parsed = Token.Parser.ParseFrom(data);
        var payload = parsed.Payload.Unpack<TestSessionPayload>();
        var test = await _session.LoadAsync<Test>(new Guid(payload.Test));
        if (!new TokenBuilder<TestSessionPayload, HMACSigner>(new
HMACSigner("test")).Verify(payload, parsed.Signature.ToByteArray()))
        {
            return Forbid();
        }
        var mt = new RandomNumberGenerator((ulong)payload.Seed +
1337133713371337);
        var cloned = test.Questions.Select(x => x).ToList();
```

```
            var counted =
_session.Events.AggregateStream<CountQuestions>(payload.UUID);
            var shuffled = Enumerable.Range(0, (counted is null ? 0 :
counted.Count) + 1).Select(x =>
            {
               var i = mt.Generate(0, cloned.Count - 1);
               var n = cloned[(int)i];
               cloned.RemoveAt((int)i);
               return n;
            }).ToList();
            return Ok(new
            {
               ID = test.Id.ToString(),
               Title = test.Title,
               Questions = Enumerable.Range(0, 1).Select(x =>
                {
                   var q = shuffled[x];
                   var sq = new StudentQuestion();
                   sq.Title = q.Title;
                   if (q.Kind == "SingularChoiceQuestion")
                   {
                      sq.SingularChoiceQuestion = new SingularChoice();
                      sq.SingularChoiceQuestion.Choice = q.Choices[0];
                   } else if (q.Kind == "MultipleChoicesQuestion")
                   {
```

```
                    sq.MultipleChoicesQuestion = new MultipleChoices();
                    sq.MultipleChoicesQuestion.Choices = q.Choices;
                }
                else if (q.Kind == "MatchQuestion")
                {
                    sq.MatchQuestion = new Match();
                    sq.MatchQuestion.Keys = q.PairKeys;
                    sq.MatchQuestion.Values = q.PairValues;
                }
                return sq;
            }).ToList()
        });
    }

    [HttpPut("{token}")]
    public async Task<IActionResult> Put([FromBody] WriteAnswerDTO dto, [FromRoute] string token)
    {
        var data = new URLSafeBase64Encoder().Decode(token);
        var parsed = Token.Parser.ParseFrom(data);
        var payload = parsed.Payload.Unpack<TestSessionPayload>();
        var test = await _session.LoadAsync<Test>(new Guid(payload.Test));
        if (!new TokenBuilder<TestSessionPayload, HMACSigner>(new HMACSigner("test")).Verify(payload, parsed.Signature.ToByteArray()))
        {
```

```
            return Forbid();
        }
        var mt = new RandomNumberGenerator((ulong)payload.Seed + 1337133713371337);
        var cloned = test.Questions.Select(x => x).ToList();
        var counted = _session.Events.AggregateStream<CountQuestions>(payload.UUID);
        var shuffled = Enumerable.Range(0, (counted is null ? 0 : counted.Count) + 1).Select(x =>
        {
            var i = mt.Generate(0, cloned.Count - 1);
            var n = cloned[(int)i];
            cloned.RemoveAt((int)i);
            return n;
        }).ToList();

        await using (var session = _session.DocumentStore.LightweightSession())
        {
            var evt = new AnswerQuestion();
            evt.UUID = dto.UUID;
            if (!(dto.MatchQuestion is null))
            {
                evt.MatchQuestion = dto.MatchQuestion;
            }
```

```
    if (!(dto.Input is null))
    {
        evt.Input = dto.Input;
    }
    if (!(dto.MultipleChoicesQuestion is null))
    {
        evt.MultipleChoicesQuestion = dto.MultipleChoicesQuestion;
    }
    if (!(dto.SingularChoiceQuestion is null))
    {
        evt.SingularChoiceQuestion = dto.SingularChoiceQuestion;
    }

    var open = new OpenQuestion();
    var last = shuffled[shuffled.Count - 1];
    open.UUID = last.Id.ToString();
    open.Title = last.Title;
    session.Events.Append(new Guid(payload.UUID), evt, open);
    await session.SaveChangesAsync();
}

return Ok(new
{
    ID = test.Id.ToString(),
```

```
Title = test.Title,
Questions = Enumerable.Range(0, 1).Select(x =>
{
    var q = shuffled[x];
    var sq = new StudentQuestion();
    sq.Title = q.Title;
    if (q.Kind == "SingularChoiceQuestion")
    {
        sq.SingularChoiceQuestion = new SingularChoice();
        sq.SingularChoiceQuestion.Choice = q.Choices[0];
    }
    else if (q.Kind == "MultipleChoicesQuestion")
    {
        sq.MultipleChoicesQuestion = new MultipleChoices();
        sq.MultipleChoicesQuestion.Choices = q.Choices;
    }
    else if (q.Kind == "MatchQuestion")
    {
        sq.MatchQuestion = new Match();
        sq.MatchQuestion.Keys = q.PairKeys;
        sq.MatchQuestion.Values = q.PairValues;
    }
    return sq;
}).ToList()
```

```
        });
    }

    [HttpPost("")]
    public async Task<IActionResult> Post([FromBody] CreateSessionDTO dto)
    {
        DateTime currentTime = DateTime.UtcNow;
        long unixTime = ((DateTimeOffset)currentTime).ToUnixTimeSeconds();
        var mt = new RandomNumberGenerator((ulong)unixTime);
        var id = Guid.NewGuid();
        await using (var session = _session.DocumentStore.LightweightSession())
        {
            session.Events.StartStream<Quest>(questId, started, joined1);
        }
        return Ok(new
        {
            Token = new URLSafeBase64Encoder().Encode(new TokenBuilder<TestSessionPayload, HMACSigner>(new HMACSigner("test"))
            .Build(new TestSessionPayload
            {
                Seed = (unixTime/2)+(long)mt.Generate(11111,99999),
                User = new Guid().ToString(),
```

```
                    UUID = id.ToString(),
                    Test = dto.Test
                }))
            });
        }

    }
}
```

**QuestionsController.cs**

```
using EventStore.Client;
using Marten;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace TestsAPI.Controllers
{
    public class MultipleChoicesQuestion
    {
        public List<string> Choices { get; set; }
        public List<int> CorrectIndices { get; set; }
    }
```

```
public class SingularChoiceQuestion
{
    public List<string> Choices { get; set; }
    public int CorrectIndex { get; set; }
}

public class MatchQuestion
{
    public List<string> ChoicesKeys { get; set; }
    public List<string> ChoicesValues { get; set; }
    public Dictionary<string, string> Correct { get; set; }
}

public class InputQuestion
{
    public string CorrectAnswer { get; set; }
}

public class CreateQuestionDTO
{
    public MultipleChoicesQuestion? MultipleChoicesQuestion { get; set; }
    public SingularChoiceQuestion? SingularChoiceQuestion { get; set; }
    public MatchQuestion? MatchQuestion { get; set; }
    public InputQuestion? InputQuestion { get; set; }
```

```
    public string Title { get; set; }
    public bool Automatic { get; set; }
    public int CorrectGrade { get; set; }
}

[Route("api/[controller]")]
[ApiController]
public class QuestionsController : ControllerBase
{
    private readonly IQuerySession _session;

    public QuestionsController(IQuerySession session)
    {
        _session = session;
    }

    [HttpPost("{id}")]
    public async Task<IActionResult> Post([FromRoute] string id,
[FromBody] CreateQuestionDTO dto)
    {
        var test = await _session.LoadAsync<Test>(new Guid(id));
        Question q = new();
        q.Id = Guid.NewGuid();
```

```
                q.Automatic = dto.Automatic;
                q.CorrectGrade = dto.CorrectGrade;
                q.Title = dto.Title;
                if (dto.InputQuestion != null)
                {
                    q.Kind = "InputQuestion";
                    q.Correct = new List<string> { dto.InputQuestion.CorrectAnswer
};
                }
                else if (dto.MultipleChoicesQuestion != null)
                {
                    q.Kind = "MultipleChoicesQuestion";
                    q.Correct = new List<string>();
                    for (int i = 0; i < dto.MultipleChoicesQuestion.Choices.Count; i++)
                    {
                        if (dto.MultipleChoicesQuestion.CorrectIndices.Contains(i))
                            q.Correct.Add(dto.MultipleChoicesQuestion.Choices[i]);
                    }
                    q.Choices = dto.MultipleChoicesQuestion.Choices;
                }
                else if (dto.SingularChoiceQuestion != null)
                {
                    q.Kind = "SingularChoiceQuestion";
```

```
                q.Correct = new List<string> {
dto.SingularChoiceQuestion.Choices[dto.SingularChoiceQuestion.CorrectIndex]
};
                q.Choices = dto.SingularChoiceQuestion.Choices;
            }
            else if (dto.MatchQuestion != null)
            {
                q.Kind = "MatchQuestion";
                q.PairAnswer = dto.MatchQuestion.Correct;
                q.PairKeys = dto.MatchQuestion.ChoicesKeys;
                q.PairValues = dto.MatchQuestion.ChoicesValues;
            }

            test.Questions.Add(q);
            await using var session =
_session.DocumentStore.LightweightSession();
            session.Store(test);
            await session.SaveChangesAsync();
            return NoContent();
        }


        [HttpPut("{id}")]
        public async Task<IActionResult> Put([FromRoute] string id,
[FromBody] List<CreateQuestionDTO> body)
```

```
        {
            var test = await _session.LoadAsync<Test>(new Guid(id));
            test.Questions = body.Select(dto =>
            {
                Question q = new();
                q.Id = Guid.NewGuid();
                q.Automatic = dto.Automatic;
                q.CorrectGrade = dto.CorrectGrade;
                q.Title = dto.Title;
                if (dto.InputQuestion != null)
                {
                    q.Kind = "InputQuestion";
                    q.Correct = new List<string> { dto.InputQuestion.CorrectAnswer };
                }
                else if (dto.MultipleChoicesQuestion != null)
                {
                    q.Kind = "MultipleChoicesQuestion";
                    q.Correct = new List<string>();
                    for (int i = 0; i < dto.MultipleChoicesQuestion.Choices.Count; i++)
                    {
                        if (dto.MultipleChoicesQuestion.CorrectIndices.Contains(i))
                            q.Correct.Add(dto.MultipleChoicesQuestion.Choices[i]);
                    }
```

```
                q.Choices = dto.MultipleChoicesQuestion.Choices;
            }
            else if (dto.SingularChoiceQuestion != null)
            {
                q.Kind = "SingularChoiceQuestion";
                q.Correct = new List<string> {
dto.SingularChoiceQuestion.Choices[dto.SingularChoiceQuestion.CorrectIndex]
};
                q.Choices = dto.SingularChoiceQuestion.Choices;
            }
            else if (dto.MatchQuestion != null)
            {
                q.Kind = "MatchQuestion";
                q.PairAnswer = dto.MatchQuestion.Correct;
                q.PairKeys = dto.MatchQuestion.ChoicesKeys;
                q.PairValues = dto.MatchQuestion.ChoicesValues;
            }

            return q;
        }).ToList();
        await using var session =
_session.DocumentStore.LightweightSession();
        session.Store(test);
        await session.SaveChangesAsync();
```

```
            return Ok(test);
        }
    }
}
```

**ДОДАТОК Б**

## Відгук керівника економічного розділу

**ДОДАТОК В**

## ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

| **Ім’я файла** | **Опис** |
|---|---|
| Пояснювальні документи | |
| Кваліфікаційна_робота.doc | Пояснювальна записка до кваліфікаційної роботи. Документ Word. |
| Кваліфікаційна_робота.pdf | Пояснювальна записка до кваліфікаційної роботи в форматі PDF |
| Програма | |
| Код.zip | Архів. Містить коди програми і откомпільовану програму |
| Презентація | |
| Презентація.ppt | Презентація кваліфікаційної роботи |