Київський національний університет імені Тараса Шевченка

ЗВІТ

до Етапу 3. Розробка веб-версії
практичного завдання (модуль 1)
з дисципліни "Інформаційні технології"
на тему:

Веб-версія системи управління табличними базами даних

Виконав
Студент четвертого курсу
Групи ТТП-41
Факультету комп'ютерних наук та кібернетики
Назарій ЯГОТІН

Київ, 2025

Зміст

1 Постановка задачі

1.1 Мета роботи

Розробити веб-версію системи управління табличними базами даних, переносячи функціонал десктоп-застосунку на архітектуру клієнт-сервер з підтримкою багатьох одночасних користувачів.

1.2 Основні вимоги

Система повинна забезпечувати:

- REST API з ASP.NET Core для управління базами даних
- Фронтенд на React + TypeScript з інтерфейсом, подібним до десктопної версії
- Підтримку сесій для ізоляції даних різних користувачів
- Всі функціональні можливості десктоп-версії: CRUD операції, валідацію, зберігання
- Адаптацію для обмежень браузера (завантаження файлів замість збереження на локальне місце)
- SPA (Single Page Application) архітектуру без маршрутизації

2 Архітектура системи

2.1 Компоненти системи

Веб-версія складається з трьох основних частин:

- **Backend (ASP.NET Core Web API)** — обробка запитів, управління сесіями, робота з даними
- **Frontend (React + TypeScript)** — інтерактивний користувацький інтерфейс
- **CoreLib (C# Business Logic)** — спільна бібліотека класів для обох версій

2.2 Управління сесіями

Ключова особливість веб-версії — підтримка багатьох користувачів з ізольованими екземплярами DatabaseService. Реалізується через `ISessionService`:

```
public interface ISessionService
{
    string GetSessionId();
    T GetOrCreateService<T>(Func<T> factory) where T : class;
}

public class SessionService : ISessionService
{
    private readonly IMemoryCache _cache;
    private const string SESSION_ID_HEADER = "X-Session-Id";
    private const int SESSION_TIMEOUT_MINUTES = 120;

    public string GetSessionId()
    {
        // Спробуйте отримати з заголовку або cookies
        // Створіть новий sessionId, якщо не існує
    }

    public T GetOrCreateService<T>(Func<T> factory) where T : class
    {
        var sessionId = GetSessionId();
        var cacheKey = $"service_{typeof(T).Name}_{sessionId}";

        if (!_cache.TryGetValue(cacheKey, out T service))
```

```
        {
            service = factory();
            var cacheOptions = new MemoryCacheEntryOptions()
                .SetSlidingExpiration(TimeSpan.FromMinutes(SESSION_TIMEOUT_MINUTES));
            _cache.Set(cacheKey, service, cacheOptions);
        }
        return service;
    }
}
```

Сесія ідентифікується унікальним GUID, який:

- Передається клієнтом у заголовку `X-Session-Id`
- Або зберігається у cookies браузера
- Автоматично генерується, якщо не існує
- Активна протягом 120 хвилин (sliding window)

2.3 Базова архітектура контролерів

Всі контролери успадковують `BaseApiController`:

```
public abstract class BaseApiController : ControllerBase
{
    protected readonly ISessionService SessionService;
    protected DatabaseService _databaseService;
    protected TableService _tableService;

    protected DatabaseService DatabaseService =>
        _databaseService ??= SessionService.GetOrCreateService(() =>
            CreateDatabaseService());

    protected TableService TableService =>
        _tableService ??= SessionService.GetOrCreateService(() =>
            new TableService(DatabaseService));
}
```

Це забезпечує автоматичний доступ до сервісів для кожної поточної сесії.

3 Використані технології

3.1 Backend

- **ASP.NET Core** — веб-фреймворк
- **C# 14.0** — мова програмування
- **Memory Cache** — зберігання сесій та сервісів
- **CORS** — для комунікації з фронтенду

3.2 Frontend

- **React 19** — бібліотека для UI
- **TypeScript** — статична типізація
- **Vite** — білд-інструмент
- **Tailwind CSS** — стилізація
- **Fetch API** — комунікація з сервером

3.3 Спільна бібліотека

- **CoreLib** — та сама, що й у десктоп-версії
- **System.Text.Json** — серіалізація

4 REST API

4.1 Перегляд базових операцій

API містить наступні endpoints у контролерах `DatabaseController` та `TableController`:

Метод	Route	Опис
POST	/api/database/create	Створити нову БД
POST	/api/database/load	Завантажити БД з файлу
POST	/api/database/save	Зберегти БД на диск
POST	/api/database/download	Експортувати БД у браузер
GET	/api/database/info	Інформація про поточну БД
GET	/api/database/tables	Список таблиць
GET	/api/database/statistics	Статистика БД
GET	/api/database/validate	Валідація структури
POST	/api/database/close	Закрити поточну БД
GET	/api/table/{tableName}	Отримати інформацію про таблицю
DELETE	/api/table/{tableName}	Видалити таблицю
POST	/api/table/create	Створити нову таблицю
GET	/api/table/{tableName}/columns	Список колонок таблиці
PUT	/api/table/{tableName}/columns/rename	Перейменувати колонку
PUT	/api/table/{tableName}/columns/reorder	Змінити порядок колонок
GET	/api/table/{tableName}/rows	Отримати всі рядки
POST	/api/table/{tableName}/rows	Додати новий рядок
GET	/api/table/{tableName}/rows/{rowIndex}	Отримати конкретний рядок
PUT	/api/table/{tableName}/rows/{rowIndex}	Редагувати рядок
DELETE	/api/table/{tableName}/rows/{rowIndex}	Видалити рядок

Вищенаведені операції забезпечуються за рахунок ізольованих екземплярів DatabaseService та TableService.

4.2 Приклад: Завантаження файлу

```
[HttpPost("load")]
[RequestSizeLimit(52428800)] // 50 MB
public async Task<IActionResult> LoadDatabase(IFormFile file)
{
    if (file == null || file.Length == 0)
        return BadRequest(new { error = "Database file is required" });

```

```
    var tempPath = Path.Combine(Path.GetTempPath(),
    ↪ $"{Guid.NewGuid()}_{file.FileName}");

    using (var stream = new FileStream(tempPath, FileMode.Create))
    {
        await file.CopyToAsync(stream);
    }

    try
    {
        var database = await DatabaseService.LoadDatabaseAsync(tempPath);
        return Ok(new {
            message = "Database loaded successfully",
            name = database.Name,
            tableCount = database.Tables.Count
        });
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error loading database");
        return StatusCode(500, new { error = "Internal server error" });
    }
}
```

4.3 Обмеження на розмір файлів

У `Program.cs` налаштовано підтримку файлів до 50 МБ:

```
builder.Services.Configure<FormOptions>(options =>
{
    options.MultipartBodyLengthLimit = 52428800; // 50 MB
});

builder.WebHost.ConfigureKestrel(options =>
{
    options.Limits.MaxRequestBodySize = 52428800; // 50 MB
});
```

5 Frontend реалізація

5.1 Структура проєкту React

- `src/` — основна директорія
  - `api/` — API клієнти для комунікації з сервером
  - `components/` — React компоненти
  - `pages/` — сторінки (хоча маршрутизації немає)
  - `hooks/` — custom React hooks
  - `types/` — TypeScript типи та інтерфейси
  - `App.tsx` — головна компонента

5.2 Сесії на клієнті

У користувацькій частині під час надсилання запитів до серверної частини, унікальний ідентифікатор сесії дістається з `localStorage`:

```
// src/api/types.ts
export const API_BASE_URL = import.meta.env.VITE_API_URL ||
    'http://localhost:5000/api';

export function getSessionId(): string {
    let sessionId = localStorage.getItem('sessionId');
    if (!sessionId) {
        sessionId = generateUUID();
        localStorage.setItem('sessionId', sessionId);
    }
    return sessionId;
}

export function getHeaders(): Record<string, string> {
    return {
        'Content-Type': 'application/json',
        'X-Session-Id': getSessionId()
    };
}
```

Сесія передається у всіх запитах через заголовок `X-Session-Id`.

5.3 API клієнти

Для кожного ресурсу існує відповідний API клієнт:

```typescript
// src/api/databaseApi.ts
export class DatabaseApi {
  static async create(name: string): Promise<ApiResponse> {
    const response = await fetch(`${API_BASE_URL}/database/create`, {
      method: 'POST',
      headers: getHeaders(),
      body: JSON.stringify({ name })
    });
    return response.json();
  }

  static async download(): Promise<Response> {
    const response = await fetch(`${API_BASE_URL}/database/download`, {
      method: 'POST',
      headers: {
        'X-Session-Id': getSessionId(),
      },
    });

    if (!response.ok) {
      if (response.headers.get('content-type')?.includes('application/json')) {
        const error = await response.json();
        throw new Error(error.error || 'Failed to download');
      }
      throw new Error('Failed to download database');
    }

    return response;
  }
}
```

6 Користувацький інтерфейс

6.1 Головне вікно

Інтерфейс побудовано як SPA без маршрутизації. Головна сторінка містить:

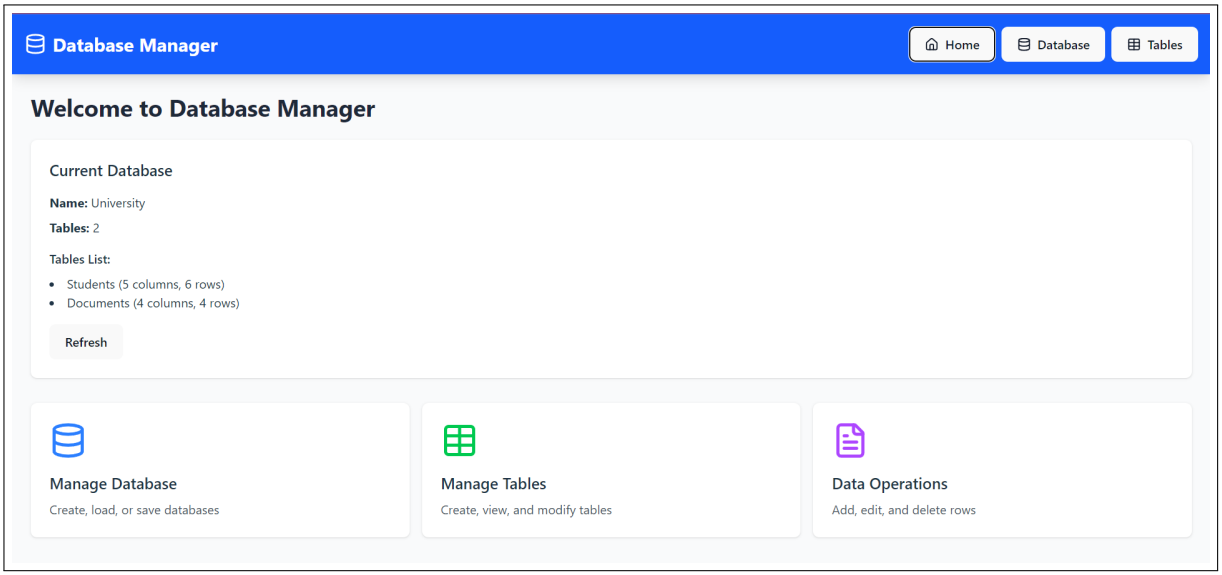

Рис. 1: Головне вікно веб-застосунку

- Навігаційну панель з посиланнями на інші сторінки
- Інформацію про поточну БД та список її таблиць
- Індикатор того, чи доступна БД для роботи

6.2 Маніпуляції з БД

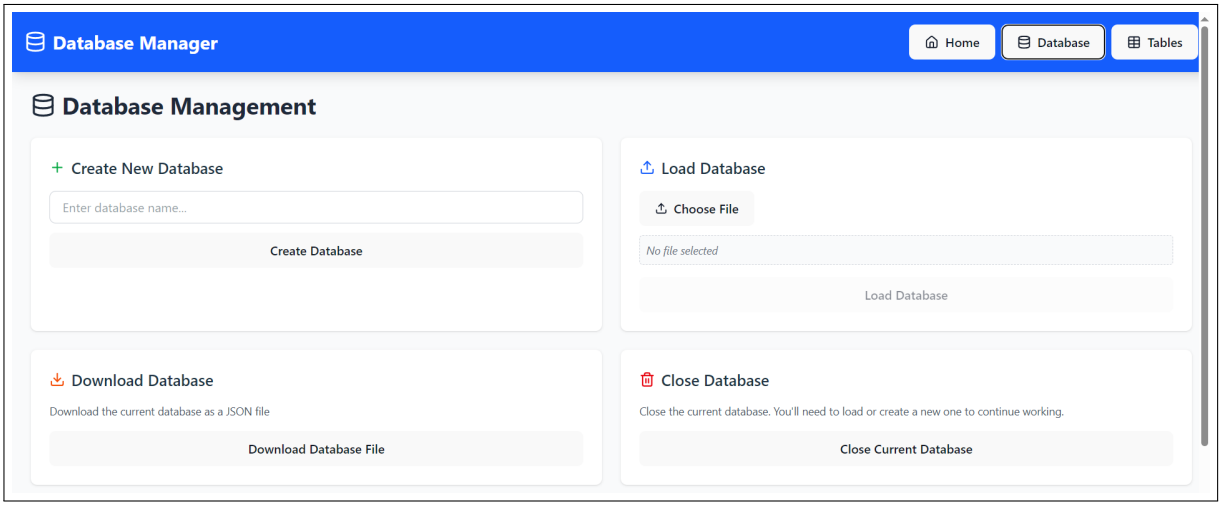

Рис. 2: Сторінка для взаємодії з БД

Друга сторінка містить весь функціонал необхідний для взаємодії з БД на найвищому рівні абстракції.

- Створення нової БД
- Завантаження БД з файлу
- Зберігання БД у .json - файл
- Закриття та очищення БД

6.3 Робота з таблицями

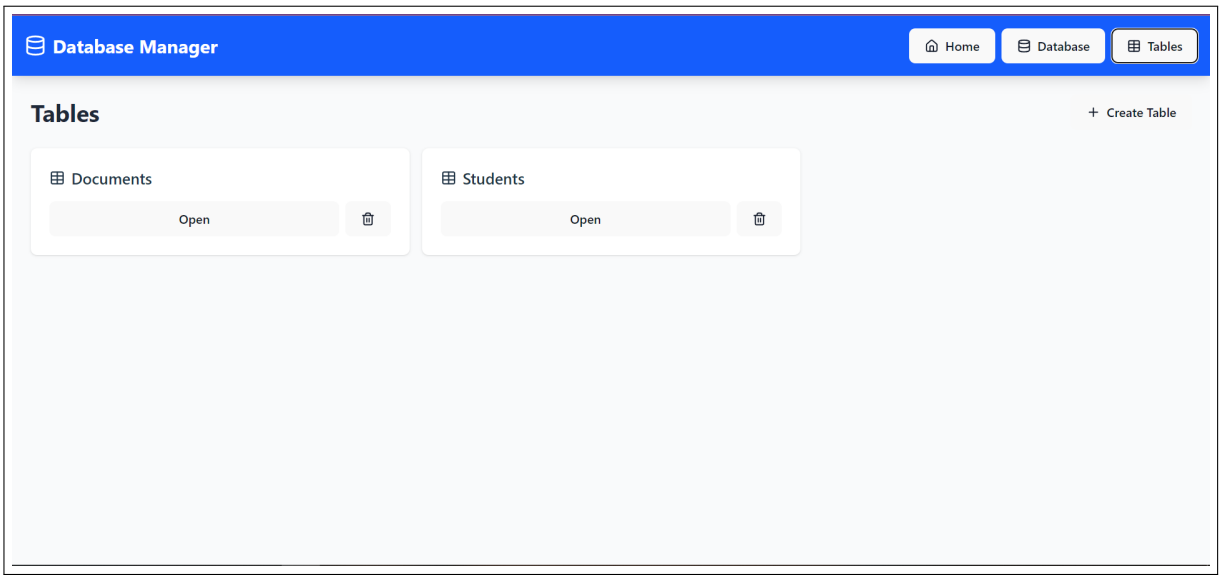

Рис. 3: Перегляд списку таблиць

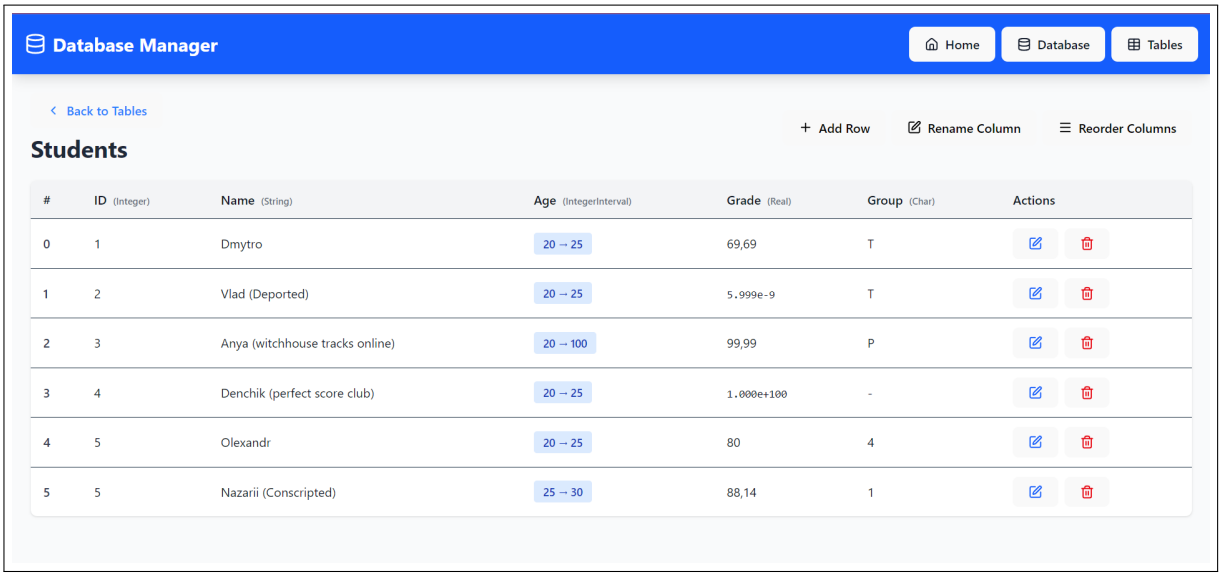

Рис. 4: Перегляд даних таблиці

6.4 CRUD операції

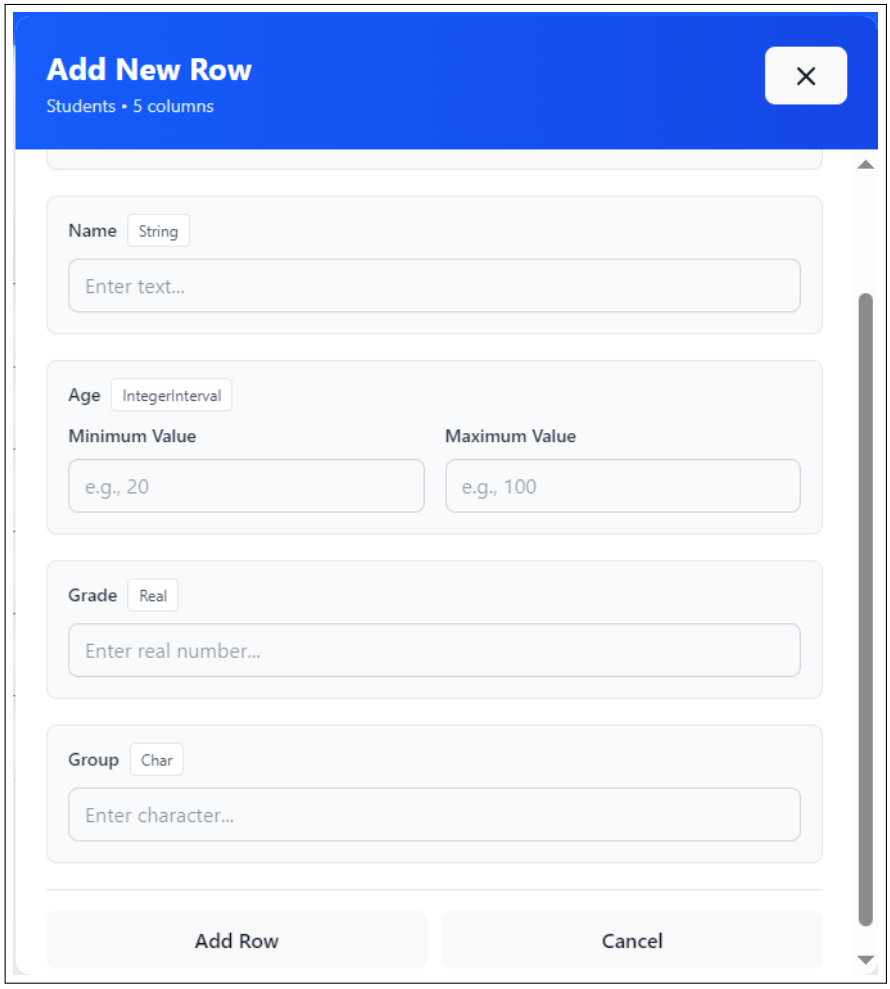

Рис. 5: Додавання нового рядка

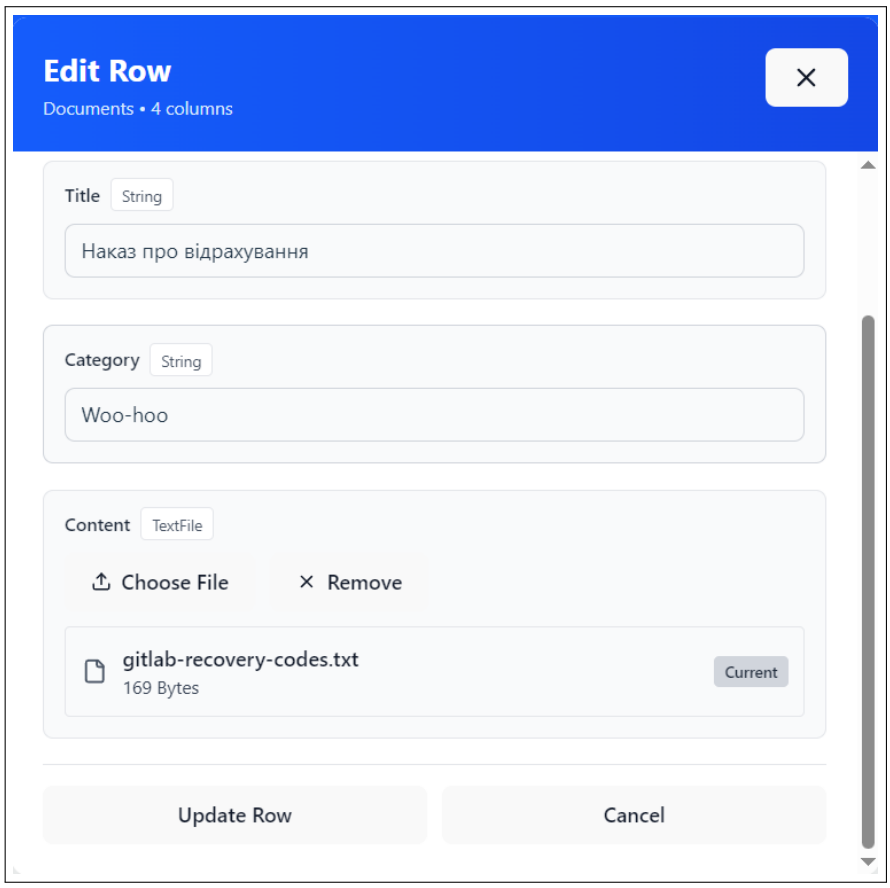

Рис. 6: Редагування рядка

6.5 Завантаження файлів

На відміну від десктоп-версії, збереження файлів на локальний диск браузером заборонено з причин безпеки. Замість цього реалізовано:

- **Завантаження файлу на сервер** через діалог вибору файлу
- **Збереження посилання** на файл у таблиці
- **Завантаження файлу** на пристрій за потреби
- **Завантаження БД у браузер** як JSON-файл

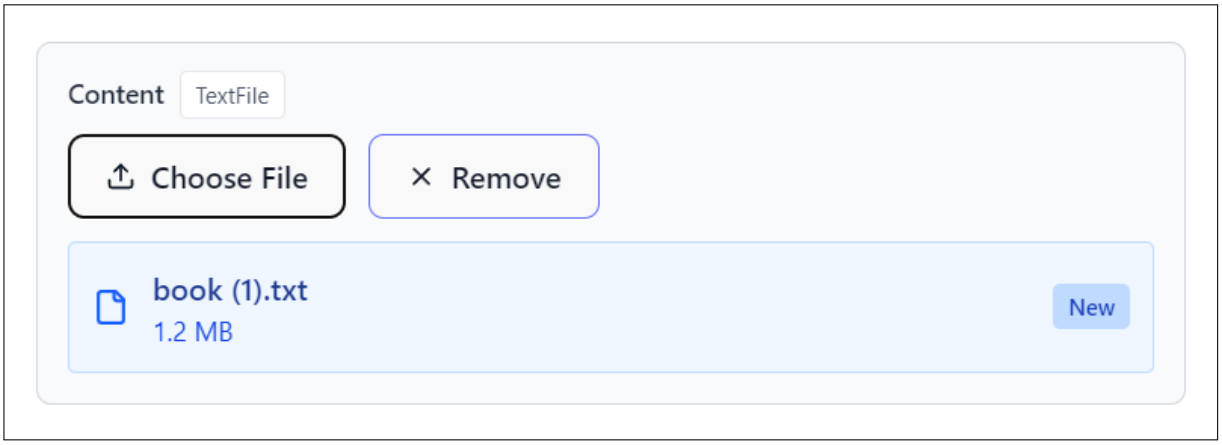

Рис. 7: Завантаження текстового файлу

6.6 Експорт БД

Експорт БД здійснюється як завантаження JSON-файлу через браузер:

```
static async handleDownload() {
  try {
    const response = await DatabaseApi.download();
    const blob = await response.blob();
    const url = window.URL.createObjectURL(blob);
    const link = document.createElement('a');
    link.href = url;
    link.download = `database_${new Date().getTime()}.json`;
    link.click();
  } catch (error) {
    console.error('Download failed:', error);
  }
}
```

7 Індивідуальні типи даних

7.1 Інтервальний тип цілих чисел

При редагуванні рядка для колонок типу `IntegerInterval` виводяться два поля (мін та макс):

Рис. 8: Введення інтервального типу

7.2 Текстові файли

Файли зберігаються на сервері з посиланнями у таблиці. Користувач може завантажити файл, натиснувши на відповідну іконку поруч з іменем файлу.

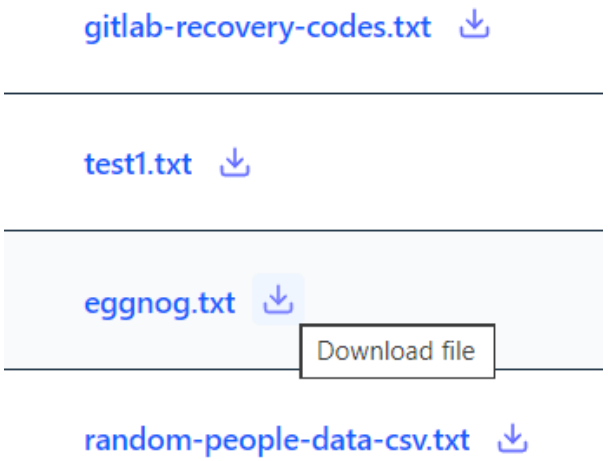

Рис. 9: Відображення файлів у таблиці

8 Індивідуальні операції

8.1 Перейменування колонок таблиці

Перейменування колонки відбувається на сторінці відповідної таблиці через випадаючий список та подальше підтвердження

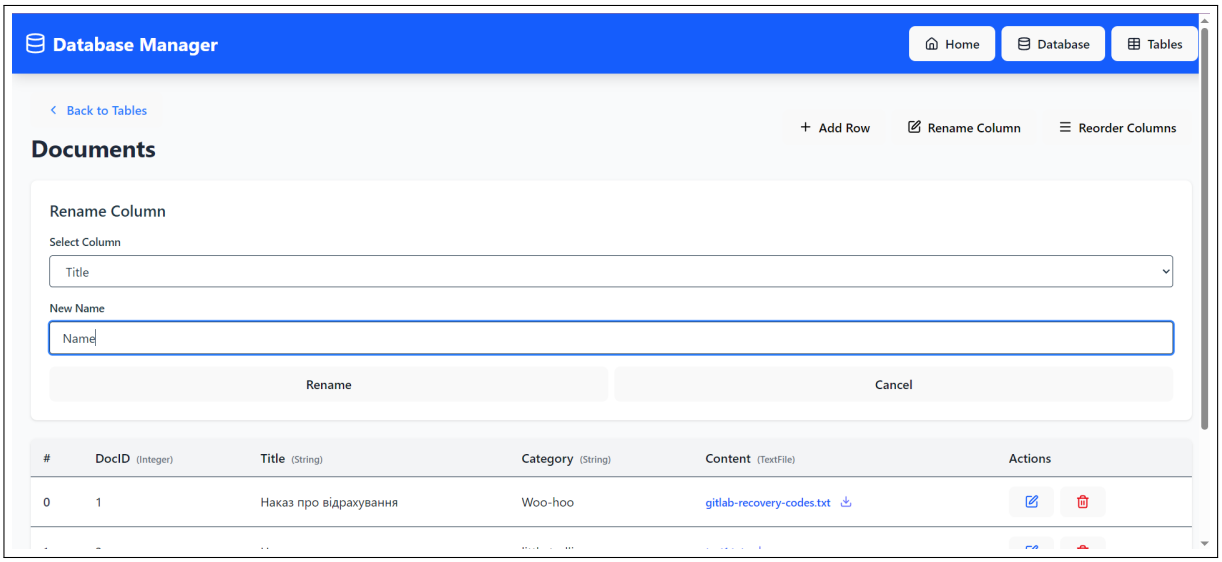

Рис. 10: Розділ для перейменування

8.2 Перестановка колонок таблиці

Перестановка колонок відбувається схожим чином. Інтерфейс реалізовано за допомогою drag-and-drop карток

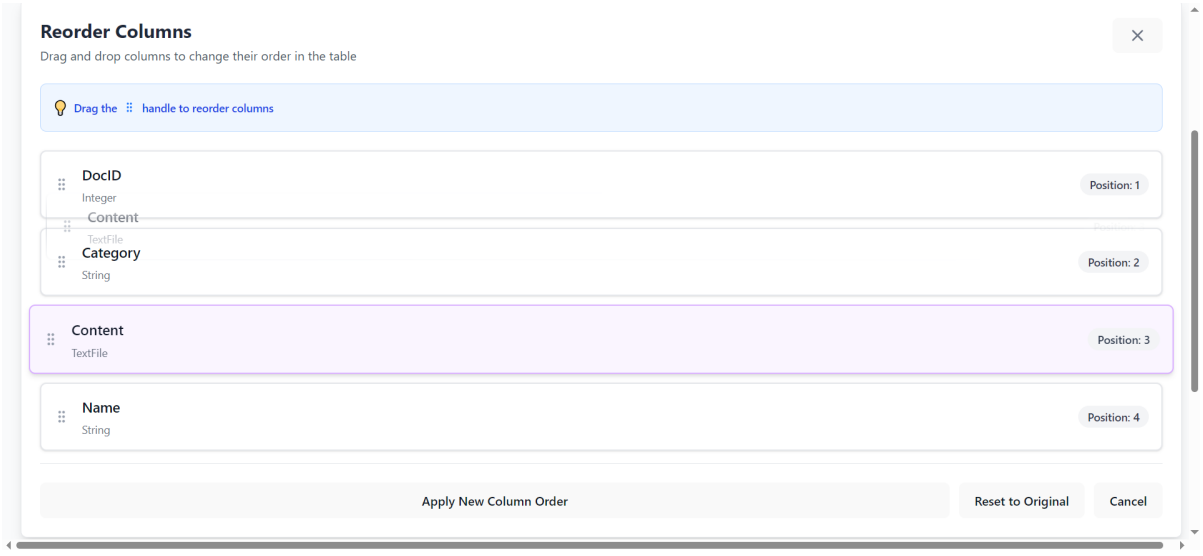

Рис. 11: Розділ для перестановки

9 Обробка помилок

9.1 Валідація даних

Обробка помилок відбувається переважно в самій бібліотеці при обробці значень, отриманих від користувача

Рис. 12: Повідомлення про помилку валідації

9.2 Обробка помилок сервера

API контролери повертають стандартизовані відповіді:

```
if (string.IsNullOrWhiteSpace(request.Name))
    return BadRequest(new { error = "Database name cannot be empty" });

catch (ArgumentException ex)
{
    return BadRequest(new { error = ex.Message });
}
catch (Exception ex)
{
    _logger.LogError(ex, "Error creating database");
    return StatusCode(500, new { error = "Internal server error" });
}
```

10 Розділення даних між користувачами

Система забезпечує повну ізоляцію даних різних користувачів через механізм сесій:

1. Кожна сесія отримує унікальний GUID (sessionId)
2. Для кожної сесії створюється окремий екземпляр `DatabaseService`
3. Екземпляр кешується у `MemoryCache` з таймаутом 120 хвилин
4. При завершенні таймаута автоматично видаляються всі файли та дані сесії
5. Два користувачі ніколи не можуть отримати доступ до даних один одного

11 Деплоймент та конфігурація

11.1 Запуск застосунку

Backend запускається як ASP.NET Core сервіс:

```
# Backend
dotnet run

# Frontend (у іншому терміналі)
npx vite
```

11.2 CORS налаштування

У `Program.cs` дозволено всі запити з фронтенду:

```
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowAll", policy =>
    {
        policy.AllowAnyOrigin()
              .AllowAnyMethod()
              .AllowAnyHeader();
    });
});

app.UseCors("AllowAll");
```

12 Висновок

12.1 Досягнуті цілі

У рамках етапу 3 успішно реалізовано:

1. **REST API** на ASP.NET Core з 20-ма точками доступу для управління БД
2. **Систему управління сесіями** для ізоляції даних користувачів
3. **React фронтенд** з інтерфейсом, подібним до десктоп-версії
4. **SPA архітектуру** без маршрутизації
5. **Всі функціональні можливості** десктоп-версії
6. **Підтримку багатьох одночасних користувачів**
7. **Обробку помилок** і валідацію на різних етапах

12.2 Архітектурні переваги веб-версії

- **Масштабованість:** можна запустити на сервері для багатьох користувачів
- **Доступність:** потребує лише веб-браузера
- **Безпека:** сесії автоматично закінчуються
- **Перевикористання коду:** спільна CoreLib для обох версій
- **Розділення обов'язків:** чіткий API контракт між користувацькою та серверною частиною

Додаток А. Посилання на репозиторій

Повний код проєкту доступний на GitHub: https://github.com/8ctag8ne/DBill