Міністерство освіти і науки України
Національний технічний університет
України
“Київський політехнічний інститут імені Ігоря
Сікорського” Факультет інформатики та обчислювальної
техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №9

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Взаємодія компонентів системи»

Виконав:
студент групи ІА-з31
Курбатов Кіріл Андрійович

Перевірив:
Амонс Олександр Анатолійович

Київ 2025

Мета: Вивчити види взаємодії додатків (Client-Server, Peer-to-Peer, Service-oriented Architecture), та реалізувати в проєктованій системі одну із архітектур.

Завдання

• Ознайомитись з короткими теоретичними відомостями.
• Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
• Реалізувати функціонал для роботи в розподіленому оточенні відповідно до обраної теми.
• Реалізувати взаємодію розподілених частин:
– Для клієнт-серверних варіантів: реалізація клієнтської і серверної частини додатків, а також загальної частини (middleware); зв'язок клієнтської і серверної частин за допомогою WCF, TcpClient, .NET-Remoting на розсуд виконавця.
– Для однорангових мереж: реалізація взаємодії клієнтських додатків за допомогою WCF Peer to peer channel.
– Для SOA додатків: реалізація сервісу, що надає послуги клієнтським застосуванням; викладання сервісу в хмару або підняття у вигляді Web Service на локальній машині; використання токенів для передачі даних про автентифікації, двостороннє шифрування.
• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє спроєктовану архітектуру. Навести фрагменти програмного коду, які є суттєвими для відображення реалізованої архітектури.

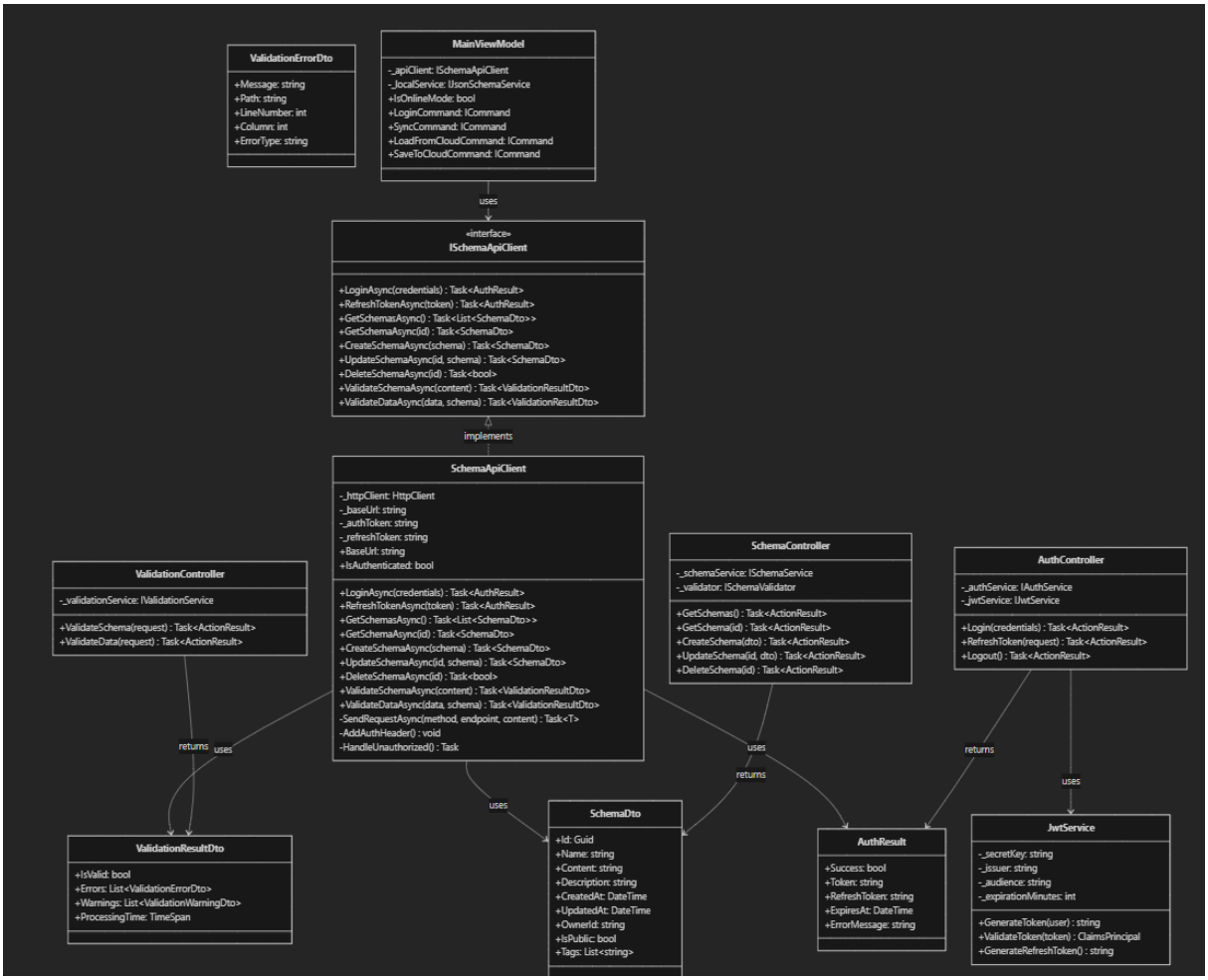


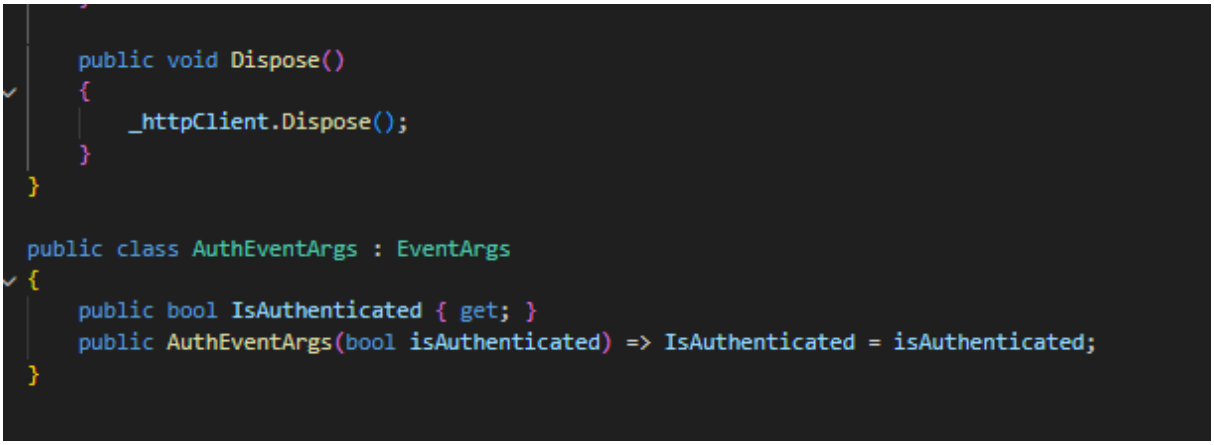
Рис. 1 — Діаграма класів SOA архітектури JSON Schema Tool

```
using System.Net.Http.Headers;
using System.Net.Http.Json;
using System.Text;
using System.Text.Json;

namespace JsonTool.Client;

public interface ISchemaApiClient
{
    bool IsAuthenticated { get; }
    Task<AuthResult> LoginAsync(string username, string password);
    Task<AuthResult> RefreshTokenAsync();
    Task<List<SchemaDto>> GetSchemasAsync();
    Task<SchemaDto?> GetSchemaAsync(Guid id);
    Task<SchemaDto> CreateSchemaAsync(CreateSchemaRequest request);
    Task<SchemaDto> UpdateSchemaAsync(Guid id, UpdateSchemaRequest request);
    Task<bool> DeleteSchemaAsync(Guid id);
    Task<ValidationResultDto> ValidateSchemaAsync(string content);
    Task<ValidationResultDto> ValidateDataAsync(string data, string schema);
}

public class SchemaApiClient : ISchemaApiClient, IDisposable
{
    private readonly HttpClient _httpClient;
    private readonly JsonSerializerOptions _jsonOptions;
    private string? _authToken;
    private string? _refreshToken;
    private DateTime _tokenExpiry;

    public string BaseUrl { get; }
    public bool IsAuthenticated => !string.IsNullOrEmpty(_authToken) &&
                                   DateTime.UtcNow < _tokenExpiry;

    public event EventHandler<AuthEventArgs>? AuthenticationChanged;

    public SchemaApiClient(string baseUrl)
    {
        BaseUrl = baseUrl.TrimEnd('/');
        _httpClient = new HttpClient
        {
            BaseAddress = new Uri(BaseUrl),
            Timeout = TimeSpan.FromSeconds(30)
        };

        _jsonOptions = new JsonSerializerOptions
        {
            PropertyNamingPolicy = JsonNamingPolicy.CamelCase,
            PropertyNameCaseInsensitive = true
        };
    }
```

```
public async Task<AuthResult> LoginAsync(string username, string password)
{
    var request = new { Username = username, Password = password };
    var response = await _httpClient.PostAsJsonAsync("/api/auth/login", request);

    var result = await response.Content.ReadFromJsonAsync<AuthResult>(_jsonOptions);

    if (result?.Success == true)
    {
        _authToken = result.Token;
        _refreshToken = result.RefreshToken;
        _tokenExpiry = result.ExpiresAt ?? DateTime.UtcNow.AddMinutes(60);

        AuthenticationChanged?.Invoke(this, new AuthEventArgs(true));
    }

    return result ?? new AuthResult { Success = false, ErrorMessage = "Unknown error" };
}

public async Task<AuthResult> RefreshTokenAsync()
{
    if (string.IsNullOrEmpty(_refreshToken))
    {
        return new AuthResult { Success = false, ErrorMessage = "No refresh token" };
    }

    var request = new { Token = _authToken, RefreshToken = _refreshToken };
    var response = await _httpClient.PostAsJsonAsync("/api/auth/refresh", request);

    var result = await response.Content.ReadFromJsonAsync<AuthResult>(_jsonOptions);

    if (result?.Success == true)
    {
        _authToken = result.Token;
        _refreshToken = result.RefreshToken;
        _tokenExpiry = result.ExpiresAt ?? DateTime.UtcNow.AddMinutes(60);
    }
    else
    {
        _authToken = null;
        _refreshToken = null;
        AuthenticationChanged?.Invoke(this, new AuthEventArgs(false));
    }

    return result ?? new AuthResult { Success = false, ErrorMessage = "Unknown error" };
}

public void Logout()
{
    _authToken = null;
    _refreshToken = null;
    AuthenticationChanged?.Invoke(this, new AuthEventArgs(false));
}
```

```
public async Task<List<SchemaDto>> GetSchemasAsync()
{
    await EnsureAuthenticatedAsync();
    AddAuthHeader();

    var response = await _httpClient.GetAsync("/api/schemas");
    response.EnsureSuccessStatusCode();

    return await response.Content.ReadFromJsonAsync<List<SchemaDto>>(_jsonOptions)
        ?? new List<SchemaDto>();
}

public async Task<SchemaDto?> GetSchemaAsync(Guid id)
{
    await EnsureAuthenticatedAsync();
    AddAuthHeader();

    var response = await _httpClient.GetAsync($"/api/schemas/{id}");

    if (response.StatusCode == System.Net.HttpStatusCode.NotFound)
        return null;

    response.EnsureSuccessStatusCode();
    return await response.Content.ReadFromJsonAsync<SchemaDto>(_jsonOptions);
}

public async Task<SchemaDto> CreateSchemaAsync(CreateSchemaRequest request)
{
    await EnsureAuthenticatedAsync();
    AddAuthHeader();

    var response = await _httpClient.PostAsJsonAsync("/api/schemas", request);
    response.EnsureSuccessStatusCode();

    return await response.Content.ReadFromJsonAsync<SchemaDto>(_jsonOptions)
        ?? throw new InvalidOperationException("Failed to create schema");
}

public async Task<SchemaDto> UpdateSchemaAsync(Guid id, UpdateSchemaRequest request)
{
    await EnsureAuthenticatedAsync();
    AddAuthHeader();

    var response = await _httpClient.PutAsJsonAsync($"/api/schemas/{id}", request);
    response.EnsureSuccessStatusCode();

    return await response.Content.ReadFromJsonAsync<SchemaDto>(_jsonOptions)
        ?? throw new InvalidOperationException("Failed to update schema");
}
```

```
public async Task<bool> DeleteSchemaAsync(Guid id)
{
    await EnsureAuthenticatedAsync();
    AddAuthHeader();

    var response = await _httpClient.DeleteAsync($"/api/schemas/{id}");
    return response.IsSuccessStatusCode;
}

public async Task<ValidationResultDto> ValidateSchemaAsync(string content)
{
    await EnsureAuthenticatedAsync();
    AddAuthHeader();

    var request = new { Content = content };
    var response = await _httpClient.PostAsJsonAsync("/api/validation/schema", request);
    response.EnsureSuccessStatusCode();

    return await response.Content.ReadFromJsonAsync<ValidationResultDto>(_jsonOptions)
        ?? new ValidationResultDto { IsValid = false };
}

public async Task<ValidationResultDto> ValidateDataAsync(string data, string schema)
{
    await EnsureAuthenticatedAsync();
    AddAuthHeader();

    var request = new { Data = data, Schema = schema };
    var response = await _httpClient.PostAsJsonAsync("/api/validation/data", request);
    response.EnsureSuccessStatusCode();

    return await response.Content.ReadFromJsonAsync<ValidationResultDto>(_jsonOptions)
        ?? new ValidationResultDto { IsValid = false };
}

private void AddAuthHeader()
{
    _httpClient.DefaultRequestHeaders.Authorization =
        new AuthenticationHeaderValue("Bearer", _authToken);
}

private async Task EnsureAuthenticatedAsync()
{
    if (string.IsNullOrEmpty(_authToken))
    {
        throw new UnauthorizedAccessException("Not authenticated");
    }

    if (DateTime.UtcNow.AddMinutes(5) >= _tokenExpiry)
    {
        var result = await RefreshTokenAsync();
        if (!result.Success)
        {
            throw new UnauthorizedAccessException("Token refresh failed");
        }
    }
}
```



```
    public void Dispose()
    {
        _httpClient.Dispose();
    }
}

public class AuthEventArgs : EventArgs
{
    public bool IsAuthenticated { get; }
    public AuthEventArgs(bool isAuthenticated) => IsAuthenticated = isAuthenticated;
}
```