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 розробки програмного забезпечення>>

Тема РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE

Виконав:
студент ІА-23
Содолиснький Вадим

Перевірив:
Мягкий М. Ю.

Київ 2024

Тема: РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE

Хід роботи

Варіант №13

Office communicator

Мережевий комунікатор для офісу повинен нагадувати функціонал програми Skype з можливостями голосового / відео / конференц-зв'язку, відправки текстових повідомлень і файлів (можливо, оффлайн), веденням організованого списку груп / контактів.

Короткий опис способів взаємодії

Архітектура Client-Server

У моделі Client-Server додаток розділений на дві основні частини: клієнт і сервер. Клієнт відповідає за надсилання запитів і надання користувачеві інтерфейсу, а сервер обробляє ці запити, забезпечує зберігання даних і реалізацію бізнес-логіки. Така модель широко використовується у веб-додатках, електронній пошті та системах управління базами даних. Вона забезпечує централізоване зберігання даних і простоту адміністрування, але залежить від доступності сервера, що може стати єдиною точкою відмови.

Також є поділ клієнтів на дві частини. Товстий клієнт — це клієнт, у якому значна частина обчислень і функціональності виконується на стороні клієнта. Сервер при цьому зазвичай відповідає лише за зберігання даних або мінімальну обробку запитів. Тонкий клієнт — це додаток, у якому більшість обчислень, логіки та обробки даних виконується на стороні сервера, тоді як клієнтська частина здебільшого займається відображенням інформації та відправкою запитів.

Архітектура Peer-To-Peer

У моделі Peer-to-Peer (P2P) всі вузли є рівноправними. Кожен вузол може виконувати роль і клієнта, і сервера, що дозволяє здійснювати обмін даними безпосередньо між учасниками мережі. Ця модель часто використовується в децентралізованих системах, таких як торрент-клієнти, блокчейни та IP-телефонія. P2P забезпечує високу стійкість до відмов, оскільки відсутній центральний сервер, але має труднощі з безпекою та синхронізацією даних.

Архітектура SOA

Service-Oriented Architecture (SOA) побудована на основі сервісів, які виконують окремі функції та взаємодіють через стандартизовані протоколи, такі як HTTP, REST чи SOAP. Ця архітектура забезпечує високу гнучкість і масштабованість, дозволяючи інтегрувати різні системи та використовувати сервіси повторно в різних додатках. SOA часто використовується в мікросервісах, хмарних платформах і корпоративних системах. Однак така модель може ускладнювати тестування та управління, а також збільшувати затримки в обміні даними між сервісами.

Архітектура Client-Server

Клієнт-Сервер — це архітектурний патерн, у якому система розділена на дві основні частини: клієнт (client) і сервер (server). Клієнт ініціює запити, а сервер обробляє ці запити та відповідає результатами.

Переваги

Сервер є централізованим вузлом, що дозволяє легко контролювати дані, логіку та функціональність системи та оновлення сервера дозволяє автоматично оновити функціональність для всіх клієнтів без втручання в їх роботу.

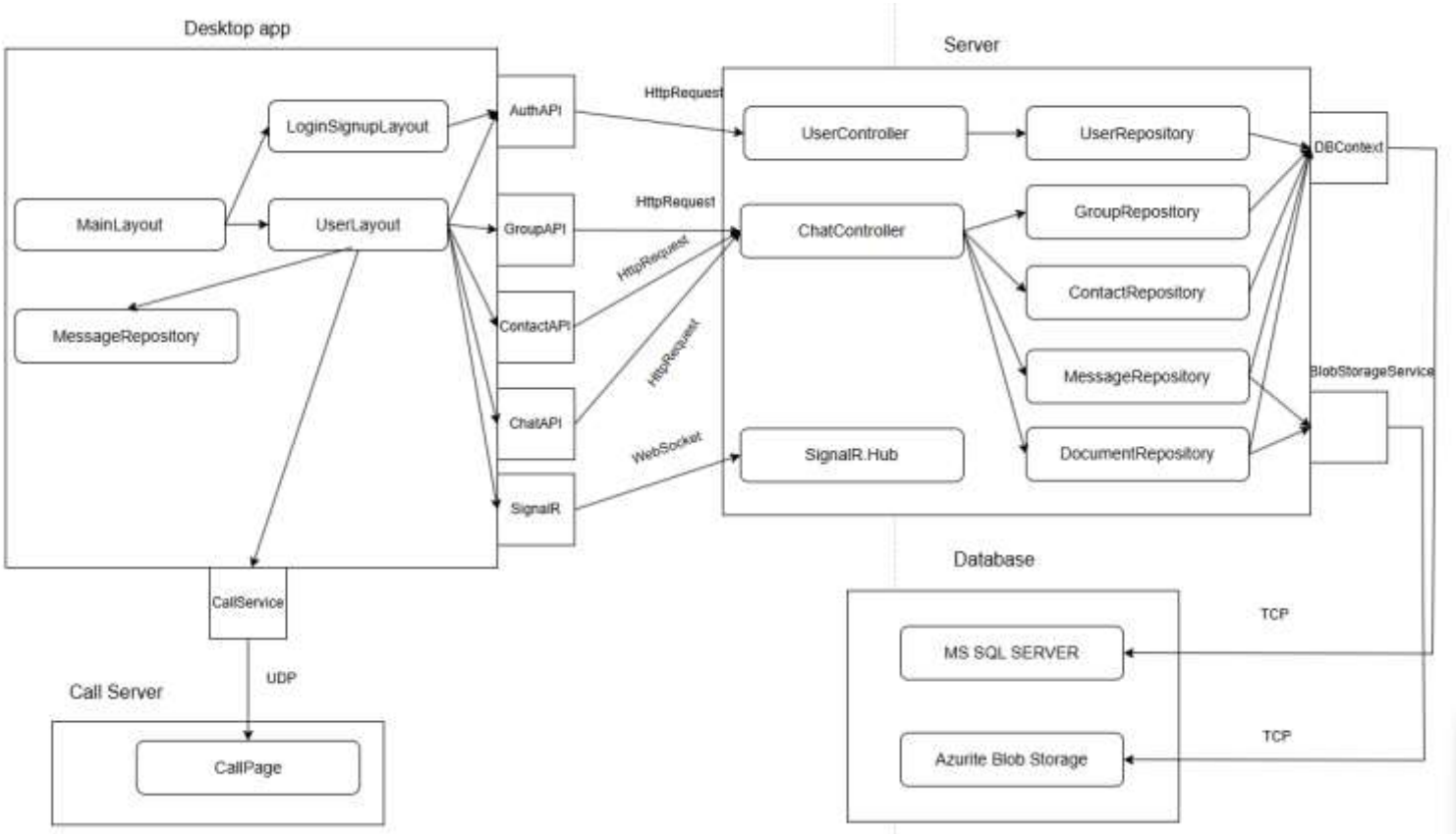

Рис. 1 – Діаграма реалізації застосунку

Застосунок поділений на дві основні частини: клієнт написаний на MAUI та сервер написаний на ASP.NET. Клієнт надсилає запити на сервер завдяки сервісам API. Запити на стороні сервера приймають контролери, які обробляють їх за допомогою бази даних або сторонніх сервісів. Найбільша перевагу такого способу - це те, що робота клієнта та сервера ізольована одна від одної та зміни в одному з них не впливають на іншого.

Запити до контролерів надходять завдяки протоколу HTTP а взаємодія з сервером SignalR відбувається через WebSocket для роботи застосунка в реальному часі.

В моєму випадку я використовую товстий клієнт, який бере велику кількість обчислень на себе таких як зберігання ненадісланих повідомлень, валідація деяких параметрів, взаємодія з сервісом дзвінків і тд. Взаємодія сервера з базою даних відбувається через клас DbContext для бази даних SQL та через BlockStorageService для взаємодії з сховищам файлів. Взаємодія клієнта з сервісом дзвінків відбувається через клас CallService.

```
[Authorize]
[HttpGet("get")]
0 references
public async Task<User?> Get()
{
    bool result = int.TryParse(User.FindFirst("userId")?.Value, out var userId);
    if (!result) return null;
    User? user = await _userRepository.GetByIdWithIncludeAsync(userId);
    user?.HideSensitiveData();
    return user;
}


[Authorize]
[HttpGet("getUserGroupsContacts")]
0 references
public async Task<IActionResult> GetUserGroupsContacts()
{
    bool result = int.TryParse(User.FindFirst("userId")?.Value, out var userId);
    if (!result) return BadRequest("Invalid user id");

    User? user = await _userRepository.GetByIdWithIncludeAsync(userId);
    if (user == null) return BadRequest("User not found");

    return Ok(new { user.Groups, user.Contacts });
}
```

Рис. 2 – Методи контролера

Метод Get() повертає загальну інформацію про користувача, приховуючи чутливі дані, а GetUserGroupsContacts надає його групи та контакти. Обидва методи перевіряють ідентифікатор користувача з токена, завантажують дані через _userRepository.

```
4 references
public async Task<ServerResponse<User>> GetUserAsync(string token)
{
    _httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);
    try
    {
        var response = await _httpClient.GetAsync(_url + "/get");
        return new ServerResponse<User>(response);
    }
    catch (Exception e)
    {
        return new ServerResponse<User>(null, 500, false, e.Message);
    }
}


4 references
public async Task<ServerResponse<List<User>>> GetUsersAsync()
{
    try
    {
        var response = await _httpClient.GetAsync(_url + "/getAll");
        return new ServerResponse<List<User>>(response);
    }
    catch (Exception e)
    {
        return new ServerResponse<List<User>>(null, 500, false, e.Message);
    }
}
```

Рис. 3 – Методи API які взаємодіють із сервером

Методи GetUserAsync() і GetUsersAsync() забезпечують отримання даних користувачів із сервера, обробляючи відповіді та формуючи уніфіковані результати.

Висновок: я реалізував застосунок завдяки client-server архітектурі, де клієнт відповідає за інтерфейс користувача, а сервер — за обробку запитів, зберігання даних і виконання бізнес-логіки. Цей підхід забезпечує централізоване управління даними, високу масштабованість і спрощує оновлення серверної частини, тому що вона ізольована від сторони клієнта.