PogadajNET project

5

https://github.com/debian-sh/pogadajNET

Adam Hryniewiecki
Arkadiusz Wieczorek
Bartosz Ziętkowski
Dawid Sroczyński
Maciej Nowak

1. Projekt się nie kompiluje.....
2. `DataModels.Enums` – konwencja w .NET = `PascalCase`, a nie `camelCase` (to jest używane w Javie), czy też `snake-case`! Zdarzają się rozjazdy u Was – często w firmach StyleCop weryfikuje takie rzeczy na poziomie commitów i szybko się nauczycie. Warto jednak o tym już teraz wiedzieć i pamiętać.
3. Dobrym zwyczajem jest deklarowanie zmiennych, jako interfejsów. Spójrzcie np. Na DataModels.User.UserFriendsList - gdzie macie _friends, jako List, a nie Ilist. Dotyczy to oczywiście wielu miejsc – jak np. Przyjmowanych parametrów, czy też rezultatów z metod. Tutaj macie odpowiedź Erica Lipperta (oraz więcej na ten temat) dlaczego warto myśleć o takich rzeczach - http://stackoverflow.com/questions/8717582/why-use-ilist-or-list
4. Wasza aplikacja u mnie nie zadziała:

```
13    public class ChatManagement
14    {
15        private string _serverAddress = "150.254.77.76";
16        private int _port = 5672;
17
```

   Takie rzeczy powinny lądować w plikach konfiguracyjnych I być z nich wyciągane na starcie aplikacji.
5. Jeśli stała powiaja się w kodzie więcej niż raz (jak np. „Queue” w Klient.ServerFacade.ChatManagement) to warto wyciagnąć ją albo jako stałą, albo nawet stworzyć specjalna „Data class” gdzie takie stałe byłyby przechowywane. Później łatwo jest taką wartość zmienić – a nie trzeba wyszukiwać wszystkich miejsc. Wyobraźmy sobie zmianę wymagań....
6. Liczba commitów na gitHubie **ZATRWAŻAJĄCA**:

   

7. Nie do końca (***W OGÓLE!***) rozumiem logikę wrzucenia WCF, który komunikuje się z RabbitMQ. To już jest - https://www.nuget.org/packages/RabbitMQ.Client/ a metoda ExeCMD(string command)....

8. `PogadajPogadajService.PogadajPogadajService` – linia 191 wywołuje u mnie smutek i zakłopotanie:

```
public byte[] TestGet()
{
    Users u = new Users();
    User uu = new User();
    uu.Username = "aa";
    u.UserList.Add(uu);
    return null;
}
```

9. Bindingi - a wlasciwie ich brak.
10. Brak wzorca MVVM - większość logiki zaimplementowana w code-behind.
11. Zarządzanie użytkownikami na poziomie użytkowników RabbitMQ to kompletne niezrozumienie wymagań. Rozwiązanie jest całkowicie nieskalowalne (wystarczy sobie wyobrazić klaster RabbitMQ - każdy z nich musiałby mieć utrzymywaną tą samą listę użytkowników). Na dłuższą metę jest to koszmar z zarządzaniem uprawnieniami (proszę zwrócić uwagę, że każdy użytkownik w tym momencie może połączyć się i odczytać dowolną wiadomość z serwera).
12. Dodawnia użytkowników jest nietransakcyjne - możliwe jest, że dla dwóch żądań o założenie konta dla tego samego użytkownika, metoda CheckIfUserExists zwróci “false”.
13. Podnoszenie sesji RabbitMQ dla każdej operacji powoduje niepotrzebne obciążenie i zwiększa czas operacji.
14. Adresat wiadomości jest wykorzystywany do stworzenia nazwy kolejki - co jeśli nazwa użytkownika zawiera znaki specjalne (“.”, “#”)?
15. Wywoływanie komend na założenie użytkownika bez walidacji nazwy (proszę sobie poczytać o atakach typu SQL Injection - zrobili Panowie coś bardzo podobnego).
16. Klient i serwer nie współdzielą konraktów - w bardzo prosty sposób można doprowadzić do sytuacji, w której klient korzysta z jednej postaci obiektu Message a serwer z drugiego.
17. Magiczne Thread.Sleep(500) - to jest ukrywanie problemów z wielowątkowością.