

Seminarski rad C - C# i .NET

Implementacija chat-a pomoću MQTT biblioteke

### Nikola Kostić 725/20 Novi Sad, 2022.

**Sadrˇzaj**

[1 Šta je MQTT 3](#_Toc114779898)

[2 Operacije 4](#_Toc114779899)

[Connection 4](#_Toc114779900)

[Publish 4](#_Toc114779901)

[Subscribe 4](#_Toc114779902)

[Topic i QoS 5](#_Toc114779903)

[3 Implementacija 6](#_Toc114779904)

[MessageItem class 6](#_Toc114779905)

[MainWindow 6](#_Toc114779906)

[Odabir servera 6](#_Toc114779907)

[Connect button 7](#_Toc114779909)

[Publish message 9](#_Toc114779910)

[Disconnect 9](#_Toc114779911)

[Lokalni server 10](#_Toc114779912)

[MainWindow.xaml 11](#_Toc114779913)

[Run 11](#_Toc114779915)

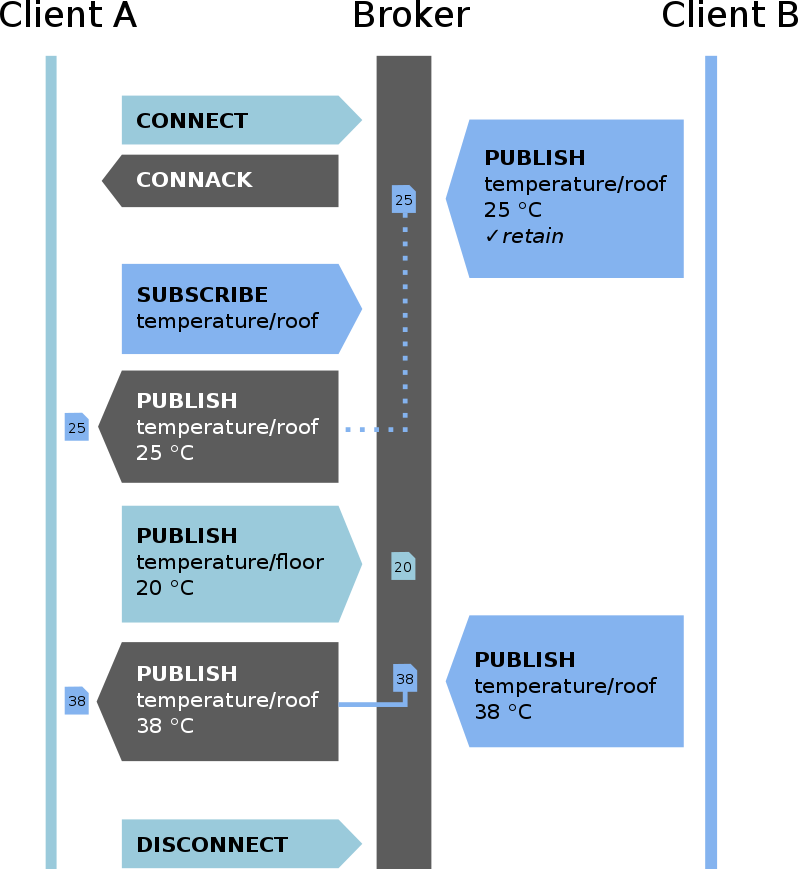
[4 Zakljuˇcak 13](#_Toc114779916)

[5 Literatura 14](#_Toc114779917)

# Šta je MQTT

MQTT klijent-server transportni protokol zasnovan na publish/subscribe načinu komunikacije. Najčešće se koristi za komunikaciju IoT uređaja sa serverom ili Machine to Machine (M2M). Postoji više verzija MQTT-a, a najviše korišćene su MQTT 3.1.1 i MQTT 5. Glavna poboljšanja kod verzije 5 su kontrola grešaka i mogućnost deploy na cloud platforme.

Za primenu MQTT-a potrebno je da postoji server (broker) i klijenti koji se kače na njega. Svaki klijent se pomoću topica kači (publish-uje) na broker i salje poruke. Klijenti koji su prijavljeni (subscribe) na određeni topic mogu da pročitaju poruke sa nje jer im ih broker šalje.



Svrha brokera je da prosleđuje poruke klijentima kojima su namenjene, a to može da radi na tri nacina:

1. Filtriranje po topicu
2. Filtriranje po sadržaju poruke koje nije baš najbolje rešenje jer mora biti otkriven sadržaj poruke i nema enkripcije poruka
3. Filtriranje po tipu, pogodno za OOP jezike tako što se filtrira tip klase poruke. Na primer mogu da se preslušaju sve poruke tipa Exception.

Ključne prednosti MQTT-a su:

1. Publisher i subscriber su prostorno razdvojeni i da bi slali i primali poruke, moraju da znaju IP ili ime hosta i port brokera;
2. MQTT pored isporuke poruka i realnom vremenu omogućava i isporuku klijentima koji nisu trenutno na mreži. Te poruke čuva broker uz dva uslova, da je klijent conektovan sa persistentnom sesiojom i da je QoS (Qualitz of Service) veći od 0.
3. MQTT radi asinhrono što znači da se zadaci obavljaju dok čekaju ili šalju poruku tj. ne blokiraju se. Po potrebi može raditi i sinhrono.

# Operacije

## Connection

MQTT je protocol baziran na TCP/IP. Konekcija je uvek između klijenta i brokera. Ne možemo direktno konektovati dva klijenta. Klijent šalje CONNECT poruku brokeru, a broker odgovara sa CONNACT porukom. Kad se uspostavi konekcija, ona traje sve dok klijent ne pošalje zahtev za prekid konekcije.

CONNECT poruka sadrži:

1. clientId
2. cleanSession – flag koji govori brokeru da li želimo persistent session
3. username
4. password
5. lastWillTopic, lastWillQoS, lastWillMessage, lastWillRetain
6. keepAlive – period u sekundama koliko može da traje konekcija, a da nema razmene poruka.

Username, password i sve last poruke su opcioni argumenti.

CONNACT poruka pored informacije da li postoji persistentSession, vraća i returnCode koji ima nekoliko značenja:

1. 0 – konektovan
2. 1 – pogrešna verzija protokola
3. 2 – loš id
4. 3 – server nije dostupan
5. 4 – pogrešna autentifikacija
6. 5 – nije autorizovan.

## Publish

Svaka poruka koja se publish-uje mora da sadrži topic koji će broker iskoristiti da prosledi poruku pravom klijentu. Pored topica postoje id druge informacije koje su značajne kao što su:

1. PacketId – za identifikaciju poruke između brokera i klijenta
2. QoS – Quality of Service koji ima tri nivoa i definiše garanciju isporuke za neku poruku.
3. retainFlag – omogućava da novi subscriber kad se prijavi vidi poslednju poruku na zadatom topicu
4. payload – predstavlja sadržaj poruke
5. dupFlag – poruka će se duplirati i poslati ponovo ako primalac nije potvrdio prijem poruke. Obično se koristi uz QoS 0.

## Subscribe

Da bi klijent mogao da čita poruke potrebno je da se subscribe-uje na željeni topic na brokeru. Subscribe

poruka sadrži pored packetId-a i listu parova vrednosti QoS-a i topica na koji se prijavljujemo.

Nakon slanja zahteva za prijavu broker odgoara SUBACK signalom gde nam daje potvrdu o uspešnoj prijavi ili o grešci. Kada smo uspešno prijavljeni, drugi klijenti mogu da šalju poruke na taj topic i broker će ih automatski publish-ovati na naš klijent.

Subscribe (1)

Publish (3)

Client

Client

Publish (4)

Broker

Suback (2)

Pored ovih signala postoje i UNSUBSCRIBE i UNSUBACK koji su slični SUBSCRIBE i SUBACK samo služe za prekid prijave na topic.

## Topic i QoS

Topic se koristi da broker može da odredi kome treba da prosledi poruke i od koga stižu poruke. Postoje neka ustaljena pravila za pisanje topica. Nivoi topica se razdvajaju „/”. Primer topica po nivoima je: srbija/zlatibor/uzice/centar. Svaki nivo može da ima više podnivoa. Svaki naziv u topicu može sadržati beline i mora imati barem jedan karakter.

Postoje i specijalni znaci za zadavanje topica koji nam u nekim slučajevima skraćuju nazive i kod.

Singl-level: srbija/zlatibor/+/centar - + označava da može bilo šta da se stavi umesto njega ako posle toga postoji i centar.

Multi-level: srbija/zlatibor/uzice/# - # označava da možemo koristit sve topice koji na početku imaju srbija/zlatibor/uzice.

Neka pravila za davanje naziva topica:

1. Ne sme počinjati sa /;
2. Beline i ako su dozvoljene ne bi trebalo da se koriste;
3. Koristiti samo ASCII karaktere
4. Ne bi trebali da se subsribujemo na sve poruke pomoću #;
5. Koristiti specifičan topic kad god je moguće.

QoS (Quality of Service) definiše garanciju isporuke za neku poruku. Postoje tri vrednosti 0, 1 i 2.

1. QoS 0 – jedom se šalje poruka i nema provere da li je stigla pa da se ponovo šalje. Obično se koristi kada imamo stabilnu konekciju, ako nije problem da se izgubi poruka i kad nam ne treba čekanje poruke.
2. QoS 1 – nam garantuje da je primalac dobio poruku bar jednom. Sender čuva poruku sve dok ne dobije PUBACK koji potvrđuje da je poruka primljena. Koristimo ga kad nam trebaju sve poruke i možemo da obrišemo duple.
3. QoS 2 – je najsigurniji, ali i najsporiji. Garantuje da primalac prima samo jednom poruku. Koristi se kad nam je baš bitno da se prime svi podaci samo jednom. Sporiji je od QoS 1 i ređe se koristi.

Sve poruke QoS 1 i QoS 2 koje čekaju u redu van mreže dok je klijent nedostupan se čuvaju u PERSISTION SESSION ako je klijent kreirao.

# Implementacija

## MessageItem class

MessageItem class nam slži za čuvanje teksta poruke i id klijenta.

internal class MessageItem

{

public string clientId { get; set; }

public string payload { get; set; }

public MessageItem(string clientId, string payload)

{

this.clientId = clientId;

this.payload = payload;

}

public MessageItem()

{

}

}

## MainWindow

public MainWindow()

{

items = new ObservableCollection<MessageItem>();

InitializeComponent();

}

U MainWindow pre inicijalizacije prozora, kreiran je ObservableCollection za smeštanje poruka. U WPF-u je bitno da se koristi Observable kolekcija da bi se promene emitovale na UI elementima.

## Odabir servera

## 

private void Odaberi\_Click(object sender, RoutedEventArgs e)

{

server = cbServer.SelectionBoxItem.ToString().Trim();

}

Klikom na dugme “Odaberi” biramo da li zelimo da se povežemo na lokalni server na IP adresi 127.0.0.1 ili na udaljeni server „test.mosquitto.org”.

## Connect button

private async void button\_Click\_Connect(object sender, RoutedEventArgs e)

{

mqttFactory = new MqttFactory();

client = mqttFactory.CreateMqttClient();

options = new MqttClientOptionsBuilder()

.WithClientId(myClientId)

.WithTcpServer(server, 1883)

.Build();

await client.ConnectAsync(options);

client.UseConnectedHandler(e =>

{

if (client.IsConnected)

{

//MessageBox.Show("Successfully connected.222");

Log.Logger.Information("Successfully connected.222");

}

});

topic = tbTopic.Text;

topicFilter = new TopicFilterBuilder()

.WithTopic(topic)

.Build();

await client.SubscribeAsync(topicFilter);

MessageBox.Show("Konektovani ste na topic: " + topic);

client.UseApplicationMessageReceivedHandler(e =>

{

string payload = Encoding.UTF8.GetString(e.ApplicationMessage.Payload);

if (payload != null && !string.Empty.Equals(payload))

{

var splittedMessage = payload.Split(';');

var clientId = splittedMessage[0];

var message = splittedMessage[1];

if (clientId.Equals(myClientId))

{

return;

}

Action invokeAction = new Action(() =>

{

items.Add(new MessageItem() { clientId = clientId, payload = message}); //ne koristi e.clientId, to je id sendera

listBox2.ItemsSource = items;

});

listBox2.Dispatcher.BeginInvoke(invokeAction);

}

});

}

Na klik Connect dugmeta koristimo naziv topika koji smo uneli da kreiramo TopicFilter koji nam je potreban da bi smo se subscribovali na željeni topik.

Connection dugme na početku vrši konekciju na server koji smo prethodno odabrali. Obadva servera se konektuju na port 1883 i koriste ID koji je zadat kao readonly string. Funkcija za konekciju (ConnectAsync) je asinhrona pa mora imati await kako bi nastavili sa radom tek kad se konekcija uspostavi. Provera da li je konekcija uspešna se proverava u UserConnectionHandler-u I poruka se može ispisati u Log-u ili kao MessageBox prozor.

Nakon toga pravimo konekciju na topic koji smo uneli preko TextBox-a na GUI-ju. Funkcija za prijavu je takođe asinhrona pa mora imati await. Ako je uspešno prošla prijava dobijamo poruku sa obaveštenjem na koji topic smo prijavljeni.

Ostatak funkcije za obradu Connect eventa sadrži handler za obradu primljenih poruka koje su namenjene tom klijentu. Za početak dobijenu poruku od servera treba da dekodiramo I vratimo u string format. Ako sve prođe kako treba i dobijemo našu poruku treba je parsirati kako bi dobili ID klijenta koji ju je poslao kao i samu poruku.

S obzirom da pristigle poruke prikazujemo u ListBoxu sšto je UI komponenta, a ona se nalazi na MainThread-u, ne može joj se direktno pristupiti iz drugih Thread-ova. Zbog toga koristimo Dispatcher koji nam to omogućava. U Dispatcher-u pozivamo na akciju koja će dodati pristigle poruke na ListBox.

## Publish message

private void button\_Click\_Push(object sender, RoutedEventArgs e)

{

var messageToSend = tbPayload.Text;

PublisHMessageAsync(client, messageToSend, topic);

}

private void PublisHMessageAsync(IMqttClient client, string message, string topic)

{

string messagePayload = $"{myClientId};{message}";

var messageObj = new MqttApplicationMessageBuilder()

.WithTopic(topic)

.WithPayload(messagePayload)

.WithAtLeastOnceQoS()

.Build();

if (client.IsConnected)

{

client.PublishAsync(messageObj);

}

}

Kada kliknemo na dugme za slanje poruke, buttonClickPush metod kupi tekst poruke sa TextBox-a i poziva metod PublishMessageAsync koji kreira string u formatu myClientId;message kako bi mogli da pošaljemo i id počiljaoca zajedno sa porukom. Pomoću MqttAplicationMessageBuildera kreiramo objekat poruke za slanje i zadajemo joj topik, payload i QoS.

## Disconnect

private void button\_Click\_Disconnect(object sender, RoutedEventArgs e)

{

client.DisconnectAsync();

client.UseDisconnectedHandler(e =>

{

MessageBox.Show("Successfully disconnected.");

});

}

Disconnect dugme će klijenta otkačiti sa brokera. Ponovnim pokretanjem aplikacije, klijent se može opet konektovati na brokera.

## Lokalni server

Lokalni server je obična konzolna aplikacija koja prihvata bilo koju TCP konekciju. Da bi se konektovali na njega, potrebno je da odaberemo localhost adresu u ComboBox-u (127.0.0.1). Preko factory i interfejsa IMqttServerOptions kreiramo server sa zadatim opcijama. Po default-u se zadaje port 1883 I ne koristi enkripciju. Na tako kreiran server pozivamo asinhroni metod za startovanje servera sa zadatim opcijama.

static void Main(string[] args)

{

\_ = Run\_Minimal\_Server();

}

public static async Task Run\_Minimal\_Server()

{

var mqttFactory = new MqttFactory();

// The port for the default endpoint is 1883.

// The default endpoint is NOT encrypted!

// Use the builder classes where possible.

IMqttServerOptions mqttServerOptions = new MqttServerOptionsBuilder().WithDefaultEndpoint().Build();

// The port can be changed using the following API (not used in this example).

// new MqttServerOptionsBuilder()

// .WithDefaultEndpoint()

// .WithDefaultEndpointPort(1234)

// .Build();

using (var mqttServer = mqttFactory.CreateMqttServer())

{

await mqttServer.StartAsync(mqttServerOptions);

Console.WriteLine("Press Enter to exit.");

Console.ReadLine();

// Stop and dispose the MQTT server if it is no longer needed!

await mqttServer.StopAsync();

}

}

## MainWindow.xaml

## 

<Grid Height="434" VerticalAlignment="Top">

<TextBox x:Name="tbPayload" HorizontalAlignment="Left" Height="23" Margin="142,187,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="266"/>

<Button x:Name="btnPush" Content="Send Message" HorizontalAlignment="Left" Height="37" Margin="448,180,0,0" VerticalAlignment="Top" Width="158" Click="button\_Click\_Push"/>

<ListBox x:Name="listBox3" SelectionMode="Extended" Height="162" Margin="88,245,72,0" VerticalAlignment="Top" Width="660">

<ListBox.ItemTemplate>

<DataTemplate>

<Grid Margin="0,2">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="319"/>

<ColumnDefinition Width="319"/>

</Grid.ColumnDefinitions>

<TextBlock Grid.Column="0" HorizontalAlignment="Left" Text="{Binding clientId}" />

<TextBlock Grid.Column="1" HorizontalAlignment="Left" Text="{Binding payload}" />

</Grid>

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>

<TextBox x:Name="tbTopic" HorizontalAlignment="Left" Height="23" Margin="142,121,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="225"/>

<Label x:Name="label" Content="Topic:" HorizontalAlignment="Left" Height="29" Margin="51,115,0,0" VerticalAlignment="Top" Width="72"/>

<Button x:Name="btnConnect" Content="Connect on topic" HorizontalAlignment="Left" Height="37" Margin="406,114,0,0" VerticalAlignment="Top" Width="158" Click="button\_Click\_Connect"/>

<Button x:Name="btnDisconnect" Content="Disconnect" HorizontalAlignment="Left" Height="37" Margin="580,115,0,0" VerticalAlignment="Top" Width="158" Click="button\_Click\_Disconnect"/>

<Label x:Name="lblMesage" Content="Message:" HorizontalAlignment="Left" Height="29" Margin="51,188,0,0" VerticalAlignment="Top" Width="72"/>

<Label Content="Server:" HorizontalAlignment="Left" Margin="51,62,0,0" VerticalAlignment="Top"/>

<ComboBox x:Name="cbServer" HorizontalAlignment="Left" Margin="142,66,0,0" VerticalAlignment="Top" Width="120">

<ComboBoxItem Content="test.mosquitto.org" Name="mosquito"/>

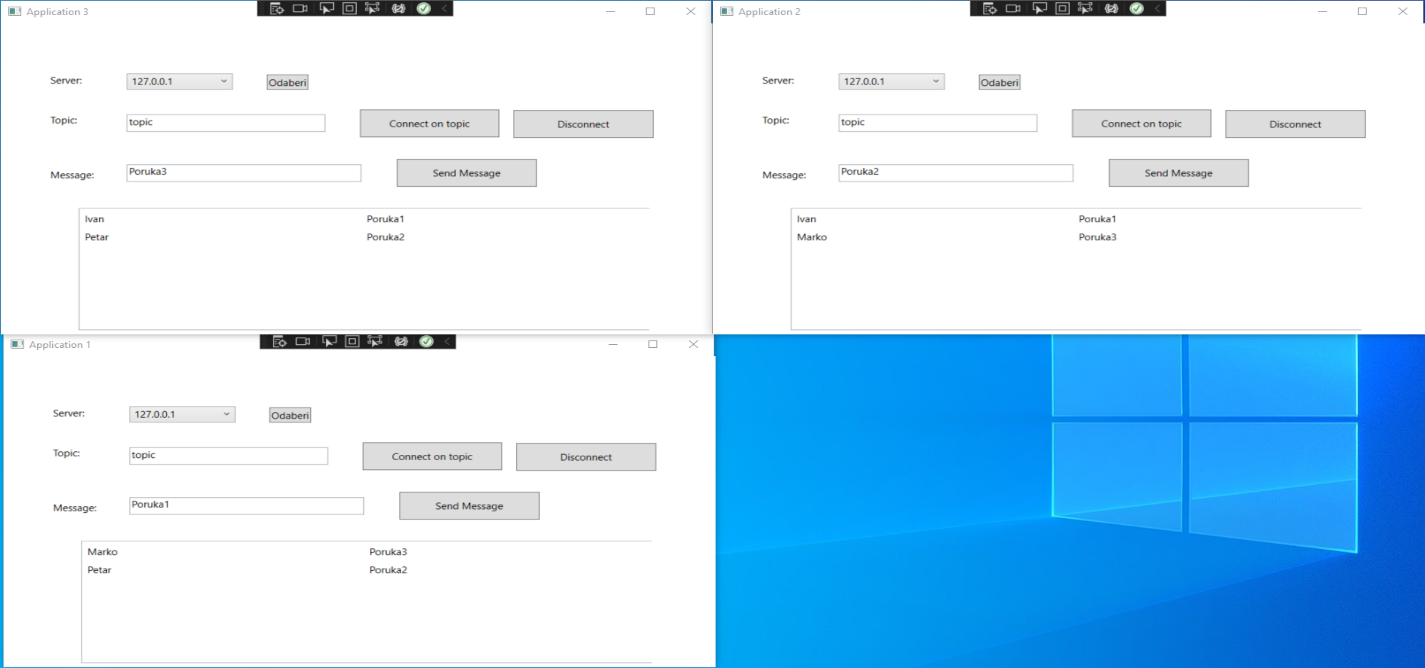
<ComboBoxItem Content="127.0.0.1" Name="localhost"/>

</ComboBox>

<Button x:Name="Odaberi" Content="Odaberi" HorizontalAlignment="Left" Margin="301,68,0,0" VerticalAlignment="Top" Click="Odaberi\_Click"/>

</Grid>

## Run



Solution se zapravo sastoji od dve odvojene WPF aplikacije koje su iste. Pokrenemo obe aplikacije u isto vreme. Kada se pokrenu, prvo iz ComboBox-a biramo server na koji želimo da se povežemo. Zatim unosimo topic kliknemo Connect on topic. Nakon toga možemo da šaljemo poruke.

Moguće je dodati još aplikacija u solution koje su iste kao prethodne i pokrenuti ih sve zajedno da dobijemo chat sa više klijenata. Testirao sam sa tri klijenta i radilo je bez problema.

# Zakljuˇcak

U ovom radu opisan je MQTT i implementacija chat aplikacije pomoću MQTT biblioteke. Aplikacija je realizovana kao WPF aplikacija koriˇs´cenjem C# i XAML programskog jezika.

Neke od ideja za dalju nadogradnju ove aplikacije su: da se omogući izbor nadimka za svakog klijenta umesto već zadatog u implementaciji; prikaz vremena kad je stigla poruka.

# Literatura

* <https://www.hivemq.com/mqtt-essentials/>
* <https://github.com/dotnet/MQTTnet>
* test.mosquitto.org
* <https://github.com/nikola031ue/MQTT-chat-app>