BlazorWebChat

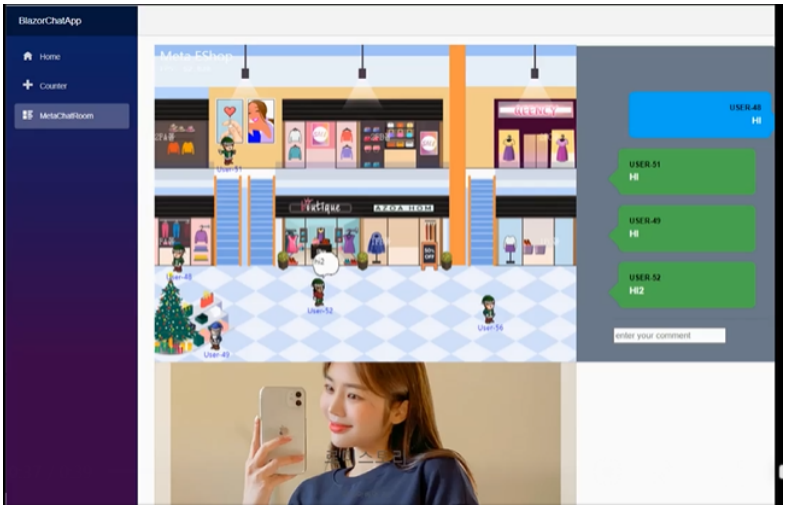
Blazor ~ .

GIT : https://github.com/psmon/BlazorChatApp

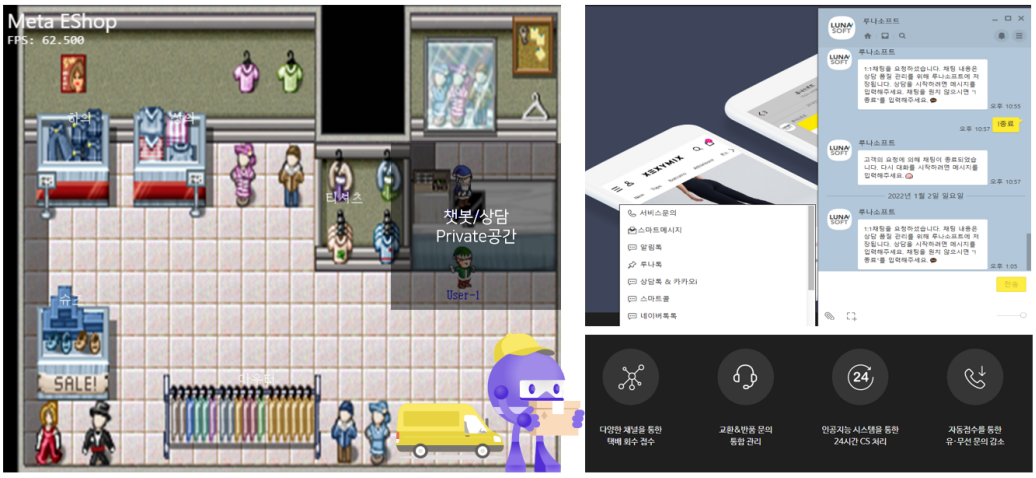
: https://sam.webnori.com/metaroom - git

- , () .
- , .
- .

, .



: .



루나톡 기능소개중 일부 - 루나소프트

: CS (~)

:

- (~)
- ( )
-
- ~

meta-chat3.mp4

2 -

() () C# . -

```
namespace BlazorChatApp.Shared
{
    public class ChatData
    {
    }

    public class UserInfo
    {
        public string Id { get; set; }
        public string Name { get; set; }

        public string Color { get; set; }
    }

    public class RoomInfo
    {
        public string Id { get; set; }
        public string Name { get; set; }
    }

    public class UpdateUserPos :UserInfo
    {
        public double PosX{get; set; }
        public double PosY{get; set; }

    }


    public class ChatMessage
    {
        public UserInfo From { get; set; }
        public string Message { get; set; }
    }

    public class JoinRoom
    {
        public RoomInfo RoomInfo { get; set; }
        public UserInfo UserInfo { get; set; }
    }

    public class SyncRoom
    {
        public RoomInfo RoomInfo { get; set; }
        public UserInfo UserInfo { get; set; }
    }

    public class LeaveRoom
    {
        public RoomInfo RoomInfo { get; set; }
        public UserInfo UserInfo { get; set; }
    }


    public class BaseCmd
    {
        public string Command {get;set; }
    }


    public class RoomCmd : BaseCmd
    {
        public UserInfo UserInfo{ get; set; }
        public object Data{get;set; }
    }

}
```

R , .

- Cilent To Server : .
- Server To Client : , .

```
using System.Collections.Generic;
using System.Threading.Tasks;

using Akka.Actor;

using BlazorChatApp.Shared;

using Microsoft.AspNetCore.SignalR;

namespace BlazorChatApp.Server.Hubs
{
    public class ChatHub : Hub
    {
        private ActorSystem actorSystem;

        private ActorSelection roomActor;

        public ChatHub(ActorSystem _actorSystem)
        {
            actorSystem = _actorSystem;
            roomActor = actorSystem.ActorSelection("user/room1");
        }


        // Client To Server

        public async Task JoInRoom(JoinRoom joinRoom)
        {
            roomActor.Tell(joinRoom);
        }

        public async Task SyncRoom(SyncRoom syncRoom)
        {
            roomActor.Tell(syncRoom);
        }

        public async Task LeaveRoom(LeaveRoom leaveRoom)
        {
            roomActor.Tell(leaveRoom);
        }

        public async Task UpdateUserPos(UpdateUserPos updateUserPos)
        {
            roomActor.Tell(updateUserPos);
        }

        // Server To Client : RoomActor  Hub Context   .



    }
}
```

.

.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.Json;
using System.Threading.Tasks;

using Akka.Actor;
using Akka.Event;

using BlazorChatApp.Shared;

using Microsoft.AspNetCore.SignalR;
using Microsoft.AspNetCore.SignalR.Client;
using Microsoft.Extensions.DependencyInjection;


namespace BlazorChatApp.Server.Hubs
{
    public class RoomActor : ReceiveActor
    {
        private readonly ILoggingAdapter log = Context.GetLogger();

        public Dictionary<string,UpdateUserPos> users = new Dictionary<string,UpdateUserPos>();

        private string roomName;

        private int userAutoNo = 0;

        private readonly IServiceScopeFactory scopeFactory;


        Random random= new Random();

        public RoomActor(string _roomName, IServiceScopeFactory _scopeFactory)
        {
            scopeFactory = _scopeFactory;

            roomName = _roomName;

            log.Info($"Create Room{roomName}");

            Receive<RoomCmd>(cmd => {
                log.Info("Received String message: {0}", cmd);
                //Sender.Tell(message);
            });

            Receive<JoinRoom>(async cmd => {
                userAutoNo++;
                string jsonString = JsonSerializer.Serialize(cmd);
                log.Info("Received JoinRoom message: {0}", jsonString);
                string RandomColor =  string.Format("#{0:X6}", random.Next(0xFFFFFF));

                UserInfo userInfo = new UserInfo()
                {
                    Id=cmd.UserInfo.Id,
                    Name=$"User-{userAutoNo}",
                    Color=RandomColor
                };

                UpdateUserPos updateUserPos= new UpdateUserPos()
                {
                    Id=cmd.UserInfo.Id,
                    Name=$"User-{userAutoNo}",
                    PosX=random.NextDouble()*300+20,PosY=random.NextDouble()*300+20,
                    ConnectionId = cmd.ConnectionId
                };

                users[cmd.UserInfo.Id] = updateUserPos;
```

```
    await OnJoinRoom(cmd.RoomInfo, userInfo, updateUserPos);
});

Receive<SyncRoom>(async cmd => {
    userAutoNo++;
    string jsonString = JsonSerializer.Serialize(cmd);
    log.Info("Received SyncRoom message: {0}", jsonString);

    List<UpdateUserPos> updateUserPosList = users.Values.ToList();
    //await hubConnection.SendAsync("OnSyncRoom", cmd.UserInfo, updateUserPosList);
    string RandomColor =  string.Format("#{0:X6}", random.Next(0xFFFFFF));

    UserInfo userInfo = new UserInfo()
    {
        Id=cmd.UserInfo.Id,
        Name=$"User-{userAutoNo}",
        Color=RandomColor
    };

    await OnSyncRoom(userInfo, updateUserPosList);

});

Receive<ChatMessage>(async cmd => {
    userAutoNo++;
    string jsonString = JsonSerializer.Serialize(cmd);
    log.Info("Received ChatMessage message: {0}", jsonString);

    ChatMessage chatMessage = new ChatMessage()
    {
        From = cmd.From,
        Message = cmd.Message
    };

    await OnChatMessage(chatMessage);

});

Receive<UpdateUserPos>(async cmd => {
    string jsonString = JsonSerializer.Serialize(cmd);
    log.Info("Received UpdateUserPos message: {0}", jsonString);

    double AbsPosX = users[cmd.Id].PosX+cmd.PosX;
    double AbsPosY= users[cmd.Id].PosY+cmd.PosY;

    if(users.ContainsKey(cmd.Id))
    {
        users[cmd.Id].PosX=AbsPosX;
        users[cmd.Id].PosY=AbsPosY;
    }

    log.Info($"UpdateUser : X=>{AbsPosX} Y=>{AbsPosY}");

    UpdateUserPos updateUserPos = new UpdateUserPos()
    {
        Id = cmd.Id,
        Name = cmd.Name,
        PosX = cmd.PosX,
        PosY = cmd.PosY,
        AbsPosX = AbsPosX,
        AbsPosY = AbsPosY
    };

    await OnUpdateUserPos(updateUserPos);

});

Receive<Disconnect>(async cmd => {
    string jsonString = JsonSerializer.Serialize(cmd);
    log.Info("Received Disconnect message: {0}", jsonString);
```

```
                var disconnectUser = users.Values.Where(e=> e.ConnectionId == cmd.ConnectionId).
FirstOrDefault();

                if(disconnectUser != null)
                {
                    if(users.ContainsKey(disconnectUser.Id))
                    {
                        users.Remove(disconnectUser.Id);

                        var leaveMsg = new LeaveRoom()
                        {
                            UserInfo = new UserInfo(){ Id =disconnectUser.Id }

                        };
                        await OnLeaveRoom(leaveMsg);
                    }
                }
            });

            Receive<LeaveRoom>(async cmd => {
                string jsonString = JsonSerializer.Serialize(cmd);
                log.Info("Received LeaveRoom message: {0}", jsonString);

                if(users.ContainsKey(cmd.UserInfo.Id))
                {
                    users.Remove(cmd.UserInfo.Id);
                    await OnLeaveRoom(cmd);
                }
            });

        }

        public async Task OnJoinRoom(RoomInfo roomInfo, UserInfo user, UpdateUserPos updateUserPos)
        {
            using(var scope = scopeFactory.CreateScope())
            {
                var wsHub = scope.ServiceProvider.GetRequiredService<IHubContext<ChatHub>>();
                await wsHub.Clients.All.SendAsync("OnJoinRoom", roomInfo, user, updateUserPos);
            }
        }

        public async Task OnSyncRoom(UserInfo user, List<UpdateUserPos> updateUserPos )
        {
            using(var scope = scopeFactory.CreateScope())
            {
                var wsHub = scope.ServiceProvider.GetRequiredService<IHubContext<ChatHub>>();
                await wsHub.Clients.All.SendAsync("OnSyncRoom", user, updateUserPos);
            }
        }

        public async Task OnLeaveRoom(LeaveRoom leaveRoom)
        {
            using(var scope = scopeFactory.CreateScope())
            {
                var wsHub = scope.ServiceProvider.GetRequiredService<IHubContext<ChatHub>>();
                await wsHub.Clients.All.SendAsync("OnLeaveRoom", leaveRoom);
            }
        }

        public async Task OnUpdateUserPos(UpdateUserPos updatePos)
        {
            using(var scope = scopeFactory.CreateScope())
            {
                var wsHub = scope.ServiceProvider.GetRequiredService<IHubContext<ChatHub>>();
                await wsHub.Clients.All.SendAsync("OnUpdateUserPos", updatePos);
            }
        }

        public async Task OnChatMessage(ChatMessage chatMessage)
        {
            using(var scope = scopeFactory.CreateScope())
```

```
            {
                var wsHub = scope.ServiceProvider.GetRequiredService<IHubContext<ChatHub>>();
                await wsHub.Clients.All.SendAsync("OnChatMessage", chatMessage);
            }
        }
    }
}
```

~ Blazor C#

C# .

ChatRoom.razor

```
protected override async Task OnInitializedAsync()
{

    LoginId = Guid.NewGuid().ToString();    //Fake Login ID

    hubConnection = new HubConnectionBuilder()
        .WithUrl(_navigationManager.ToAbsoluteUri("/chathub"))
        .Build();

    hubConnection.On<RoomInfo, UserInfo, UpdateUserPos>("OnJoinRoom", (room, user, pos) =>
    {
        Console.WriteLine($"WS - OnJoinRoom");
        if(user.Id == LoginId)
        {
            Name = user.Name;
            RoomName = room.Name;
            StateHasChanged();
        }
        else
        {
            BallField.AddUser(user.Id,user.Name, pos.PosX, pos.PosY);
        }
    });

    hubConnection.On<UserInfo,List<UpdateUserPos>>("OnSyncRoom", (user, updateUserPos) =>
    {
        Console.WriteLine($"WS - OnSyncRoom");
        if(user.Id == LoginId)
        {
             foreach(var pos in updateUserPos)
             {
                 BallField.AddUser(pos.Id, pos.Name, pos.PosX, pos.PosY);
             }
        }
    });

    hubConnection.On<UpdateUserPos>("OnUpdateUserPos", (userPos) =>
    {
        BallField.UpdateUserPos(userPos);
    });

    hubConnection.On<LeaveRoom>("OnLeaveRoom", (room) =>
    {
        Console.WriteLine($"WS - OnLeaveRoom");
        BallField.RemoveUser(room.UserInfo.Id);
    });

    await hubConnection.StartAsync();

    JoinRoom sendMsg = new JoinRoom()
    {
        UserInfo = new UserInfo(){Name="user", Id= LoginId},
        RoomInfo = new RoomInfo(){Name="room1"}
    };

    SyncRoom syndMsg = new SyncRoom()
    {
        UserInfo = new UserInfo(){Name="user", Id= LoginId},
        RoomInfo = new RoomInfo(){Name="room1"}
    };

    await hubConnection.SendAsync("JoInRoom", sendMsg);

    await hubConnection.SendAsync("SyncRoom", syndMsg);

}
```

Blazor , Canvas()

C# .

 ~

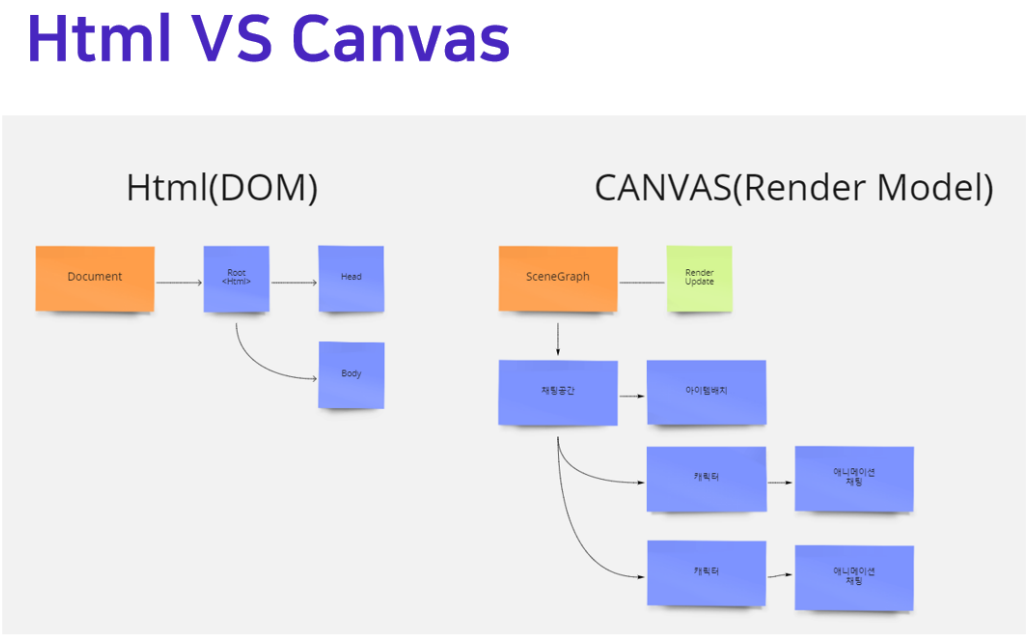
ChatRoom.razor

```
@using Blazor.Extensions.Canvas
@using Blazor.Extensions.Canvas.Canvas2D;

    [JSInvokable]
    public async ValueTask RenderInBlazor(float timeStamp)
    {
        double fps = 1.0 / (DateTime.Now - LastRender).TotalSeconds;
        LastRender = DateTime.Now;

        await this.ctx.BeginBatchAsync();
        await this.ctx.ClearRectAsync(0, 0, BallField.Width, BallField.Height);
        await this.ctx.SetFillStyleAsync("#003366");
        await this.ctx.FillRectAsync(0, 0, BallField.Width, BallField.Height);
        await this.ctx.SetFontAsync("26px Segoe UI");
        await this.ctx.SetFillStyleAsync("#FFFFFF");
        await this.ctx.FillTextAsync("Blazor WebAssembly + HTML Canvas", 10, 30);
        await this.ctx.SetFontAsync("16px consolas");
        await this.ctx.FillTextAsync($"FPS: {fps:0.000}", 10, 50);
        await this.ctx.SetStrokeStyleAsync("#FFFFFF");

        await this.ctx.SetFontAsync("12px consolas");
        await this.ctx.SetStrokeStyleAsync("#FFFFFF");

        foreach (var ball in BallField.Balls)
        {
            await this.ctx.FillTextAsync($"{ball.Name}", ball.X -10, ball.Y -20);
            await this.ctx.BeginPathAsync();
            await this.ctx.ArcAsync(ball.X, ball.Y, ball.Radius, 0, 2 * Math.PI, false);
            await this.ctx.SetFillStyleAsync(ball.Color);
            await this.ctx.FillAsync();
            await this.ctx.StrokeAsync();
        }
        await this.ctx.EndBatchAsync();
    }
```

Blazor - conf 2022

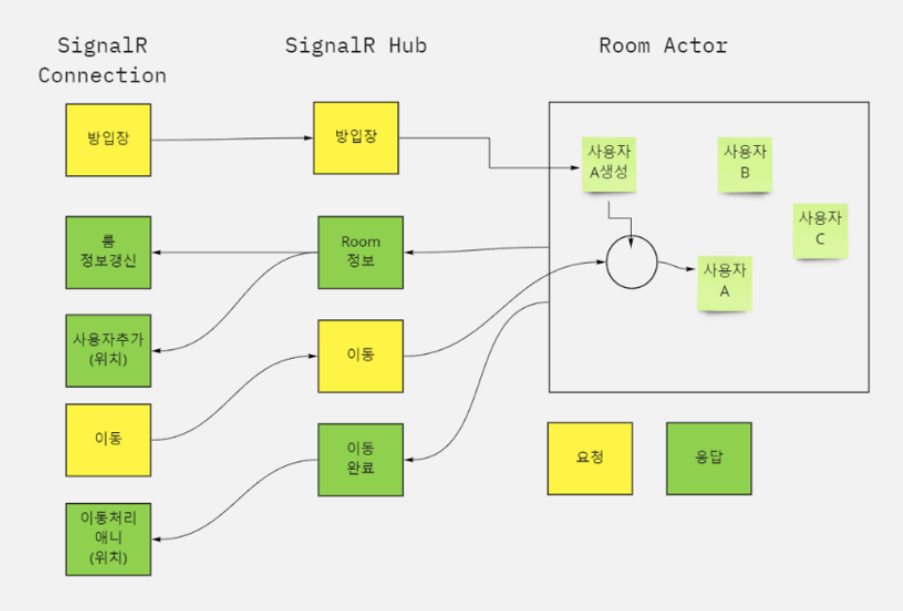
dotnetconf2022-□□□□□□□-□□□.pptx

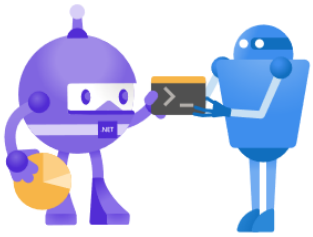
Html VS Canvas



Html5 : http://psmon.x-y.net/pscoco/sample.html
JS Canvas .(html5 ie6,7 ~)

SignalR + Akka.net 통신메시지





.

, .

Pub/Sub Redis .

Akka-Actor .

> Node.js Socket.io
>
> AkkaServerLess . / .
>
> https://developer.lightbend.com/docs/akka-serverless/javascript/index.html

,, .

2D .

- https://github.com/mizrael/BlazorCanvas - (.)
- https://www.mapeditor.org/ - 2D