

# LUX.API

---

Classe per routing/gestione comunicazione tra servizi, ad esempio

- ricezione richiesta REST di calcolo di un SVG a partire da un JWD
- inoltro richiesta (via REDIS channel) all'Engine EgwMultiEngine
- attesa risposta, salvataggio in cache REDIS + inoltro su canale pubblicazione immagini x interfacce

## Sviluppi

### Multi-esecutore

potremmo avere tanti EgwMultiEngine in ascolto (ognuno con un suo channel x le richieste e le risposte) che potrebbero concorrere ad aumentare le capacità di calcolo e far scalare la soluzione (e vendere quindi upgrade al sistema).

per farlo andrebbero implementati REDIS channels dinamicamente sottoscritti/gestiti

### Gestione Channels dinamici

Si potrebbe ipotizzare un insieme di classi come le seguenti (da valutare)

```
public class RedisSubscriptionManager
{
    private readonly ISubscriber _subscriber;
    private readonly ILogger<RedisSubscriptionManager> _logger;

    private readonly ConcurrentDictionary<string, ChannelSubscription>
    _subscriptions = new();

    public RedisSubscriptionManager(IConnectionMultiplexer connection,
    ILogger<RedisSubscriptionManager> logger)
    {
        _subscriber = connection.GetSubscriber();
        _logger = logger;
    }

    public bool Subscribe(string channel, Action<RedisChannel, RedisValue>
    handler)
    {
        if (_subscriptions.ContainsKey(channel))
            return false;

        var task = _subscriber.SubscribeAsync(channel, handler);
        _subscriptions[channel] = new ChannelSubscription(channel, handler);
        _logger.LogInformation($"📌 Subscribed to channel: {channel}");

        return true;
    }
}
```

```

    public bool Unsubscribe(string channel)
    {
        if (!_subscriptions.TryRemove(channel, out var sub))
            return false;

        _subscriber.Unsubscribe(channel, sub.Handler);
        _logger.LogInformation($"✖ Unsubscribed from channel: {channel}");

        return true;
    }

    public IEnumerable<string> GetActiveChannels() => _subscriptions.Keys;
}

public record ChannelSubscription(string Channel, Action<RedisChannel, RedisValue>
Handler);

```

```

public class RedisDynamicSubscriberService : BackgroundService
{
    private readonly RedisSubscriptionManager _subManager;

    public RedisDynamicSubscriberService(RedisSubscriptionManager subManager)
    {
        _subManager = subManager;
    }

    protected override Task ExecuteAsync(CancellationToken stoppingToken)
    {
        // Subscribe to some defaults
        _subManager.Subscribe("default:events", HandleDefault);
        _subManager.Subscribe("notifications", HandleNotify);

        // Later you can call _subManager.Subscribe(...) based on inputs

        return Task.CompletedTask;
    }

    private void HandleDefault(RedisChannel ch, RedisValue msg) =>
        Console.WriteLine($"✉ {msg}");
    private void HandleNotify(RedisChannel ch, RedisValue msg) =>
        Console.WriteLine($"🔔 {msg}");
}

```

## Version History

tabella versioni