Logging Structured Message Templates selbst verwenden

Um selbst Strukturierte Messages Templates wie im Logging zu verwenden, kann man folgende Annotations nutzen:
[crayon-6766bbb212d59957677119/]

Host oder Webhost mit Autofac

[crayon-6766bbb213164737460217/]

Entity Framework EF – Tabellen Models generieren lassen

Alt + T -> „NU“ eingeben
[crayon-6766bbb2132ae609035241/]

Template parsen in C#

Ok habe ich mir gedacht. Es wird ja wohl nicht so schwer sein folgenden Code zu parsen und die Templates entsprechend zu ersetzen:
[crayon-6766bbb213486795070097/]
vue.js oder mustache verwendet eine ähnliche Syntax.

Ich könnte an dieser Stelle mustache für C# verwenden und er würde mir das Template ersetzen, allerdings benötige ich nicht nur das Ergebnis als String sondern ich möchte bei jedenm Iterationsaufruf eingreifen können.

Bisher habe ich mir folgende Tools angeschaut:

- Mustache (Stubble): GitHub – StubbleOrg/Stubble: Trimmed down {{mustache}} templates in .NET
- Antlr4: antlr4/index.md at master · antlr/antlr4 · GitHub
- RegEx (Möchte ich aus performance Gründen nicht nutzen)
- Sprache: GitHub – sprache/Sprache: A tiny, friendly, C# parser construction library
- Superpower (Erweiterung von Sprache mit besserer Fehlerausgabe) GitHub – datalust/superpower: A C# parser construction toolkit with high-quality error reporting
- Handlebars: GitHub – Handlebars-Net/Handlebars.Net: A real .NET Handlebars engine

Leider ist Sprache sehr schlecht dokumentiert und es fehlen Beispiele wie man das umsetzen könnte. Dennoch würde ich es gerne bevorzugen.

Bibliotheken für Authorisierung

Das Thema Authorisierungen ist sehr komplex und es gibt dazu viele Ansätze. Hier ist eine Liste von Bibliotheken:

Casbin

Vergleich Casbin mit OPA: OPA vs Casbin (github.com)

https://www.openpolicyagent.org/

oso Documentation – oso Documentation (osohq.com) (Nur Python und Java)

ory/ladon: A SDK for access control policies: authorization for the microservice and IoT age. Inspired by AWS IAM policies. Written for Go. (github.com) (Veraltet)

teramoby/speedle-plus: Speedle+ is an open source project for access management. It is based on Speedle open source project and maintained by previous Speedle maintainers. (github.com)

ASP Core 3.1 inkl VueCLI mit mehreren VueApps verwenden

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
```

```
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.SpaServices;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.FileProviders;
using Microsoft.Extensions.Hosting;
using VueCliMiddleware;

namespace ASPMultipleVueSPA
{
public class Startup
{
// This method gets called by the runtime. Use this method to add services to the container.
// For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940
public void ConfigureServices(IServiceCollection services)
{
}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
if (env.IsDevelopment())
{
app.UseDeveloperExceptionPage();
}

app.UseRouting();

app.UseEndpoints(endpoints =>
{
endpoints.MapGet(„/“, async context =>
{
await context.Response.WriteAsync(„Hello World!“);
});
```

```
/*
* // Wichtig: In vue.config.js publicPath eintragen! z.B.
module.exports = {
publicPath: ,/en/‘
};
*/

endpoints.MapToVueCliProxy(
pattern: „en/{*path}“,
options: new SpaOptions { SourcePath = „ClientApp“ },
npmScript: (System.Diagnostics.Debugger.IsAttached) ? „serve“
: null,
port: 8080,
https: false,
runner: ScriptRunnerType.Npm,
regex: „Compiled successfully“,
forceKill: true,
wsl: false
);

endpoints.MapToVueCliProxy(
pattern: „fr/{*path}“, // Wichtig: In vue.config.js publicPath
eintragen: publicPath: ,/fr/‘
options: new SpaOptions { SourcePath = „ClientApp2“ },
npmScript: (System.Diagnostics.Debugger.IsAttached) ? „serve“
: null,
port: 8081,
https: false,
runner: ScriptRunnerType.Npm,
regex: „Compiled successfully“,
forceKill: true,
wsl: false
);

});

}
}
```

```
}
```

ASP Core Configuration vererben

```
public IConfiguration Configuration { get; }
```

in ServiceCollection

```
services.Configure<BaseConfiguration>(this.Configuration);
services.Configure<ChildConfiguration>(this.Configuration);
```

Die ChildConfiguration erbt von BaseConfiguration

Hat die ChildConfiguration Unterknoten, erben auch die Unterknoten. So wird die Konfiguration „erweitert“

Integrations Test mit Dependency Injection MsTest v2 .Net Core 3.1

Möchte man einen Integrationstest schreiben und dabei dependency injection nutzen, muss man folgendermaßen vorgehen:

1. Im Besten Fall teilt ma die Ausführende Applikation z.B. Web Applikation, Console etc und eine Bibliothek.

2. In die Bibliothek kommt die Auslagerung der StartUp:
   [crayon-6766bbb213643520917623/]
3. Im Testprojekt wird eine Basis Klasse definiert, die dieses Modul einliest. Nun kann in CofigureServices services.UseMyModule() aufgerufen werden oder der Host selbst gebaut werden:
   [crayon-6766bbb213646579306046/]
4. Die eigentliche Test Klasse erbt nun von diesen BaseUnitTest. Nun kann ein Propertiy erzeugt werden, welche einen Service aus der Dependency Injection ausliest:
   [crayon-6766bbb213648798090043/]

Async await

Wie funktionier await?

Ein Thread ist wie eine Pipeline, die einen bestimmten Code in die CPU gibt.

Mal angenommen Thread 1 (Ui Thread) will etwas downloaden und dann auf der UI darstellen. Während des Http Requests ist die Ui gesperrt. Weil der Thread 1 darauf wartet, bis der Server geantwortet hat.

Mit await wird ein freier Thread genommen und der macht den Request. Thread 1 wird nun freigegeben. Der Benutzer kann z.B. weiter die Ui wie gewohnt nutzen.

Wenn Thread 2 fertig ist, sagt er, ich bin fertig und möchte zu dem Thread zurück kehren, der ihn erzeugt hat: Thread 1.

Thread 1 übernimmt und der Codeblock wird weiter ausgeführt.

Wenn es nun egal ist, welcher Thread den Codeblock weiter ausführen soll, kann man
[crayon-6766bbb21385f066055524/]
nutzen.

.Result sollte nicht genutzt werden, da im Exceptio Stacktrace Fehlermeldungen wie „MoveNext()“ erscheinen. Diese kommen aus der kompilierten Statemachine. Besser ist es
[crayon-6766bbb213862042862586/]
zu nutzen, da dann unser Typ und unsere Exception zurück gegeben werden.

Zahl als String mit 2 Nachkommastellen und mit Punkt als Tausendertrennzeichen

[crayon-6766bbb213a09386973212/]