Bonyan Library Overview and Project Blueprint

Welcome to the Bonyan Library! This guide will help you understand the core structure and clean architecture approach of `Bonyan`, an encapsulated .NET Core solution that emphasizes maintainability and modularity while not relying too heavily on all .NET Core features. The focus is on Clean Architecture principles to ensure separation of concerns and a highly maintainable codebase.

Installation Methods

You can install the `Bonyan` packages using any of the following methods:

1. .NET CLI

Use the .NET Command Line Interface to add the `Bonyan.AspNetCore` package to your project:

```
dotnet add package Bonyan.AspNetCore
```

2. Package Manager Console (Visual Studio)

Alternatively, you can use the Package Manager Console available in Visual Studio:

```
Install-Package Bonyan.AspNetCore
```

3. PackageReference

Another option is to manually add the package reference in your project `.csproj` file:

```
<ItemGroup>
  <PackageReference Include="Bonyan.AspNetCore" Version="latest" />
</ItemGroup>
```

Make sure to replace `latest` with the version of the package you want to use.

Using Bonyan in Your Project

Once the `Bonyan` package is installed, you can configure it in your project by setting up an application builder. Here's a sample code snippet to get you started:

```
var builder = WebApplication
    .CreateApplicationBuilder<BonyanTemplateModule>(args);

var app = builder.Build();
```

```
app.Run();
```

This will create and configure a basic Bonyan application.

Module Structure and Clean Architecture Blueprint

In `Bonyan`, each module should inherit from `Module` (or `WebModule` for web-related modules). A module encapsulates a specific part of the application, promoting modularity and separation of concerns in line with Clean Architecture principles. This approach limits the direct use of the .NET Core features in favor of focusing on a more abstract and flexible design.

High-Level Structure

The `Bonyan` library is built on a modular architecture that promotes:

- **Encapsulation**: Each module has well-defined responsibilities and hides implementation details.
- **Separation of Concerns**: Different application layers are clearly separated to facilitate easier testing and maintainability.
- **Dependency Management**: Modules define their dependencies explicitly using attributes like `DependOnAttribute` to ensure proper initialization order.

Example Module

Below is a basic example of how you can create a module:

```
namespace BonyanTemplate.Api
{
    public class BonyanTemplateModule : Module
    {
        public override Task OnConfigureAsync(ModularityContext context)
        {
            // Custom configuration code here
            return base.OnConfigureAsync(context);
        }

        public override Task OnInitializeAsync(ModularityInitializedContext context)
        {
            // Custom initialization code here
            return base.OnInitializeAsync(context);
        }
    }
}
```

In this setup, each module encapsulates its behavior, ensuring that the logic is self-contained and follows the Clean Architecture principles.

Specifying Dependencies with `DependOnAttribute`

Bonyan allows you to define module dependencies using the `DependOnAttribute`. This attribute helps in managing dependencies between various modules, ensuring that dependent modules are loaded in the correct order. This approach aligns with Clean Architecture's focus on managing dependencies to keep the application flexible and testable.

Here's how you can specify a dependency between modules:

```
using Bonyan.Modularity;

namespace BonyanTemplate.Api
{
    [DependOn(typeof(AnotherModule))]
    public class BonyanTemplateModule : Module
    {
        public override Task OnConfigureAsync(ModularityContext context)
        {
            // Module configuration code
            return base.OnConfigureAsync(context);
        }
    }
}
```

In this example, `BonyanTemplateModule` depends on `AnotherModule`, meaning `AnotherModule` will be initialized before `BonyanTemplateModule`. This is particularly useful for managing complex applications that contain multiple interdependent modules.

Summary

- Install `Bonyan.AspNetCore` using CLI, Visual Studio, or by adding it to your `.csproj` file.
- Create a custom module by inheriting from `Module` or `WebModule`.
- Use `DependOnAttribute` to manage dependencies between your modules and adhere to Clean Architecture principles.

For more detailed information, please refer to the official documentation or contact the maintainers of the Bonyan library. We hope this guide helps you get started with Bonyan!

Example Console Output

Here is an example of what the console output might look like when modules with dependencies are created:

```
- BonyanTemplateModule created
  - BonyanFastEndpointSecurityModule created
    - BonyanFastEndpointModule created
      - BonyanAspNetCoreModule created
```

This output shows the order in which the modules are initialized, respecting their dependencies, and emphasizes the Clean Architecture approach to modularity.

Bonyan Library Installation and Setup Guide

Welcome to the Bonyan Library! This guide will walk you through installing and configuring the `Bonyan` and `Bonyan.AspNetCore` packages in your .NET Core project. Below, you'll find multiple methods to install the packages and set up a Bonyan module structure for your application.

Installation Methods

You can install the `Bonyan` packages using any of the following methods:

1. .NET CLI

Use the .NET Command Line Interface to add the `Bonyan.AspNetCore` package to your project:

```
dotnet add package Bonyan.AspNetCore
```

2. Package Manager Console (Visual Studio)

Alternatively, you can use the Package Manager Console available in Visual Studio:

```
Install-Package Bonyan.AspNetCore
```

3. PackageReference

Another option is to manually add the package reference in your project `.csproj` file:

```
<ItemGroup>
  <PackageReference Include="Bonyan.AspNetCore" Version="latest" />
</ItemGroup>
```

Make sure to replace `latest` with the version of the package you want to use.

Using Bonyan in Your Project

Once the `Bonyan` package is installed, you can configure it in your project by setting up an application builder. Here's a sample code snippet to get you started:

```
var builder = BonyanApplication
    .CreateApplicationBuilder<BonyanTemplateModule>(args);

var app = builder.Build();
```

```
app.Run();
```

This will create and configure a basic Bonyan application.

Module Structure

In `Bonyan`, each module should inherit from `Module` (or `WebModule` for web-related modules). A module encapsulates a specific part of the application, making it more maintainable and modular. Below is a basic example of how you can create a module:

```
namespace BonyanTemplate.Api
{
    public class BonyanTemplateModule : Module
    {
        public override Task OnConfigureAsync(ModularityContext context)
        {
            // Custom configuration code here
            return base.OnConfigureAsync(context);
        }

        public override Task OnInitializeAsync(ModularityInitializedContext context)
        {
            // Custom initialization code here
            return base.OnInitializeAsync(context);
        }
    }
}
```

Specifying Dependencies with `DependOnAttribute`

Bonyan allows you to define module dependencies using the `DependOnAttribute`. This attribute helps in managing dependencies between various modules, ensuring that dependent modules are loaded in the correct order.

Here's how you can specify a dependency between modules:

```
using Bonyan.Modularity;

namespace BonyanTemplate.Api
{
    [DependOn(typeof(AnotherModule))]
    public class BonyanTemplateModule : Module
    {
```

```
        public override Task OnConfigureAsync(ModularityContext context)
        {
            // Module configuration code
            return base.OnConfigureAsync(context);
        }
    }
}
```

In this example, `BonyanTemplateModule` depends on `AnotherModule`, meaning `AnotherModule` will be initialized before `BonyanTemplateModule`. This is particularly useful for managing complex applications that contain multiple interdependent modules.

Summary

- Install `Bonyan.AspNetCore` using CLI, Visual Studio, or by adding it to your `.csproj` file.
- Create a custom module by inheriting from `Module` or `WebModule`.
- Use `DependOnAttribute` to manage dependencies between your modules.

For more detailed information, please refer to the official documentation or contact the maintainers of the Bonyan library. We hope this guide helps you get started with Bonyan!

Example Console Output

Here is an example of what the console output might look like when modules with dependencies are created:

```
- BonyanTemplateModule created
  - AnotherModule created
    - Another2Module created
```

This output shows the order in which the modules are initialized, respecting their dependencies.