Rappel sur les Dependecy Injection

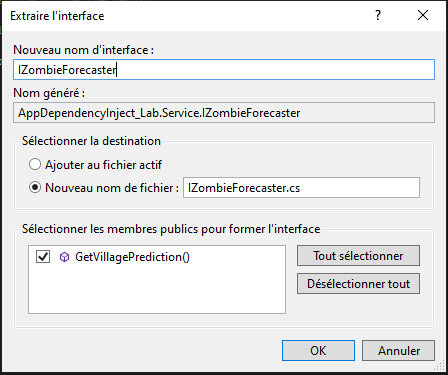
Le projet contient deux classes dans le dossier **Service**, *ZombieForecaster.cs* et *ZombieForecasterV2.cs* qui simulent les résultats de **Waze** pour prédire le déplacement des Zombies, *hardCodé* pour l’exercice (prédictions possible : enHausse, enBaisse, Imprevisible). **Voir le vidéo pour plus de détails.**

1. Créer la branche pour cette semaine **02\_DependencyInjection**
2. Ouvrez Visual Studio au niveau du dossier de la solution.

# Extraire l’abstraction de l’implémentation

## Extraire un Interface à partir d’une classe

1. Ouvrez la classe ***Service*** *ZombieForecaster.cs*
2. Sélectionnez le nom de la classe
3. *Extraire l’interface* dans un nouveau fichier (clic-droit, Actions rapides et refactorisation, Extraire interface)



1. Mettre l’interface *public*

# Dependency Injection 1 service

## Construtor dependency injection

1. Ouvrez le *HomeController*
2. Créez une propriété *readonly* pour enregistrer la prévision du taux de déplacement des zombies

public HomeVM homeVM { get; set; }

private readonly IZombieForecaster \_zombieForecaster;

1. Modifier le constructeur avec le service en paramètre

public HomeController(IZombieForecaster zombieForecaster)

{

homeVM = new HomeVM();

\_zombieForecaster = zombieForecaster;

}

1. Récupérez le résultat dans l’action **Index**

NbrZombiesResult currentNbrZombie = \_zombieForecaster.GetVillagePrediction();

1. Ajoutez le code qui détermine l’information envoyée à la View selon la valeur envoyée par le service

switch (currentNbrZombie.NbrZombiesCondition)

{

case NbrZombiesCondition.EnHausse:

homeVM.ZombieForecast = "Cours Forest! Cours!";

break;

case NbrZombiesCondition.EnBaisse:

homeVM.ZombieForecast = "Relaxe et respire les fleurs. Namsté.";

break;

case NbrZombiesCondition.Imprevisible:

homeVM.ZombieForecast = "En ces temps incertains, prends des forces en mangeant du chocolat.";

break;

default:

homeVM.ZombieForecast = "L'abonnement ou la vie!";

break;

}

1. Passez le homeVM en paramètre

return View(homeVM);

## Enregistrer le service au démarrage

1. Ouvrez le fichier *Startup.cs*
2. Enregistrez le service dans **ConfigureServices**

public void ConfigureServices(IServiceCollection services)

{

…

services.AddTransient<IZombieForecaster, ZombieForecaster>();

## Afficher dans la View Home, Index

1. Ajoutez la référence au ViewModels HomeVM

@model AppDependencyInject\_Lab.Models.ViewModels.HomeVM

1. Ajoutez la référence au ViewModels HomeVM

<p>Message du ZombieForecast Utility: @Model.ZombieForecast</p>

## Tester l’application

1. Exécutez l’application



# Mise en place de services thirdParty utilisant des clés

**Services :**

* **Stripe :** Passerelle de paiement en ligne sécurisé
* **Twilio :** Plateforme de communication cloud (ajouter messagerie, voix et la vidéo dans les applications.)
* **WazeTacker :** Application de suivi du traffic, police et accidents.

## Enregistrer les clés d’applications third party

1. Ouvrez le fichier *appsettings.json*
2. Ajoutez les clés bidons suivantes dans le fichier appsettings.json (n’oubliez pas la virgule)

,

"WazeForecast": {

"WazeTrackerEnabled": true

},

"Stripe": {

"SecretKey": "sk\_test\_WKWEURIFF324244l1BJwR00mx0mZGy6",

"PublishableKey": "pk\_test\_dw4earmEUBN73423423RER0FoTuud1j"

},

"Twilio": {

"PhoneNumber": "+1222345323",

"AuthToken": "er39",

"AccountSid": "QK32"

}

## Configurez le Container avec le IService Collection Version Séparément

1. Ouvrez le *Startup.cs*
2. Configurez les trois services afin de récupérer les données (clés, infos) de *appsettings.json.* On les récupère par leur nom (section). Insérez-les entre la connexion BD et les controller avec view

services.Configure<WazeForecastSettings>(configuration.GetSection("WazeForecast"));

services.Configure<StripeSettings>(configuration.GetSection("Stripe"));

services.Configure<TwilioSettings>(configuration.GetSection("Twilio"));

## Injecter les dépendances dans le controller

1. Ouvrez le ***HomeController***
2. Ajoutez les propriétés pour contenir les optionsSettings des 3 services

private readonly StripeSettings \_stripeOptions;

private readonly TwilioSettings \_twilioOptions;

private readonly WazeForecastSettings \_wazeOptions;

1. Ajoutez les 3 dans les paramètres du constructor en utilisant le type abstrait **IOptions<>**

public HomeController(

IOptions<WazeForecastSettings> wazeOptions,

IOptions<TwilioSettings> twilioOptions,

IOptions<StripeSettings> stripeOptions,

ILogger<HomeController> logger)

1. Passez les valeurs dans les propriétés. Attention d’utiliser la Value, qui extrait la valeur actuelle des appSettings ou n’importe quelle autre classe.

\_stripeOptions = stripeOptions.Value;

\_twilioOptions = twilioOptions.Value;

\_wazeOptions = wazeOptions.Value;

1. Ajoutez une action qui va appeler les configs.

public IActionResult AllConfigSettings()

{

List<string> messages = new List<string>();

messages.Add($"Waze config - Forecast tracker : " + \_wazeOptions.ForecastTrackerEnabled);

messages.Add($"Stripe Publishable key : " + \_stripeOptions.PublishableKey);

messages.Add($"Stripe Secret key : " + \_stripeOptions.SecretKey);

messages.Add($"Twilio Phone : " + \_twilioOptions.PhoneNumber);

messages.Add($"Twilio SID : " + \_twilioOptions.AccountSid);

messages.Add($"Twilio Token : " + \_twilioOptions.AuthToken);

return View(messages); }

## I

## njecter les dépendances dans la View

1. Modifier la view **AllConfigSettings,** ajoutez la référence au model List<string>
2. Ajoutez un code pour afficher l’ensemble des settings disponibles, après le h3

<ul>

@foreach (var message in Model)

{

<Li>@message</Li>

}

</ul>

1. Modifiez les tagHelpers du \_topNav afin que le menu Configuration Services

<a class="nav-link text-dark" **asp-area**="" **asp-controller**="Home"   
**asp-action**="AllConfigSettings">Configuration Services</a>

## Tester l’application

1. Exécutez et allez sur Configuration Services



# Organiser les Services pour garder le **startup** facile à consulter

Configurez le **Container** avec le **IService Collection**

1. Ouvrez le *Startup.cs*
2. Coupez les trois configurations de services.
3. Ouvrez **le Utility/***DI\_AppSettingsConfig.cs*
4. Créez une méthode public static qui retourne les abstractions **IServiceCollection object**et ***IConfiguration***
5. Déplacez les trois services afin de récupérer les données (clés, infos) de appsettings.json

public static IServiceCollection AddAppSettingsConfig(this IServiceCollection services, IConfiguration configuration)

{

services.Configure<WazeForecastSettings>(configuration.GetSection("WazeForecast"));

services.Configure<StripeSettings>(configuration.GetSection("Stripe"));

services.Configure<TwilioSettings>(configuration.GetSection("Twilio"));

return services;

}

1. Ouvez le *Startup.cs*
2. Ajoutez l’appel des services via le DI\_*AppSettingCofig*, entre la connexion BD et les controller avec view.

services.AddAppSettingsConfig(Configuration).AddAllServices();

1. Ouvrez le ***HomeController***
2. Ajoutez les propriétés pour contenir les optionsSettings des 3 services

private readonly StripeSettings \_stripeOptions;

private readonly TwilioSettings \_twilioOptions;

private readonly WazeForecastSettings \_wazeOptions;

1. Exécutez et allez sur **Configuration Services,** vous devriez obtenir le même résultat

# Comparer les 3 lifeTime Services possibles

Dans **Service, LifeTimeExample**, il y a trois classes identiques, représentant un service extérieur, qui sera créé avec un lifeTIme différent. Le GetGuid est le ID associé à chaque intenciation du service.

## Utiliser un mini Middleware

Dans le dossier **Middleware** (un vrai middleware serait dans un autre projet), vous avez un Middleware qui ne fait pas grand-chose à part ajouter le sevice pour les trois durée de vie possible et afficher une étiquette et le ID associés à chacun.

## Voir les trois versions du service

1. Ouvrez *Service, LifeTimeExample*
2. Consultez les trois versions du service
   * *ScopedService*
   * *SingletonService*
   * *TransientService*
3. Ouvrez *Middleware, CustomMiddleware*
4. Consultez la classe : elle contient les trois versions du service.

## Injecter le service du Middleware

1. Ouvez le *Startup.cs*
2. Enregistrez le Middleware dans **Configure,** après l’authentification

app.UseMiddleware<CustomMiddleware>();

1. Ajoutez un appel pour chaque service dans **ConfigureServices,** après les services du THirdParty

services.AddTransient<TransientService>();

services.AddScoped<ScopedService>();

services.AddSingleton<SingletonService>();

## Créer un controller pour le Middleware

1. Ouvez le *LifeTimeController.cs*
   * Ajoutez les propriétés
   * Complétez le constructor avec les paramètres
   * Complétez l’action de l’affichage de l’Index

public class LifeTimeController : Controller

{

private readonly TransientService \_transientService;

private readonly ScopedService \_scopedService;

private readonly SingletonService \_singletonService;

public LifeTimeController(TransientService transientService,

ScopedService scopedService, SingletonService singletonService)

{

\_transientService = transientService;

\_scopedService = scopedService;

\_singletonService = singletonService;

}

public IActionResult Index()

{

var messages = new List<String>

{

HttpContext.Items["CustomMiddlewareTransient"].ToString(),

$"Transient Controller - {\_transientService.GetGuid()}",

HttpContext.Items["CustomMiddlewareScoped"].ToString(),

$"Scoped Controller - {\_scopedService.GetGuid()}",

HttpContext.Items["CustomMiddlewareSingleton"].ToString(),

$"Singleton Controller - {\_singletonService.GetGuid()}",

};

return View(messages);

}

}

}

## Modifier la vue pour le Middleware

1. Ouvez la *LifeTime, Index*
2. Ajoutez le code suivant :

@model List<String>

@{

ViewData["Title"] = "Index";

}

<h1>Life Time Services</h1>

@{

var count = 0;

}

@foreach (string mess in Model)

{

if (count == 0)

{

<div style="font-size:20px;padding-top:20px;">

Transient Lifetime : différent à chaque requête

</div>

}

if (count == 2)

{

<div style="font-size:20px;padding-top:20px;">

Scoped Lifetime à chaque refresh

</div>

}

if (count == 4)

{

<div style="font-size:20px;padding-top:20px;">

Singleton Lifetime à chaque fois qu'on ouvre l'application

</div>

}

count++;

<div>

@mess

</div>

}

## Ajouter à la Navigation Bar

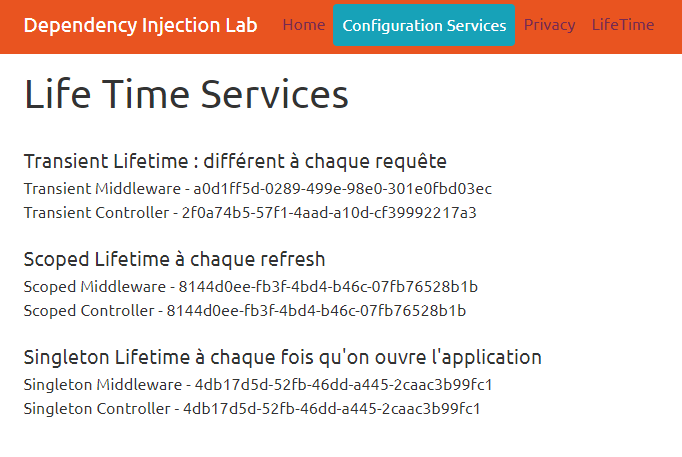
1. Ajoutez les informations du controller afin d’afficher la vue *Index* de *LifeTime*

<li class="nav-item">

<a class="nav-link text-dark" **asp-area**="" **asp-controller**="LifeTime" **asp-action**="Index">LifeTime</a>

## Tester l’application

1. Exécutez et allez sur Configuration LifeTime (les ID seront différents)



1. Rafraichissez la page plusieurs fois et portez attention aux IDs associés à chaque service