Laboratoire: Dependency injection

# Objectif(s)

* Dependency Injection
  + Durée de vie
  + Endroits de l’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). Les prédictions sont accessiblent via la méthode **GetVillagePrediction()**.

# Créer la branche de la séance

## Création du Repository et de la branche de la séance

1. Téléchargez les fichiers du laboratoire de la séance **S09\_DependencyInjection**
2. Dans Gitkraken, clonez votre repository GitHub **A21\_3W6\_Labos\_NOM\_PRENOM**

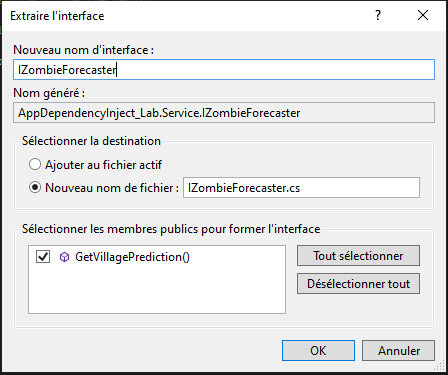
Pour ce laboratoire, vous devez partir de la copie de départ de l’enseignant

1. Cliquez sur la branche **Main**
2. Créez la branche de la séance: **S09\_DependencyInjection**

# 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. Clic-droit*, Actions rapides et refactorisation*
4. *Extraire l’interface* dans un nouveau fichier



1. Ouvrez le fichier créé **IzombieForcaster**
2. Mettre l’interface *public*

# Dependency Injection 1 service

## Enregistrer le service au démarrage

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

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<ApplicationDbContext>(options =>

{

options.UseSqlServer(Configuration.GetConnectionString("ZombiePartyContext"));

});

…

services.AddTransient<IZombieForecaster, ZombieForecaster>();

services.AddControllersWithViews().AddRazorRuntimeCompilation();

## **Construtor** dependency injection

1. Ouvrez le *HomeController*
2. Créez une propriété *readonly* pour enregistrer la prévision du taux de déplacement des zombies (ZombieForecaster) de type IzombieForecaster. Nommez-la \_zombieForecaster

public HomeVM homeVM { get; set; }

private readonly IZombieForecaster \_zombieForecaster;

1. Modifier le constructeur afin de passer le service en paramètre

public HomeController(IZombieForecaster zombieForecaster)

{

homeVM = new HomeVM();

\_zombieForecaster = zombieForecaster;

}

**Version 1 Injection de service**

1. Récupérez le résultat (GetVillagePrediction) dans l’action **Index,** dans une variable de NbrZombiesResult

NbrZombiesResult currentNbrZombie = \_zombieForecaster.GetVillagePrediction();

1. Ajoutez le code qui détermine l’information envoyée à la View selon la valeur envoyée par le service. Le message concerne le niveau de danger en fonction du danger représenté par le déplacement des Zombies (copiez le code ci-dessous)

public IActionResult Index()

{

// Version 1 injection dans le contructeur Action Index, récupérer le résultat dans une variable de NbrZombiesResult

NbrZombiesResult currentNbrZombie = \_zombieForecaster.GetVillagePrediction();

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;

}

return View();

1. Passez le homeVM en paramètre afin que le homeVM soit passé à la View Index

return View(homeVM);

## Afficher dans la View Home, Index

1. Ouvrez la view Home/**Index**, ajoutez la référence au ViewModels HomeVM

@model AppDependencyInject\_Lab.Models.ViewModels.HomeVM

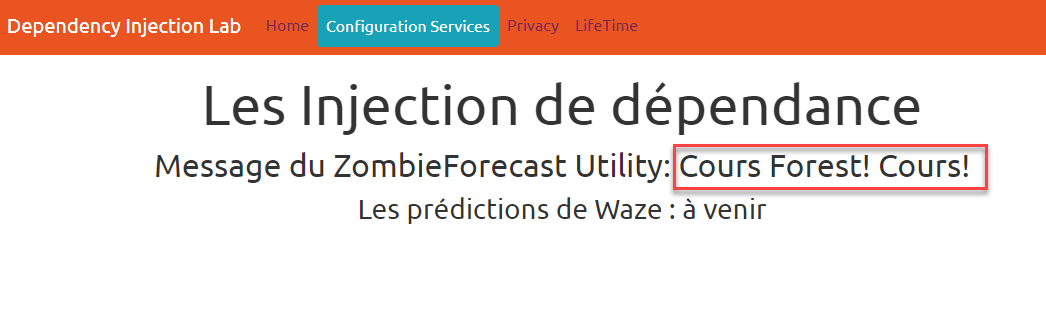
1. Remplacez **à venir**, par la référence au ViewModels HomeVM afin d’afficher le message en fonction du danger retourné par le controller

<h1 class="display-4">Les Injection de dépendance</h1>

<p>Message du ZombieForecast Utility: @Model.ZombieForecast <span class="text-info"><strong></strong></span></p>

## Tester l’application

1. Exécutez l’application



## Commentaires et validation (Commit) des changements dans le code

1. Dans Gitkraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modification: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés (vous serez noté là-dessus):
   * **Summary** FCT ForeCaster Version 1
   * **Description**: Affichage du message en fonction du danger des déplacements de zombies.
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

# 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)

"ConnectionStrings": {

"ZombiePartyContext": "server=(localdb)\\mssqllocaldb;Database=ZombieParty;Integrated Security=true;"

} ,

"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  
   Installer le package **System.Configuration.ConfigurationManager**

#region Insérez les références à l'ensemble des services ThirdParty ici Version Séparément

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

// Ajouter les propriétés multi-services (Stripe, twilio et waze) Version séparés ici

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,

IZombieForecaster zombieForecaster)

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.

\_zombieForecaster = zombieForecaster;

\_stripeOptions = stripeOptions.Value;

\_twilioOptions = twilioOptions.Value;

\_wazeOptions = wazeOptions.Value;

1. Ajoutez une action qui va appeler les configs.

#region Action AllConfigSetting version du constructeur muli-services

public IActionResult AllConfigSettings()

{

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

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

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); }

#endregion

## 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

<h3>Voici les configurations des services disponibles</h3>

<ul>

@foreach (var message in Model)

{

<Li>@message</Li>

}

</ul>

</div>

1. Modifiez les tagHelpers de la PartialView **\_topNav** afin que le menu Configuration Services appelle l’action **AllConfigSettings** du controller **Home**

<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



## Commentaires et validation (Commit) des changements dans le code

1. Dans Gitkraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modification: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés (vous serez noté là-dessus):
   * **Summary** FCT Injection de service ThirdParty
   * **Description**: Afficher les clés et autres valeurs fournies par des ThirdParty
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

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

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

1. Ouvrez le *Startup.cs*
2. Mettez les trois configurations de services injectés séparément en commentaire

#region Insérez les références à l'ensemble des services ThirdParty ici Version Séparément

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

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

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

#endregion

1. Ouvrez **le Utility*/****DI\_Config/DI\_AppSettingsConfig.cs*
2. Créez une méthode public static qui retourne les abstractions **IServiceCollection object**et ***IConfiguration***
3. Copiez-collez 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.

#region Insérez les références à l'ensemble des services Thirdparty ici Version Groupés

services.AddAppSettingsConfig(Configuration).AddAllServices();

#endregion

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

## Commentaires et validation (Commit) des changements dans le code

1. Dans Gitkraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modification: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés (vous serez noté là-dessus):
   * **Summary** FCT Injection de service ThirdParty version regroupée
   * **Description**: Afficher les clés et autres valeurs fournies par des ThirdParty via AppSettingConfig
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

# 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.UseAuthorization();

// Ajoutez la configuration du Middleware ICI

app.UseMiddleware<CustomMiddleware>();

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

#region Injection des trois version du Middleware

services.AddTransient<TransientService>();

services.AddScoped<ScopedService>();

services.AddSingleton<SingletonService>();

#endregion

## 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 View *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

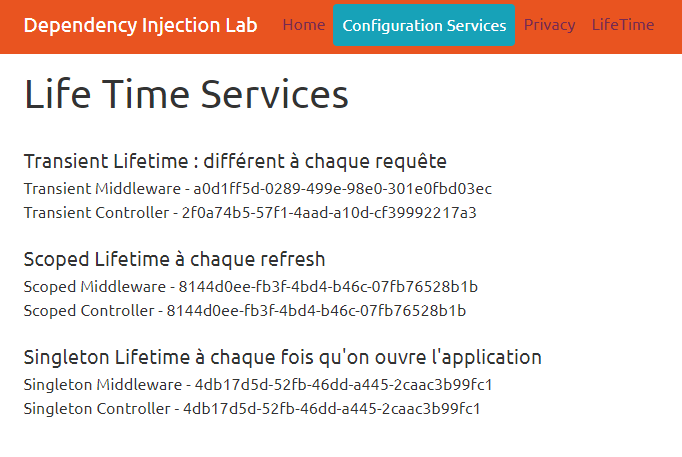
1. Ajoutez les informations du controller afin d’afficher la vue *Index* de *LifeTime* à la PartialView **TopNav**

<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)
2. Rafraichissez la page plusieurs fois et portez attention aux IDs associés à chaque service



## Commentaires et validation (Commit) des changements dans le code

1. Dans Gitkraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modification: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés (vous serez noté là-dessus):
   * **Summary** FCT Injection de service LifeTime différents
   * **Description**: Afficher ID des services à chaque injection
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 