# TP - Pattern Gang Of Four

Ce TP est la suite du TP Pattern GRASP. Nous allons d’abord exposer notre service « Basket », via une API Rest. Puis nous allons maintenant nous occuper de l’interface graphique. Les interfaces graphiques sont souvent très complexes et du coup, par nécessité très riche en « design-pattern ».

Dans le TP Précédent nous avons déjà utilisé certains patterns du « Gang Of Four » qui ne sont pas éloigné des pattern « GRASP » :

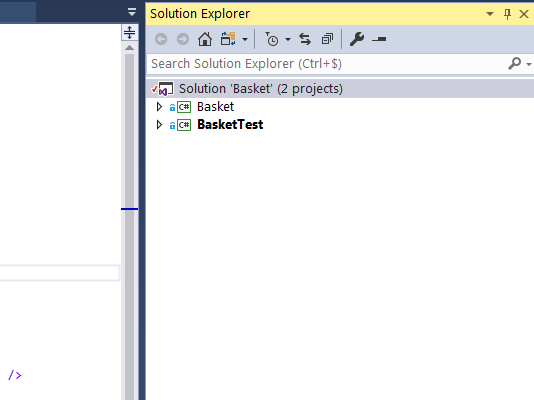
* **Façade** : équivalent au terme controller « BasketOperation » vu sous l’angle GRASP.
* **Adpater** : Notre « ArticleBase » est adapté avec ces différentes implémentations ; c’est équivalent au terme polymorphisme du pattern GRASP (qui lui-même est équivalent à faire tout simplement de la programmation objet).
* **Strategy** : Nous avons créé des interfaces et différentes implémentations, afin d’injecter des comportements différents dans les objets. C’est à peu de chose près équivalent au pattern « protection des variations » du GRASP.
* **Factory** : Pour instancier les différents « Article » et aussi pour les « BasketLine ». Il est vraiment propre au pattern du Gang Of Four. Il s’agit d’une « pure invention » si l’on regarde du côté des pattern « GRASP », mais qui est un peu plus spécifique.

L’IOC (**Inversion Of Control**), ou plutôt l’injection de dépendance dans le constructeur est un terme ni GRASP ni Gang Of Four ; ce pattern en fait un peu plus, par contre il utilise l’équivalent du pattern GRASP « protection des variation » et l’équivalent du pattern « Strategy » du « Gang Of Four ».

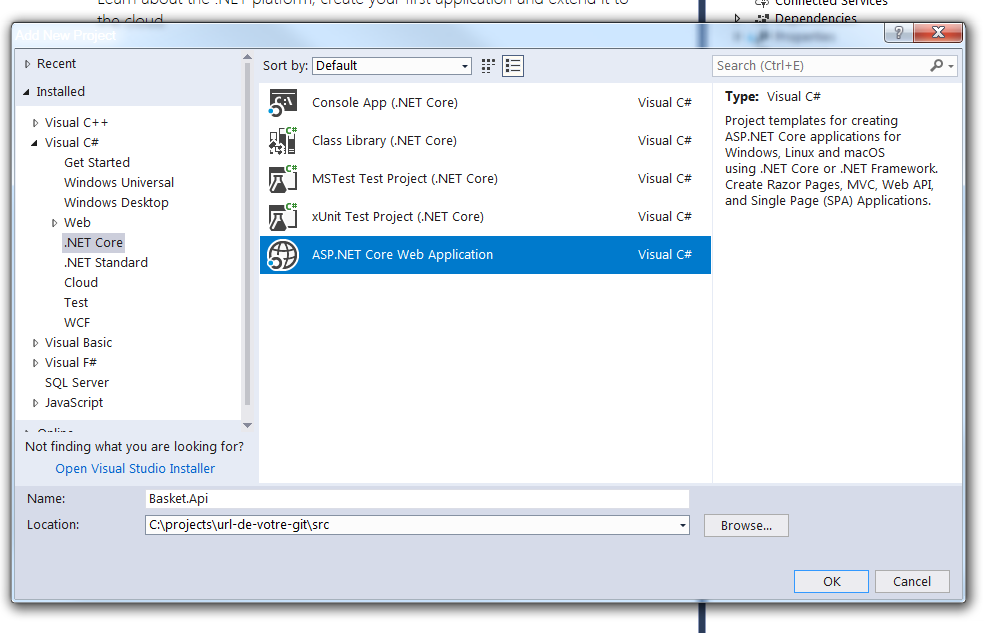
De nombreux noms de pattern apparaissent encore aujourd’hui. En générale il s’agit de nouveau nom qui duplique le nom d’un pattern qui existe déjà, ou il s’agit d’un pattern qui commune plusieurs notions de pattern existant.

## Mise en place de l’API

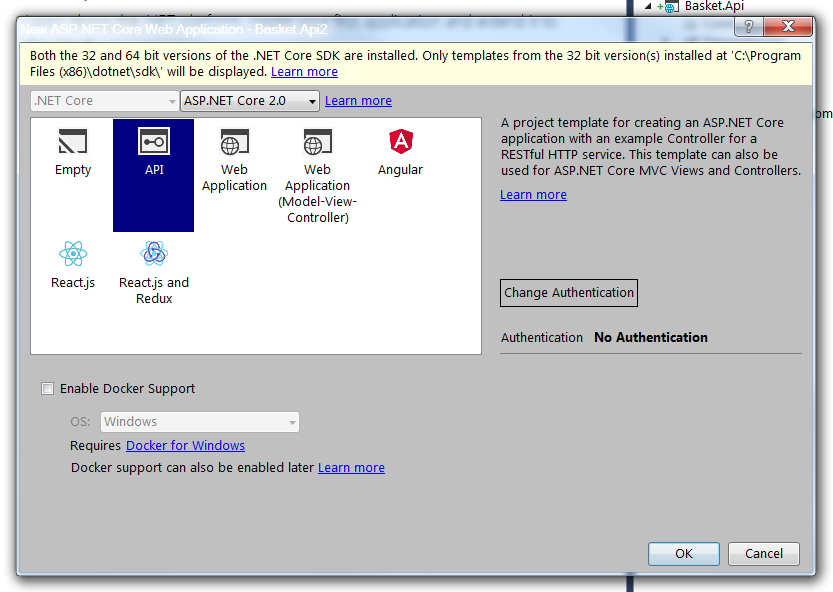
Réaliser un clic droit sur la solution afin d’ajouter un « nouveau projet ».



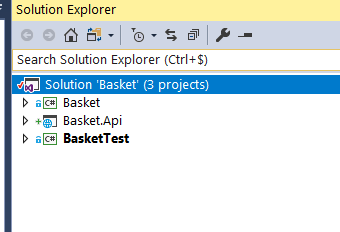
Créer un projet de type Web « net.core », nommer le « Basket.Api ».



Sélectionner le type API.



Vous allez maintenant avoir un projet « Basket.Api ».



Vous pouvez maintenant Remplacer le « ValuesController » par « BasketController »

|  |
| --- |
| namespace Basket.Api.Controllers  {  [Route("api/[controller]")]  public class BasketController : Controller  {  private BasketOperation \_basketOperation;  public BasketController(BasketOperation basketOperation)  {  \_basketOperation = basketOperation;  }  public class BasketModel  {  public List<BasketLineArticle> BasketLineArticles { get; set; }  }  public class BasketResult  {  public int Total { get; set; }  }  [HttpPost]  public BasketResult Post([FromBody]BasketModel basketModel)  {  return new BasketResult() { Total = \_basketOperation.CalculateAmount(basketModel.BasketLineArticles) };  }  } |

**Basket.Api/Controllers/BasketController.cs**

**Remarque :** on encapsule les informations d’entrée/sortie dans un objet, ce qui est un bonne pratique (plus de sécurité et flexibilité)

ASP.NET Core possède un moteur d’injection de dépendance intégré. Celui-ci a besoin de réel objet pour fonctionner. Si dans le TP précédent vous être arrivé jusqu’à la dernière question. Vous avez surement à un moment injecté une structure « DateTime ». Il vous faut donc l’encapsuler par une classe comme montrer dans le code ci-dessous :

|  |
| --- |
| public interface IDateTime  {  DateTime Now { get; }  } |

**Basket/IDateTime.cs**

|  |
| --- |
| public class MockDateTime : IDateTime  {  private readonly DateTime \_dateTime;  public MockDateTime(DateTime dateTime)  {  \_dateTime = dateTime;  }  public DateTime Now  {  get { return \_dateTime; }  }  } |

**Basket/MockDateTime.cs**

|  |
| --- |
| public class SystemDateTime : IDateTime  {  public DateTime Now  {  get { return DateTime.Now; }  }  } |

**Basket/SystemDateTime.cs**

Pour finir de configurer le moteur d’injection, ajouter les lignes ci-dessous afin de prévenir le moteur que quand dans un constructeur, une classe s’attente à recevoir « Basket.IDateTime » ; il va alors injecter dans le constructeur une instance de « Basket.SystemDateTime ».

|  |
| --- |
| public void ConfigureServices(IServiceCollection services)  {  services.AddMvc();  services.AddScoped<Basket.IDateTime, Basket.SystemDateTime>();  services.AddScoped<IArticleDatabase, ArticleDatabaseJson>();  services.AddScoped<ArticleFactory, ArticleFactory>();  services.AddScoped<ILogger, LoggerExternal>();  services.AddScoped<BasketService, BasketService>();  services.AddScoped<BasketOperation, BasketOperation>();  } |

**Basket.Api/Startup.cs**

## Tester manuellement votre API => swagger

Pour être sûr que votre API Fonctionne, vous aller ajouter une auto génération de swagger. Swagger est en quelque sorte une interface graphique qui permet d’exposer une documentation testable de son API a d’autre équipe.

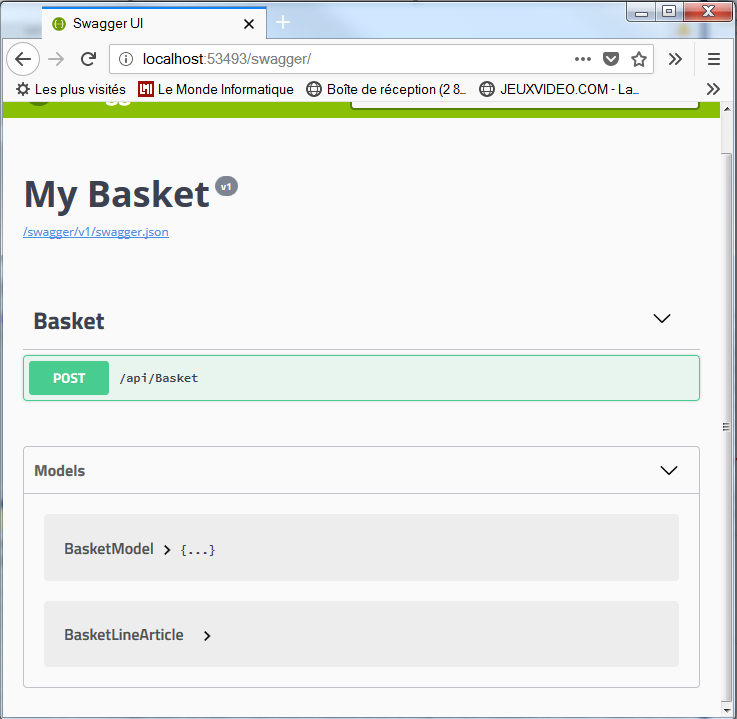
|  |
| --- |
| <Project Sdk="Microsoft.NET.Sdk.Web">  <PropertyGroup>  <TargetFramework>netcoreapp2.0</TargetFramework>  <ProjectUISubcaption>My Basket</ProjectUISubcaption>  </PropertyGroup>  <ItemGroup>  <Folder Include="wwwroot\" />  </ItemGroup>  <ItemGroup>  <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.0" />  <PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Design" Version="2.0.4" />  <PackageReference Include="Swashbuckle.AspNetCore" Version="2.4.0" />  </ItemGroup>  <PropertyGroup Condition="'$(Configuration)|$(Platform)'=='Debug|AnyCPU'">  <DocumentationFile>bin\Debug\$(TargetFramework)\$(MSBuildProjectName).xml</DocumentationFile>  <NoWarn>1701;1702;1705;1591</NoWarn>  </PropertyGroup>    <ItemGroup>  <DotNetCliToolReference Include="Microsoft.DotNet.Watcher.Tools" Version="2.0.0" />  </ItemGroup>  <ItemGroup>  <ProjectReference Include="..\Basket\Basket.csproj" />  </ItemGroup>  </Project> |

**Basket.Api/Basket.Api.csproj**

|  |
| --- |
| public class Startup  {  public Startup(IConfiguration configuration)  {  Configuration = configuration;  }  public IConfiguration Configuration { get; }  // This method gets called by the runtime. Use this method to add services to the container.  public void ConfigureServices(IServiceCollection services)  {  services.AddMvc();  services.AddSwaggerGen(c =>  {  c.SwaggerDoc("v1", new Info { Title = "My Basket", Version = "v1" });  c.IncludeXmlComments(  $@"{AppDomain.CurrentDomain.BaseDirectory}\Basket.Api.xml");  });  services.AddScoped<Basket.IDateTime, Basket.SystemDateTime>();  services.AddScoped<IArticleDatabase, ArticleDatabaseJson>();  services.AddScoped<ArticleFactory, ArticleFactory>();  services.AddScoped<ILogger, LoggerExternal>();  services.AddScoped<BasketService, BasketService>();  services.AddScoped<BasketOperation, BasketOperation>();  }  // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.  public void Configure(IApplicationBuilder app, IHostingEnvironment env)  {  if (env.IsDevelopment())  {  app.UseDeveloperExceptionPage();  }  app.UseSwagger();  app.UseSwaggerUI(c => { c.SwaggerEndpoint("/swagger/v1/swagger.json", "My Basket"); });  app.UseMvc();  }  } |

**Basket.Api/Startup.cs**

Lorsque vous lancer votre API maintenant, vous avez accès à une interface swagger qui vous permet de tester que votre api fonctionne correctement. Cette interface est autogénéré à partir de votre code.



<http://localhost:53493/swagger/>

## Mise en place de l’interface web

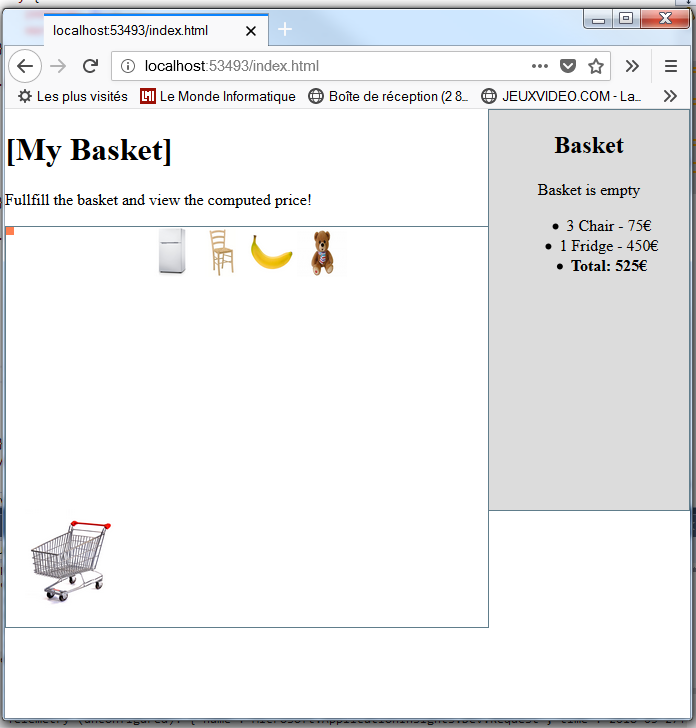
Votre collègue « Hubert » a commencé à réaliser une interface graphique en html/javascript (es6) sans utiliser de Framework. Coup de chance ou pas pour vous, il est parti en vacances pendant 2 mois. Vous reprenez ainsi son développement qui n’est pas terminé. Le code est téléchargeable à cette adresse :

<https://github.com/guillaumechervet-teacher/introduction-guillaumechervet.git>

Positionner ce code dans le répertoire « wwwroot » de votre API.

|  |
| --- |
| // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.  public void Configure(IApplicationBuilder app, IHostingEnvironment env)  {  if (env.IsDevelopment())  {  app.UseDeveloperExceptionPage();  }  app.UseSwagger();  app.UseSwaggerUI(c => { c.SwaggerEndpoint("/swagger/v1/swagger.json", "My Basket"); });  app.UseStaticFiles();  app.UseMvc();  } |

**Basket.Api/Startup.cs**



<http://localhost:53493/index.html>

Vous constatez que le code existant est plus ou moins propre. Seul le composant « réfrigérateur » semble réagir au clic de souris. De plus, votre collègue a eu la mauvaise idée de ne pas utiliser de Framework JavaScript tel que (React, Angular, Vue.js ou Elm). Vous n’avez pas le temps de tout changer, vous allez devoir faire avec.

## Faire communiquer les composants sans qu’ils se connaissent => « Publish/Subscribe » équivalent à « Mediator »

Vous remarquez que le composant « PanelBasket » et « PanelArticle » ne possèdent visuellement pas de lien, par contre dans le code ils sont couplés, le « PanelArticle » possède une référence de « PanelBasket ». Cela vous pose problème car vous voudriez que chaque composant soit simple à maintenir, réutilisable et autonome.

**A vous d’appliquer le bon pattern afin de faire en sorte que les 2 composants communiquent entre eux sans pour autant se connaître.**

On vous demande de créer un nouveau composant qui va afficher en temp réel les coordonnée X et Y de la souris à l’intérieur du composant « PanelArticle ». Ce composant pourra être mis en dessous des autres.   
Le html doit ressembler à cela : <p id="coordinatePanel">Panel (x,y)</p>

X et Y doivent être remplacé par les coordonnées de la souris dans le « PanelArticle » en pixel. Si la souris sort du panel, les dernières cordonnées sont affichées et ne sont pas mise à jour.  
**A vous de réaliser le code sans créer de couplage.**

## Grumly joue de la musique => Ajoute de comportement => Polymorphisme

A vous maintenant de faire en sorte que le jouet « Grumly » puisse être ajouté au panier. Une petite règle est demandée. La sélection de « Grumly » doit déclencher une musique au format mp3 (musique de votre choix). Sa désélection doit éteindre la musique.

N’oubliez pas google : rechercher par exemple « sample mp3 » et « html5 mp3 »

## Les bananes laissent des traces

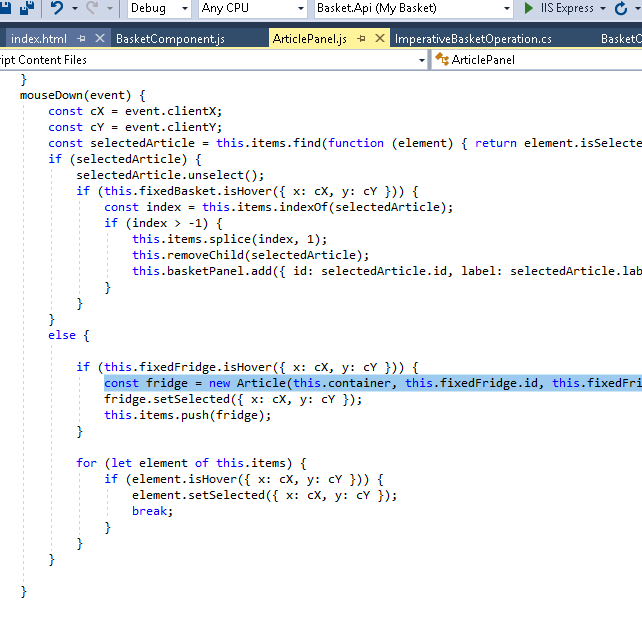
Le déplacement des bananes laisse des traces de saleté derrière elles. Les traces peuvent par exemple être des petits carrés dont la représentation html est :

<div style="background-color:coral; width:8px;height:8px;position:absolute;"></div>

Une fois qu’une « banane » est ajouté au panier, toutes ces traces associées disparaissent.

## Netoyage de code => Prototype

Vous souhaitez nettoyer un peu le code et mettre en place le pattern « Prototype » sur la ligne surlignée dans l’image ci-dessous.



**A vous de réaliser le code.**

## Undo/Redo

C’est une demande métier courante dans les interfaces graphiques. Nous souhaitons ajouter un composant en bas de la page qui permettra de faire de « undo/redo » sur les actions du panier (uniquement du panier). Ce composant bien sûr, ne sera pas couplé avec le « basketPanel » et ainsi sera un entièrement réutilisable pour tout autre besoin similaire. Le html du composant :

<div id="undoRedoPanel">

<button>Undo</button>

<button>Redo</button>

</div>

Il n’est pas possible de faire « Undo » si aucune action a été joué.  
Il n’est pas possible de faire « Redo » si toutes les actions ont été jouées.

## Ou sont les tests unitaires ?

Vous avez développé sans faire de tests unitaires. Vous n’avez pas fait de TDD (Test Driven Développement). Cela vous a permis de monter en compétence afin de comprendre comment tous les éléments graphiques s’agences (en effet, il est difficile voire impossible de faire du TDD sans avoir une vision globale des Frameworks que vous utilisez, dans notre cas ; le Framework est le DOM html/javascript natif des browsers web).

Vous aimerez ajouter des tests unitaires afin de vous assurer que par la suite, vous ne ferez pas de régression.

A vous de vous débrouiller afin de mettre en place Jest :

<https://facebook.github.io/jest/>

Pour bien tester le code, il faut qu’il soit testable. Vous pouvez lire cet article qui est très bien expliqué :

<https://medium.com/javascript-scene/mocking-is-a-code-smell-944a70c90a6a>

Vous allez probablement devoir modifier votre code pour le rendre plus propre/testable. Par exemple, si vous souhaitez tester la méthode « isHover ».

|  |
| --- |
| import Component from "./Component.js";  export default class BasketComponent extends Component {  constructor(parentContainer, position) {  const size = { width: 100, height: 100 };  const image = document.createElement('img');  image.src = "./img/basket.jpg";  image.style.width = size.width + "px";  image.style.height = size.height + "px";  if (position) {  image.style.position = "absolute";  image.style.top = position.top + "px";  image.style.left = position.left + "px";  }    super(parentContainer, image);  this.size = size;  }  isHover(point) {  const offsets = this.container.getBoundingClientRect();  const x = offsets.left;  const y = offsets.top;  const width = this.size.width;  const height = this.size.height;  var isX = false;  if (x < point.x && point.x < (x + width)) {  isX = true;  }  var isY = false;  if (y < point.y && point.y < (y + height)) {  isY = true;  }  return isX && isY;  }  } |

**Basket.Api/wwwroot/BasketComponent.js**

Vous pouvez transformer le code en :

|  |
| --- |
| import Component from "./Component.js";  export const isHoverFunc = (getBoundingClientRect, size) => (point) => {  const offsets = getBoundingClientRect();  const x = offsets.left;  const y = offsets.top;  const width = size.width;  const height = size.height;  var isX = false;  if (x < point.x && point.x < (x + width)) {  isX = true;  }  var isY = false;  if (y < point.y && point.y < (y + height)) {  isY = true;  }  return isX && isY;  }  export default class BasketComponent extends Component {  constructor(parentContainer, position) {  const size = { width: 100, height: 100 };  const image = document.createElement('img');  image.src = "./img/basket.jpg";  image.style.width = size.width + "px";  image.style.height = size.height + "px";  if (position) {  image.style.position = "absolute";  image.style.top = position.top + "px";  image.style.left = position.left + "px";  }    super(parentContainer, image);  this.size = size;  }  isHover(point) {  var getBoundingClientRect = this.container.getBoundingClientRect.bind(this.container);  return isHoverFunc(getBoundingClientRect, this.size)(point);  }  } |

**Basket.Api/wwwroot/BasketComponent.js**

Vous pourrez ainsi tester la fonction « isHoverFunc » comme dans l’exemple ci-dessous.

|  |
| --- |
| import { isHoverFunc } from './BasketComponent';  test('isHoverFunc shoud return true only when point is hover', () => {  const getBoundingClientRect = () => { left: 100, top: 100 };  const size = { width: 200, height: 200 };  let point = { x: 0, y: 0 };  let isHover = isHoverFunc(getBoundingClientRect, size)(point);  expect(isHover).toBe(false);  point = { x: 105, y: 110 };  isHover = isHoverFunc(getBoundingClientRect, size)(point);  expect(isHover).toBe(true);  }); |

**Basket.Api/wwwroot/BasketComponent.spec.js**

**Ajouter maintenant un test unitaire sur la méthode « ArticleComponent.move » puis « ArticleComponent.setSelected ».**

## Clean code

Dans le code il y a beaucoup de de duplication, de fonctions volumineuses. Rendez le code plus lisible. Votre collègue « Hubert » sera ainsi prêt à reprendre votre code !