

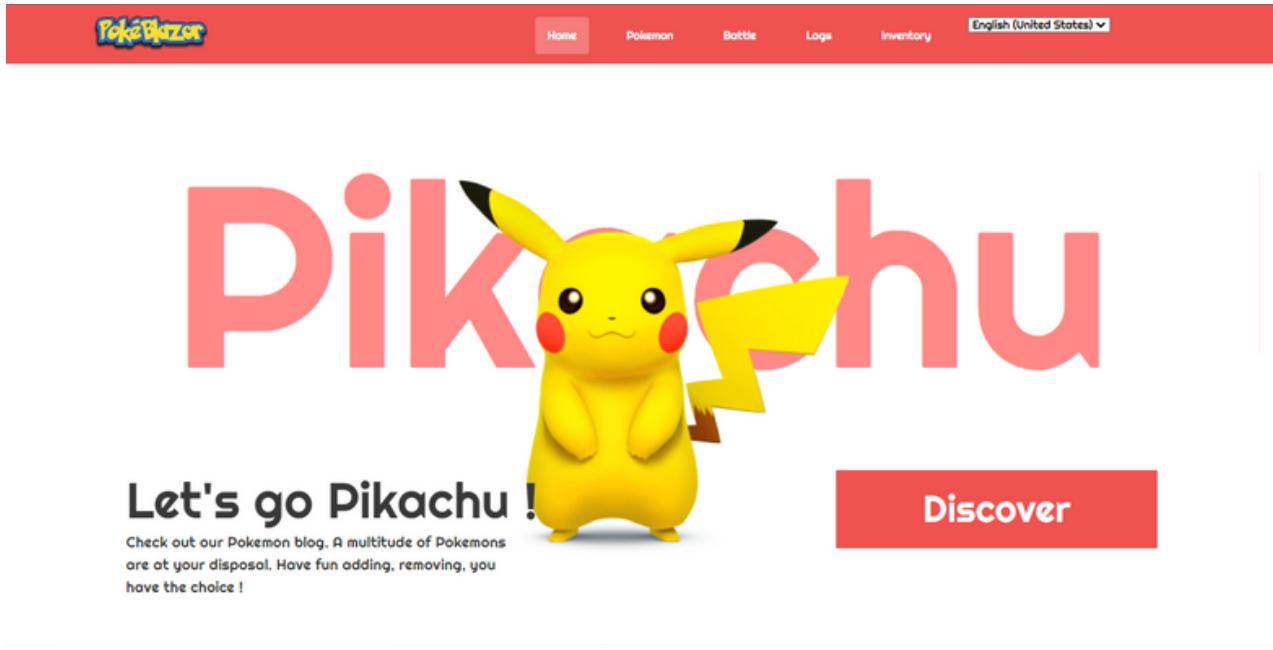


PROJET : POKEBLAZOR

2022 / 2023

PRÉPARÉ PAR
BENJELLOUN OTHMANE
BRUNO DA COSTA CUNHA

1/ Page d'accueil



La page d'accueil est la première page sur laquelle on tombe lorsqu'on a suivi les étapes du **ReadMe**.

Cette page est constituée en haut de la **navbar** située dans le fichier **Shared/NavMenu.razor**

```
<div class="nav-items">
    <div class="nav-item px-3">
        <a href="">
            <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
                @Localizer["Home"]
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <a href="list">
                <NavLink class="nav-link" href="list">
                    @Localizer["Pokemon"]
                </NavLink>
            </div>
            <div class="nav-item px-3">
                <a href="bataille">
                    <NavLink class="nav-link" href="bataille">
                        @Localizer["Bataille"]
                    </NavLink>
                </div>
                <div class="nav-item px-3">
                    <a href="logs">
                        <NavLink class="nav-link" href="logs">
                            @Localizer["Logs"]
                        </NavLink>
                    </div>
                    <div class="nav-item px-3">
```

Grâce aux différents **NavLink**, nous pouvons accéder aux différentes pages que nous allons expliquer par la suite.

Cette page **d'accueil** permet uniquement d'accéder aux autres pages via la **navbar** ou via le bouton "Discover" pour accéder à la **liste des Pokémons**.

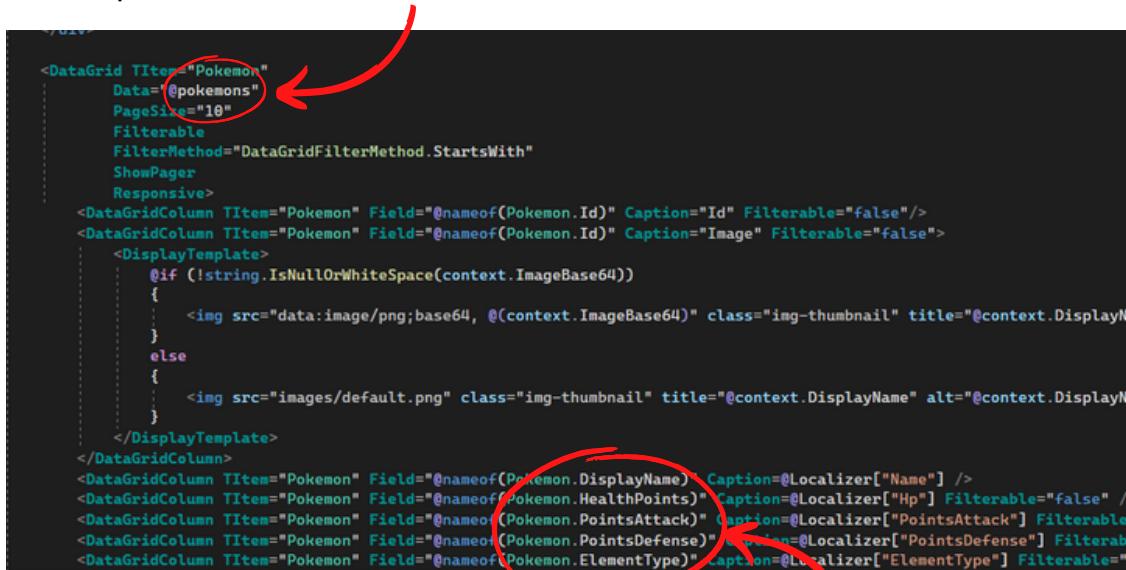
2 / Liste des Pokemons

Une fois dans la page de la liste des Pokemons. C'est la méthode suivante qui est lancée.

```
protected override async Task OnInitializedAsync()
{
    int size = await DataService.Count();
    pokemons = await DataService.List(0, size);
}
```

Elle va permettre de connaître le nombre de pokemons ainsi que de charger tout les Pokémons dans la liste "pokemons". Cette méthode va faire appel à **DataService** qui va appeler l'**api** pour avoir accès au fichier JSON "pokemons.json" qui contient tout les Pokémons et leur informations.

Ils sont ensuite listé via le DataGrid où on passe en paramètre les Pokémons qu'on a récupéré.



Les colonnes du DataGrid sont constituées des **différents attributs** de l'objet "Pokemon" que l'on a défini dans **models/Pokemon.cs**.

```

namespace BlazorPokemon.Models
{
    public class Pokemon
    {
        public int Id { get; set; }

        public string DisplayName { get; set; }

        public string Name { get; set; }

        public int HealthPoints { get; set; }

        public int PointsAttack { get; set; }

        public int PointsDefense { get; set; }

        public List<string> ElementType { get; set; }

        public DateTime CreatedDate { get; set; }

        public DateTime? UpdatedDate { get; set; }

        public byte[] ImageContent { get; set; }

        public string ImageBase64 { get; set; }

        public static int compareType(Pokemon p1, Pokemon p2)
    }
}

```

Models/Pokemon.cs

Dans cette page de liste de Pokémons, nous pouvons également **Ajouter**, **éditer** et **supprimer** des Pokémons.

Lorsqu'on clique sur le bouton **Ajouter**. Nous nous retrouvons dans la page **pages/add.razor** qui va dans un premier temps instancier un **pokemonModel** situé dans le fichier **models/pokemonModel.cs**.

```

    /// <summary>
private PokemonModel pokemonModel = new()
{
    ElementType = new List<string>()
};

using System.ComponentModel.DataAnnotations;

namespace BlazorPokemon.Models
{
    public class PokemonModel
    {
        public int Id { get; set; }

        [Required]
        [StringLength(50, ErrorMessage = "Le nom affiché ne doit pas dépasser 50 caractères.")]
        public string DisplayName { get; set; }

        [Required]
        [StringLength(50, ErrorMessage = "Le nom ne doit pas dépasser 50 caractères.")]
        [RegularExpression(@"^a-zA-Z-\s{1,40}$", ErrorMessage = "Seulement les caractères en minuscule sont acceptées.")]
        public string Name { get; set; }

        [Required]
        [Range(0, 100)]
        public int healthPoints { get; set; }

        [Required]
        [Range(10, 100)]
        public int PointsAttack { get; set; }

        [Required]
        [Range(10, 100)]
        public int PointsDefense { get; set; }

        [Required(ErrorMessage = "Le type du pokémon est obligatoire")]
        public List<string> ElementType { get; set; }
    }
}

```

Models/PokemonModel.cs

La page nous présente différents champs qui, une fois **remplis** et **verifiés** vont être **binder aux attributs du pokemonModel**.

```

        @Localizer["Name"]
        <InputText id="name" @bind-Value="pokemonModel.Name" />
    
```

Une fois tout les champs remplis et vérifiés. L'utilisateur peut cliquer sur le bouton envoyer qui va appeler la méthode **HandleValidSubmit()**.

Cette méthode va appeler la méthode **Add** de **DataService**. Le code de la méthode **Add** est défini dans **Data ApiService.cs** (Hérite de **DataService**). **Add()** va appeler **PokemonFactory()** qui va prendre en paramètre le modèle et va retourner un **Pokémon**. Ce même **Pokémon** va ensuite être serialiser dans le fichier json de l'api **pokemon.json**.

```
1 référence
private async void HandleValidSubmit()
{
    await DataService.Add(pokemonModel);
    NavigationManager.NavigateTo("list");
}

2 références
public async Task Add(PokemonModel model)
{
    // Get the Pokemon
    var pokemon = PokemonFactory.Create(model);

    // Save the data
    await _http.PostAsJsonAsync("https://localhost:7234/api/Pokemon/", pokemon);
}

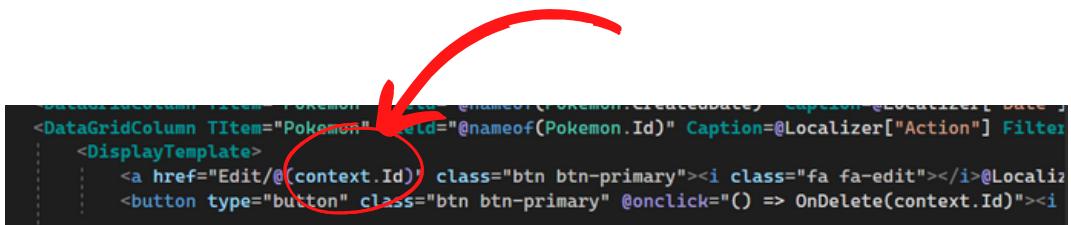
3 références
public static Pokemon Create(PokemonModel model)
{
    return new Pokemon
    {
        Id = model.Id,
        DisplayName = model.DisplayName,
        Name = model.Name,
        HealthPoints = model.HealthPoints,
        PointsAttack = model.PointsAttack,
        PointsDefense = model.PointsDefense,
        ElementType = model.ElementType,
        CreatedDate = DateTime.Now,
        ImageBase64 = Convert.ToBase64String(model.ImageContent)
    };
}
```

L'utilisateur est ensuite redirigé vers la page **ListePokemon.razor**

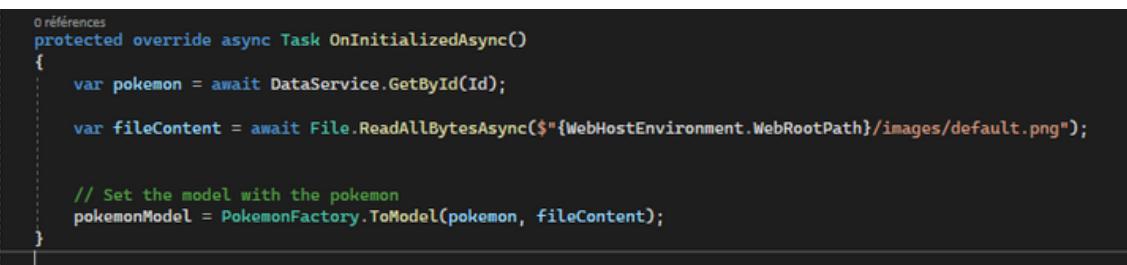
```
1 référence
private async void HandleValidSubmit()
{
    await DataService.Add(pokemonModel);
    NavigationManager.NavigateTo("list");
}
```

2. éditer un Pokemon

Le fonctionnement de la page éditer est semblable à celui d'ajouter. Lorsque l'utilisateur clique sur éditer, l'**id** du Pokémon est passé en paramètre du **href**.

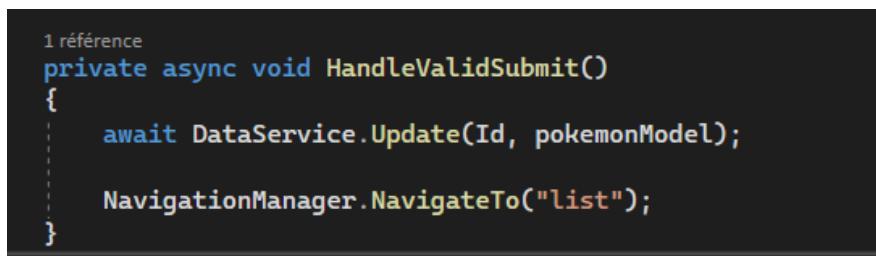


Nous sommes ensuite dirigés vers la page **Editer**. Un **pokemonmodel** est créée. La méthode **OnInitializedAsync()** est lancée.



Cette méthode va créer le **pokemon** dont l'**Id** est passé en paramètre en appelant la fonction GetById de DataService qui va **déserialiser** le fichier pour créer le Pokémon et le mettre dans la variable pokemon. Un model est ensuite créé grâce à la méthode **ToModel** de PokemonFactory.

Une fois que l'utilisateur a remplis les champs et a appuyer sur le bouton **envoyer**. Les valeurs entrées sont bindées avec les attribut de Pokemon model et la méthode **HandleValidSubmit()** est lancée. Cette méthode va appeler la méthode **update** de DataAPIService qui va prendre en paramètre le modèle et l'id du Pokémon



L'utilisateur est ensuite redirigé vers la page **ListePokemon.razor**

3. Supprimer un Pokemon

Une fois dans la page **ListePokemon.razor**. L'utilisateur a le choix de supprimer des Pokémons en appuyant sur le bouton supprimer mis à disposition à côté de chaque Pokémon.



1		Bulbizarre	90	20	30	Plante, Eau	26/12/2022	<button>Edit</button> Delete
2		Dracaufeu	100	90	60	Dragon, Feu	26/12/2022	<button>Edit</button> Delete
3		Corapuce	100	20	45	Eau	26/12/2022	<button>Edit</button> Delete

Une fois le bouton cliqué. La méthode **OnDelete** est lancée, avec en paramètre l'id du Pokémon.

```
1 référence
private async void OnDelete(int id)
{
    var parameters = new ModalParameters();
    parameters.Add(nameof(Pokemon.Id), id);

    var modal = Modal.Show<DeleteConfirmation>("Delete Confirmation", parameters);
    var result = await modal.Result;

    if (result.Cancelled)
    {
        return;
    }

    await DataService.Delete(id);

    // Reload the page
    NavigationManager.NavigateTo("list", true);
}
```

Cette méthode va appeler **DeleteConfirmation** dans **Modals/DeleteConfirmation.razor** qui va demander la confirmation à l'utilisateur de supprimer ou non.

```
[Inject]
1 référence
public IDataService DataService { get; set; }

[Parameter]
1 référence
public int Id { get; set; }

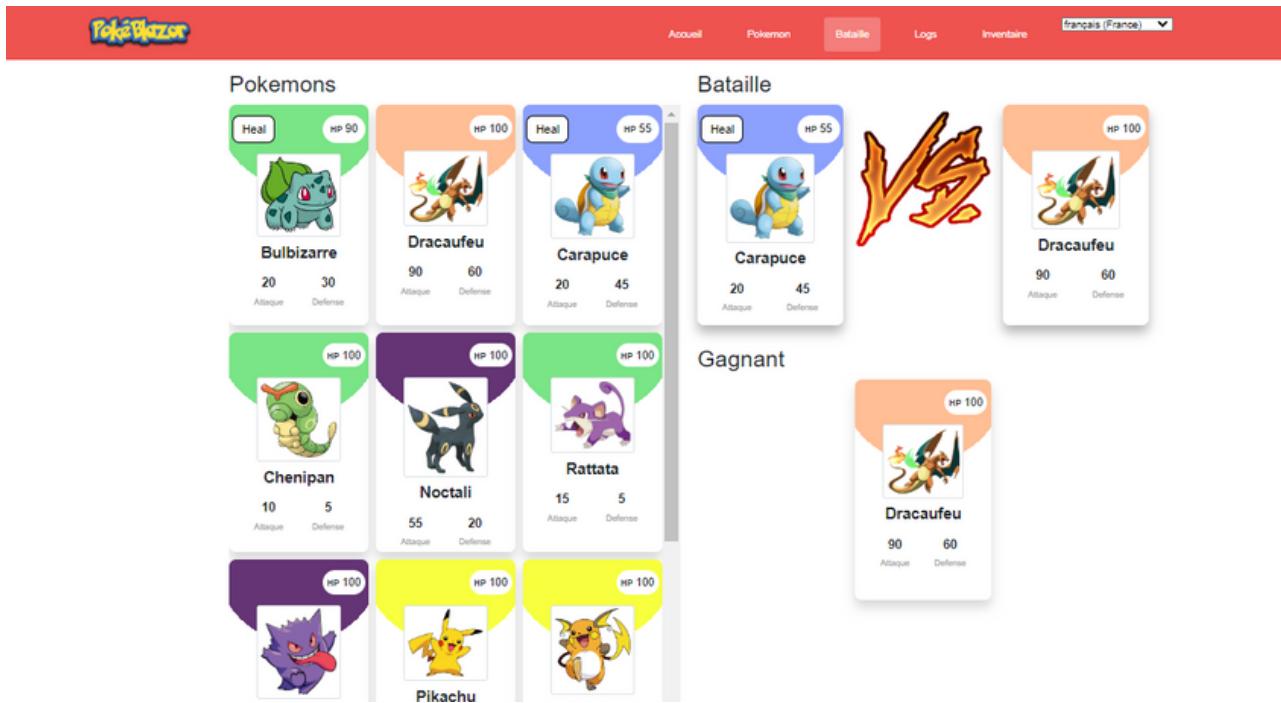
private Pokemon pokemon = new Pokemon();

0 références
protected override Task OnInitializedAsync()
{
    // Get the pokemon
    pokemon = await DataService.GetById(Id);
}

1 référence
void ConfirmDelete()
{
    ModalInstance.CloseAsync(ModalResult.Ok(true));
}

1 référence
void Cancel()
{
    ModalInstance.CancelAsync();
}
```

3 / Bataille



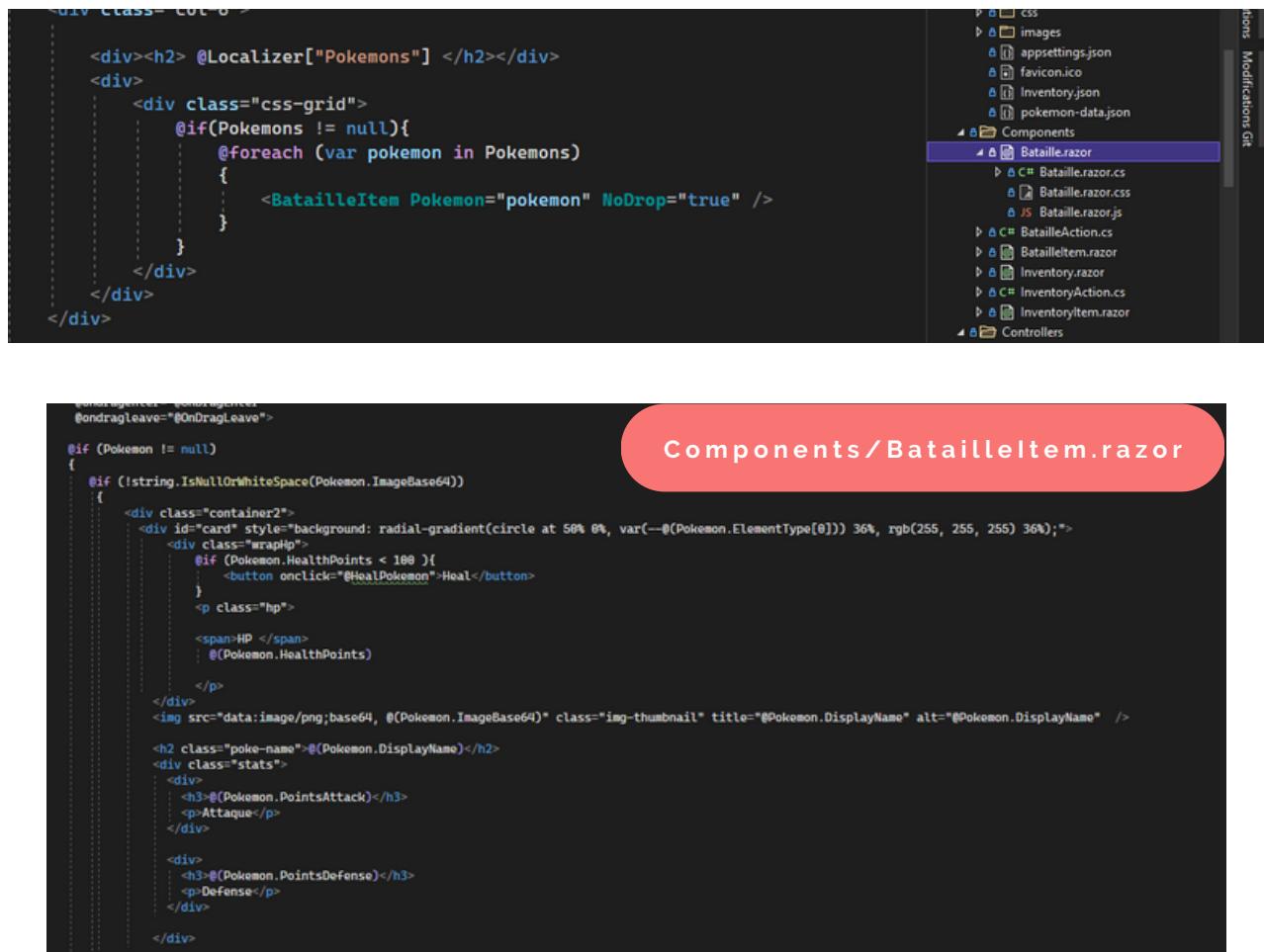
Dans la page **Bataille**, l'utilisateur peut mettre aux combat ses différents Pokémons. C'est en fonction de leur points de vie, leurs point d'attaque ainsi que leurs points de défense que le gagnant de la bataille est déterminé. Le perdant de la bataille perd un certain nombre de **point de vie** en fonction des **points d'attaque** du gagnant ainsi que de ses propres **points de défense**.

Les Pokémons ayant moins de 100 points de vie peuvent être soignés (grâce au bouton Heal) pour restituer leurs points de vie à 100.

En ce qui concerne le code. Des que la page est lancée, la méthode **OnInitializedAsync** est lancée. Cette dernière va appeler la méthode **All** de **Data ApiService** qui va accéder au fichier JSON de l'api pour **déserialiser** tout les Pokémons et les retourner pour les stocker dans la variable **Pokemons**.

```
protected override async Task OnInitializedAsync()
{
    Pokemons = await DataService.All();
}
```

Les Pokémons seront ensuite listés sur la page **Bataille.razor** qui va parcourir tout les Pokémons dans la variable **Pokemons** et les lister en faisant appel au fichier **BattleItem.razor**.



The screenshot shows the Visual Studio Code interface with the following details:

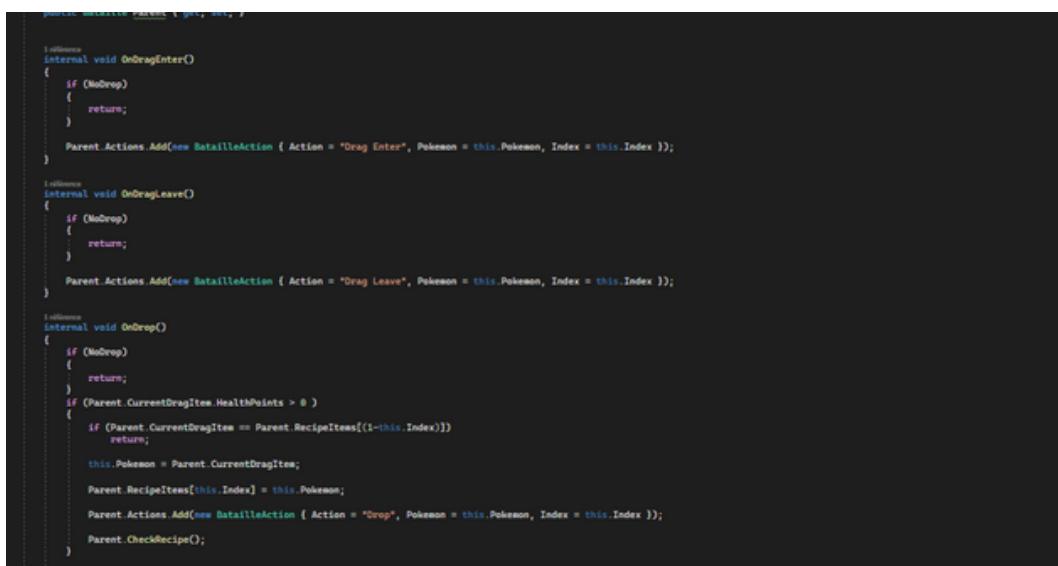
- File Explorer:** Shows the project structure with files like CSS, images, appsettings.json, favicon.ico, Inventory.json, pokemon-data.json, Components, Bataille.razor, BatailleAction.cs, BatailleItem.razor, Inventory.razor, InventoryAction.cs, InventoryItem.razor, and Controllers.
- Code Editor:** Displays the **BatailleItem.razor** component code. It includes a CSS class definition for `.grid`, a `<div>` block containing a `<h2>` element with the text `@Localizer["Pokemons"]`, and a `<div>` block with a `@if` condition. Inside this block, there's a `@foreach` loop that iterates over `Pokemons`. For each iteration, it generates a `<BattleItem Pokemon="pokemon" NoDrop="true" />` element.
- Search Bar:** Contains the text **Components/BattleItem.razor**.

```


<div><h2> @Localizer["Pokemons"] </h2></div>
    <div>
        <div class="css-grid">
            @if(Pokemons != null){
                @foreach (var pokemon in Pokemons)
                {
                    <BattleItem Pokemon="pokemon" NoDrop="true" />
                }
            }
        </div>
    </div>
</div>


```

Le fichier **BattleItem.cs** contient toutes les méthodes permettant le drag and drop.



The screenshot shows the **BattleItem.cs** code, which contains three methods: `OnDragEnter`, `OnDragLeave`, and `OnDrop`. These methods handle the logic for dragging and dropping Pokémons, including updating health points and managing actions.

```

public void OnDragEnter(DragEventArgs e)
{
    if (NoDrop)
    {
        return;
    }

    Parent.Actions.Add(new BattleAction { Action = "Drag Enter", Pokemon = this.Pokemon, Index = this.Index });
}

public void OnDragLeave(DragEventArgs e)
{
    if (NoDrop)
    {
        return;
    }

    Parent.Actions.Add(new BattleAction { Action = "Drag Leave", Pokemon = this.Pokemon, Index = this.Index });
}

public void OnDrop(DragEventArgs e)
{
    if (NoDrop)
    {
        return;
    }

    if (Parent.CurrentDragItem.HealthPoints > 0 )
    {
        if (Parent.CurrentDragItem == Parent.RecipeItems[(Index - this.Index)])
            return;

        this.Pokemon = Parent.CurrentDragItem;

        Parent.RecipeItems[this.Index] = this.Pokemon;

        Parent.Actions.Add(new BattleAction { Action = "Drop", Pokemon = this.Pokemon, Index = this.Index });

        Parent.CheckRecipe();
    }
}

```

Le gagnant de la bataille est déterminé grâce à la fonction **compareType** qui est appelé dans la fonction **CheckRecipe**.

```
public static int compareType(Pokemon p1, Pokemon p2)
{
    if (p1.getPower() > p2.getPower()) {
        if(p2.PointsDefense > p1.PointsAttack){ // malgré que P1 gagne, P2 a plus de défense que P1 a d'attaque. P2 recois donc 50% de l'attaque de p1.
            p2.HealthPoints = p2.HealthPoints - (int)(p1.PointsAttack*0.5);
            if (p2.HealthPoints <= 0)
            {
                p2.HealthPoints = 0;
            }
            return 0;
        }

        p2.HealthPoints = p2.HealthPoints + p2.PointsDefense - p1.PointsAttack;
        if (p2.HealthPoints <= 0)
        {
            p2.HealthPoints = 0;
        }
        return 0;
    }
    else{
        if(p1.PointsDefense > p2.PointsAttack){ // malgré que P2 gagne, P1 a plus de défense que P2 a d'attaque. P1 recois donc 50% de l'attaque de p2.
            p1.HealthPoints = p1.HealthPoints - (int)(p2.PointsAttack*0.5);
            if (p1.HealthPoints <= 0)
            {
                p1.HealthPoints = 0;
            }
            return 1;
        }

        p1.HealthPoints = p1.HealthPoints + p1.PointsDefense - p2.PointsAttack;
        if (p1.HealthPoints <= 0)
        {
            p1.HealthPoints = 0;
        }
        return 1;
    }
}
```

```
1 référence
public async void CheckRecipe()
{
    RecipeResult = null;

    if (RecipeItems[0]!=null && RecipeItems[1] != null)
    {
        int winner = Pokemon.compareType(RecipeItems[0], RecipeItems[1]);
        await DataService.UpdateLoser(RecipeItems[1-winner].Id, RecipeItems[1-winner]);
        RecipeResult = RecipeItems[winner];

        if (RecipeItems[1 - winner].HealthPoints <= 0)
        {
            RecipeItems[1 - winner] = null;
            refPokemon[1 - winner].Pokemon = null;
            RecipeResult = null;
        }
    }
    else
    {
        return;
    }
}
```

4 / Inventaire

The screenshot shows the 'Inventaire' (Inventory) page of the PokeBlazer application. At the top, there's a navigation bar with links for Accueil (Home), Pokemon, Bataille (Battle), Logs, and Inventario (Inventory). A language dropdown shows 'français (France)'. The main content area has a header 'Equipe' (Team) with a 3x3 grid of icons representing the current team members. Below this is a table listing 15 items (Pokemons) from index 1 to 15:

Index	Image	Nom
1	Bulizane	
2	Dracaufeu	
3	Carapuce	
4	Chenapan	
5	Noctali	
6	Rattata	
7		
8		
9		
10		
11		
12		
13		
14		
15		

Below the table are navigation buttons: Premier, Précédent, 1, 2, 3, Suivant, Dernier, and a note '1 - 6 de 15 items'. At the bottom, a 'Actions' section lists several drag-and-drop events:

```
Action: Drag Start - Index: 0  
Action: Drag Enter - Index: 0  
Action: Drop - Index: 0  
Action: Drag Start - Index: 0  
Action: Drag Enter - Index: 3  
Action: Drag Leave - Index: 0  
Action: Drag End - Index: 3
```

La page "Inventaire" affiche la liste complète des **Pokémons** de la même manière que la page "Bataille". Cette page permet à l'utilisateur de constituer son équipe de Pokémons. Bien entendu, l'équipe reste enregistrée même si l'utilisateur actualise la page. Le système de "Drag and Drop" est similaire à celui utilisé sur la page "Bataille".

Pour afficher la liste des **Pokémons** dans l'inventaire, nous appelons l'API et utilisons la fonctionnalité "drag and drop" pour ajouter des Pokémons à l'inventaire. Nous pouvons utiliser la fonction "dragend" pour retirer un Pokémon de l'inventaire, et nous enregistrons les modifications dans **un fichier JSON**.