ÉTAPE 7 : VALIDATION

Sur le web, toutes les données doivent être validées à un niveau serveur pour préserver la cohérence des données (anticorruption layer) et la sécurité

ASP.NET dispose d'un système de validation intégré mais qui montre vite ses limites

Au lieu de cela, installer le package FluentValidation (dotnet add package FluentValidation)

Pour créer un validateur, créer une classe qui hérite de AbstractValidator<Model>
Dans le constructeur, implémenter les règles avec RuleFor(x => …)

Enregistrer le validator en tant que IValidator<Model> dans le conteneur de services et l'importer dans les endpoints concernés pour valider le modèle avec la méthode Validate

Valider et faire le nécessaire pour le modèle « AttackRequest »

Note : si un endpoint a plusieurs codes de retour, utiliser Results<Code1, Code2, etc.>

THÉORIE : GRPC

GRPC permet d'échanger des données au format binaire entre les différentes applications

Souvent utilisé pour les échanges entre microservices car le contenu est concis et les échanges réduits (haute performance), il est possible de l'utiliser sur le web

Les échanges par le protocole gRPC nécessite de définir un contrat de communication (un fichier .proto) qui sera utilisé par le deux parties pour être compris

En ASP.NET, le tooling génère beaucoup d'éléments de façon automatique pour simplifier les échanges

Installer le package gRPC.AspNetCore sur le projet d'API pour activer le fonctionnement gRPC

ÉTAPE 8 : COMMUNICATION GRPC

Nous allons communiquer avec GRPC entre l'API et l'application Blazor (tout en gardant les endpoints REST pour le débug)

Installer l'extension vscode-proto3 pour avoir une coloration syntaxique et une aide à la complétion

Créer un dossier de solution pour stocker les fichiers proto
Créer un fichier proto pour les requêtes et les réponses

Exemple:

```
syntax = "proto3";

import "google/protobuf/wrappers.proto";

message AttackRequestGRPC {
    int32 col = 1;
    int32 row = 2;
}
```

```
service BattleshipService {
    rpc Attack(AttackRequestGRPC) returns AttackResponseGRPC;
```

ÉTAPE 8 bis : COMMUNICATION GRPC (API)

Lorsque tous les contrats ont été produits, modifier le fichier csproj de l'API pour inclure les contrats protobuf :

```
<ItemGroup>
  <Protobuf Include="..\requests_responses.proto" GrpcServices="Server" />
</ItemGroup>
```

Si le contrat a bien été créé et que les liens sont faits, il est possible de créer un service qui hérite du service automatiquement généré par le tooling

Le service automatiquement généré reprend le nom du service avec Base à la fin :

```
0 references
public class BattleshipGRPCService : BattleshipService.BattleshipServiceBase
{
```

Il faudra overrider les méthodes pour donner une implémentation
Attention : les collections en GRPC doivent être remplie après la création

ÉTAPE 8 ter : COMMUNICATION GRPC (API)

Il faut ensuite mapper notre nouveau service gRPC au niveau de ce qu'ASP.NET gère

Pour ceci, il suffira simplement d'appeler MapGrpcService avec le service que l'on souhaite mapper

Note : il est possible de tester le service gRPC sans implémenter le client grâce à une extension de Visual Studio Code (gRPC Clicker)

Attention: GRPC nécessite HTTP2,
il faut configurer l'application dans le appsettings.json

Avec cette configuration, le port 5001 tourne
en HTTP 1 &HTTP2

Et le port 5224 en HTTP1 pour les endpoints « historiques »

```
"Kestrel": {
  "Endpoints": {
    "Http2": {
      "Url": "http://localhost:5001",
      "Protocols": "Http1AndHttp2"
    },
    "Http1":{
      "Url": "http://localhost:5224",
      "Protocols": "Http1"
    }
  }
}
```

ÉTAPE 9 : COMMUNICATION GRPC (CLIENT)

Les appels GRPC depuis le navigateur ne sont pas encore possible en l'état. Heureusement, l'équipe ASP.NET met à disposition un package permettant aux applications web de réaliser les appels GRPC depuis Blazor WASM

➔ Installer le package Google.Protobuf, Grpc.Net.Client.Web, Grpc.Tools & Grpc.Net.Client sur le projet Blazor

1. Ajouter le support du fichier proto dans le csproj (en mode Client cette fois) + la classe qui override le client de base
2. Ajouter le code dans les services pour créer et utiliser un client compatible GRPC mais conserver le code des services REST

```
builder.Services.AddScoped(sp =>
{
    var httpClient = new HttpClient(new GrpcWebHandler(GrpcWebMode.GrpcWeb, new
    HttpClientHandler()));
    var channel = GrpcChannel.ForAddress("http://localhost:5001", new
    GrpcChannelOptions { HttpClient = httpClient });
    return new BattleshipService.BattleshipServiceClient(channel);
});
```