# Préparation d’un environnement Angular

## Création d’un nouveau projet Angular

ng new MyFrontend

## Exécution de l’application Angular

cd MyFrontend

ng serve

# Préparation d’un environnement ASP.NET Core

## Création d’un nouveau projet ASP.NET Core Web API

## Ajout de packages NuGet

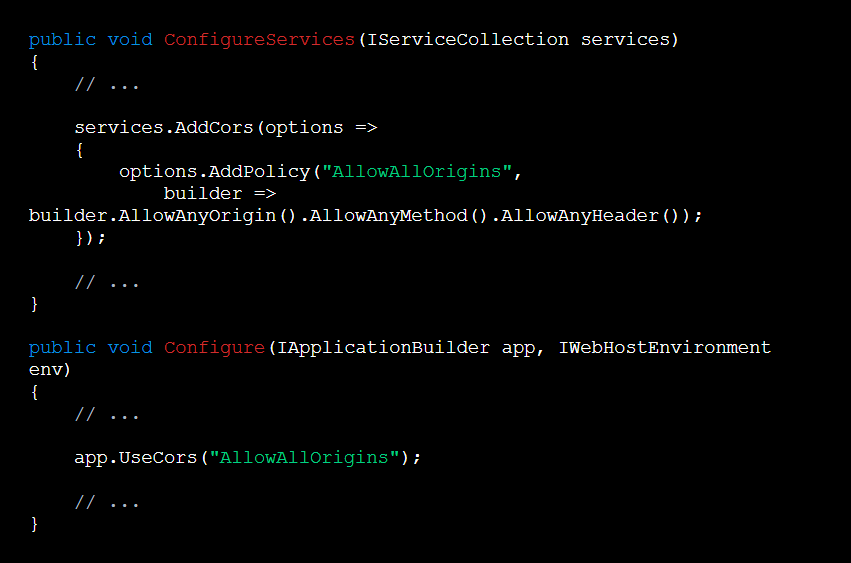
* Microsoft.EntityFrameworkCore.SqlServer
* AutoMapper
* AutoMapper.Extensions.Microsoft.DependencyInjection
* System.Text.Json
* Swashbuckle.AspNetCore

## Exécution de l’application via IIS Express

# Integration d’ASP.NET Core avec Angular

## Activer CORS (Cross-Origin Resource Sharing)

Dans *startup.cs*:



## Configurer les appels vers l’API

Dans l’application Angular, utiliser *HttpClient* pour faire des appels vers le backend.

# Développement du frontend

## Architecture de Angular:

Les applications Angular sont structures autour de 2 types d’éléments:

* Composants (Components): “Blocs de construction” des applis Angular. Chacun est compose d’un template HTML, d’une classe TS et d’un fichier CSS.
* Services: Classes qui encapsulent des fonctionnalités logiques ou des opérations que l’on souhaite partager à travers plusieurs composants de l’application.

## Création de components

### Génération d’un nouveau component

Ng generate component nom-component

### Implémentation du template

Ecrire le code HTML dans le fichier .html (Peut inclure du data binding, de directives, ...)

### Ajout de la logique

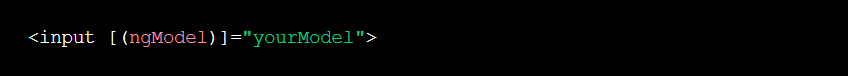
Définir les variables et les fonctions que le component va utiliser dans le fichier .ts.

## Gestion de l’état et Data Binding

### Directive *ngModel*

A utiliser pour du data binding dans les deux sens.

Exemple:



* Lie la valeur de l’input a la propriété *yourModel* dans le component.

### Utilisation de services pour la gestion de l’état

Implémenter des services pour gérer et partager les données entre components.

## Routage et navigation

### Configuration du routage

Dans *app.module.ts*, importer *RouterModule* et définir les routes.

### Router Outlet

Utiliser *<router-outlet></router-outlet>* dans le template (.html) où le routeur devrait afficher les différentes vues.

### Navigation

Utiliser le service *Router* dans le component pour naviguer entre les vues.

## Intégration avec le backend ASP.NET Core

### Création d’un service

Créer un service pour gérer les appels vers l’API en utilisant *HttpClient* (Importer *HttpClientModule* dans *app.module.ts*).

A computer screen with colorful text

Description automatically generated

A computer screen with colorful text

Description automatically generated

### Utiliser le service

Injecter le service dans les components pour utiliser les méthodes de l’API.

# Développement du backend

## Structure d’un projet ASP.NET Core

* Fichier *startup.cs*: Configure les services et le pipeline de requêtes de l’application (Spécifie la manière dont l’application devrait répondre à la requête HTTP).
* Fichier *program.cs*: Héberge le point d’entrée principal de l’application. Il configure l’hôte web, prepare les services et initie l’application.
* Repertoire des Contrôleurs (Controllers): Contient les controlleurs de l’API qui gerent les requêtes HTTP.
* Repertoire des modèles (Models): Là où sont définis les modèles de données.

## Construire un controller

### Créer un controller

Dans le repertoire *Controllers*, ajouter un nouveau controller. Utiliser le template *API controller* fourni par VS.

### Definir les routes

Utiliser des attributs comme *[HttpGet]* et *[HttpPost]* pour définir les routes.

A screen shot of a computer program

Description automatically generated

## Se connecter à une base de données

### Entity Framework Core

A utiliser pour pouvoir utiliser ORM (Object-Relational Mapping). Permet d’interagir avec la DB avec des objets C#.

### Créer une connection string dans *appsettings.json*

### Créer les modèles de données

Créer les classes C# dans le repertoire *Models*, puis assembler un contexte de DB et générer les tables avec Entity Framework.

## Implémenter la logique

### Services et Repositories

Créer les services et les Repositories pour encapsuler la logique et la logique d’accès aux données.

### Injection de dépendances

Utiliser la fonctionnalité d’injection de dépendance d’ASP.NET Core pour injecter les services dans les controllers.

## Gestion de CORS

Dans *startup.cs*:

A screen shot of a computer program

Description automatically generated

## Création d’endpoints API

Définir des méthodes dans les controllers servant d’endpoints.

A screen shot of a computer code

Description automatically generated

## Préparer Angular pour communiquer avec ASP.NET Core

Vérifier si *HttpClient* et bien importé dans *app.module.ts*, puis créer des services pour gérer les appels vers l’API.

A computer screen with colorful text

Description automatically generated

# Effectuer des tests

## Tester avec Angular

Utiliser le service pour faire des requêtes vers le backend et mettre les réponses dans un log.

A screen shot of a computer

Description automatically generated

## Utiliser les outils de développement

La console de développeur du navigateur ou l’onglet *Network* peuvent être très utiles.

## Debugger dans VS

Utiliser des breakpoints dans les controllers pour s’assurer que les requêtes arrivant du frontend sont reçues et traitées correctement.

# Mesures de sécurité

## Sécuriser le frontend

### Prévenir les attaques XSS

A screen shot of a computer

Description automatically generated

### Utiliser Https

A green and white text on a black background

Description automatically generated

## Sécuriser le backend

### Sécuriser les endpoints avec JWT (JSON Web Tokens)

A computer screen with text on it

Description automatically generated