-----\*\*\*\*\* PROJET REALISE AVEC VISUAL STUDIO CODE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*------------------

launchsetting.json --> http seulement

-Lancer **dotnetcore: dotnet watch run** --- watch en option

-creer .net visualstudio code :

en ligne de commande : **dotnet new 'nomprojet'**

package pour entity DbContext ----> voir DattingApp.api.csproj

<PackageReference Include="Microsoft.EntityFrameworkCore" Version="3.1.3"/>

<PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite" Version="3.1.3"/>

---> créer

-les classes pour les tables

-le dataContext pour communiquer entre projet et base via entity

-option dans dataContext la **connectionString**

Dans **appsettings.json** "ConnectionStrings":{

"DefaultConnection":"Data Source=datingapp.db"

Dans Startup ---> injection public void **ConfigureServices**(IServiceCollection services)

{

services.AddDbContext<DataContext>(X=>X.UseSqlite(Configuration.GetConnectionString("DefaultConnection")));

services.AddControllers();

}

entity code first avec **migration**

I) installer Microsoft.EntityFrameworkCore.Design nuget---

en ligne de commande : **dotntet tool install --global** si non installé

si installé ---->

**dotnet ef -h** ---> pour trouver l'option à entrer pour creer la bdd depuis le code

puis dans le rep du projet :--

Création de la base :

-**dotnet ef migrations** add InitialCreate

**Dans une api les controleurs dérivent de la classe ControllerBase -🡪 pas de vue associée au projet**

**Dans une appli mvc les controleurs dérivent de la classe Controller 🡪 vues**

Installation de Angular client:

dans le repertoire du projet

**npm install -g @angular/cli**

dans le repertoire parent du projet

créer le projet angular :

**ng new DattingApp-SPA**

Outils à installer dans visual :

- Angular snippets --> ajoute des extraits de code

- Angular files ---> ajout de fonctionnalités éditeur

- Angular language ---->

* - Rename tag ---> permet de modifier les balises html plus facilement
* - Braket pair colorised2 ---> config des couleurs de visual
* - Debugger for chrome --
* - Material icon theme --> change les icones de fichier dans VSC
* - Pretier code formatter --> améliore l'apparence du code
* - TS lint ??
* - Angular 2 switched --> raccourci clavier pour bascule html css

**Exemple page simple développée sur visual studio code**

Affichage : Données de la table value (id,name) avec **DattingApp-SPA** Angular

Données : Table value créée dans l’api **DattingApp.api** utilisant Entity code first

Ajout d’une page

Angular-->nlle page : bn dr--> dans app--> generate componenent (nom minuscule)

* création d'un rep avec .css .html .ts

Pour fonctionnement en client

**Dans app.modules.ts**

* En tête 🡪 référencer le chemin

import { HttpClientModule } from '@angular/common/Http';

* Ajouter HttpClientModule

Imports [

HttpClientModule

]

Comme c’est un projet page unique lancement auto au démarrage, **dans value.component.ts**, ajouter une requête dans le constructeur :

1. Importer HttpClient et l’ ajouter comme instance du constructeur
2. import {HttpClient} from '@angular/common/http';

export class ValueComponent implements OnInit {

  values : any;

  constructor(private http:HttpClient) { }

  ngOnInit() {

    this.getValues();

  }

  getValues(){

    this.http.get('http://localhost:5000/api/values').subscribe(response=>{

      this.values = response;

    },error => {console.log(error);});

  }

}

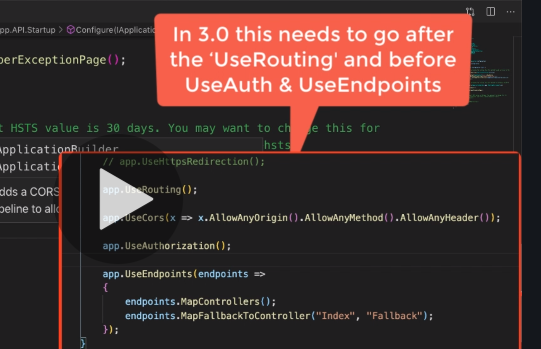
Il faut enfin ajouter un service pour pouvoir communiquer du port 5000 à 4200 malgré les règles de sécurité dans l’app serveur Api: **startup.cs**

Dans **ConfigureServices** :

  services.AddCors();//ajout service pour le passage d'un port à un autre

            // ce service est configuré dans void Configure

2)dans Configure :



1. Afficher dans valueComponent.html

<p \*ngFor="let value of values">

  {{value.id}},{{value.name}}

</p>

1. Mise en page:

**Install bootstrap font-awesome** en ligne de commande

Importer bootstrap dans le fichier CSS

@import '../node\_modules/bootstrap/dist/css/bootstrap.min';

@import '../node\_modules/font-awesome/css/font-awesome.min.css';

copie du projet Angular pour transmettre --> ne pas inclure node\_module ====> les fichiers sont rechargés avec le package.json

Cacher les ifo sensibles

1er solution :

Gitignore – ajouter le appsetting.json qui contient la clé de token

2eme solution :

SecretLocalSrotage

Dans DattingApp.api.csproj :

<UserSecretsId>79a3edd0-2092-40a2-a04d-dcb46d5ca9ed</UserSecretsId>

En ligne de commande :

dotnet user-secrets set "AppSettings:Token" "Super clé hyper secrete"

pour contrôler en ligne de commande :

dotnet user-secrets list

*🡪 AppSettings:Token = Super clé hyper secrete*

Ensuite on peut enlever la ligne dans Appsettings:

"AppSettings":{

    "Token":"Super clé hyper secrete"

  },

PROBLEME : NE FONCTIONNE QU’EN MODE DEVELOPPEMENT

Token authentification :

Token Json 🡪 Standards RFC7519



Informations publiques

Algo codage

Type de codage

Zone cryptée

Cryptage en tête

Crypotage Public

Permet au programme de vérifier l’authentification d’un utilisateur sans interroger le serveur

Token :

En tête

Coprs

Signature : Infos de login à contrôler

* A la première connexion envoi mdp et nom au serveur
* Le serveur hache et contrôle la concordance entre mdp envoyé et contenu de la base
* Le serveur envoi un jeton au client
* Le client sauvegarde le jeton en local
* Ensuite les requêtes se font ensuite en envoyant le jeton

Autentification

Modules à ajouter pour l’autentification :

* Microsoft.IdentityModel.Tokens
* System.IdentityModel.Tokens.Jwt

Autentification par logiciel :

**Module à ajouter :**

* Microsoft.AspNetCore.Authentication.JwtBearer

Dans startup.cs

* Configurer le service
* services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme).AddJwtBearer(options=>{
* options.TokenValidationParameters = new TokenValidationParameters
* {
* ValidateIssuerSigningKey = true,
* IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes(Configuration
* .GetSection("AppSettings:Token").Value)),
* ValidateIssuer = false,
* ValidateAudience = false
* };
* });
* Configurer
* app.UseRouting();
* app.UseAuthentication();
* app.UseAuthorization();

Page accueil avec login dans Angular :

* Dans app 🡪 *new component* ---- répertoire **nav**
* Copie modele jumbotron depuis Bootstrap 🡪<nav class= ‘navbar’…….
* Modifications selon affichage désiré
* Dans le <Form> contenant le login on crée une variable pour binder la form
* Chaque input est associé à un binder [(ngModel)]= ‘’ et le submit [(ngSubmit)]= ‘function()’

Après… les services :

Pour ne pas répéter le même code plusieurs fois, on crée un service.

Pour le log : **auth.service**

@Injectable({

  providedIn: 'root'

})

export class AuthService {

  baseUrl  = 'http://localhost:5000/api/auth/';

// tslint:disable-next-line: typedef-whitespace

constructor(private Http : HttpClient) { }

// tslint:disable-next-line: whitespace

// tslint:disable-next-line: typedef-whitespace

login(model:any){

      // pipe empile chk fonction avec le resultat de la fonction précédente

  return this.Http.post(this.baseUrl + 'login', model).pipe(

      // on passe user venant du serveur en paramètre

    map((response: any) => {

      const user = response;

      if (user){

        // le token est enregistré en local pour les connexions futures

        localStorage.setItem('token', user.token);

      }

    })

  // tslint:disable-next-line: semicolon

  )

}

}

Puis l’**injection** du service :

Dans nav.component.ts

import { Component, OnInit } from '@angular/core';

import { AuthService } from '../\_services/auth.service';

@Component({

  selector: 'app-nav',

  templateUrl: './nav.component.html',

  styleUrls: ['./nav.component.css']

})

export class NavComponent implements OnInit {

  // pour récupérer les valeurs depuis navcomponent.html

  model: any ={};

  constructor(private authService : AuthService) { }

  ngOnInit() {

  }

  login(){

    //il faut souscrire au service avec le this.model

      //subscribe(observer=>)

    this.authService.login(this.model).subscribe(next =>{

      console.log('loggin OK !!!')} , error => {

        console.log('error');

      });

  }

Passage de paramètres du parent vers l’enfant

Dans **home.compnent**, on récupère les valeurs depuis l’api **DattingApp**

**register.component.html (app-register)** est l’enfant appelé dans **home.component.html**

Dans **homeController.html** on binde la variable de **register.component.ts**

<app-register [valuesFromHome] = "values"></app-register>

Dans l’enfant (**register.component.ts**), créer la variable en Input

import { Component, OnInit,Input } from '@angular/core';

export class RegisterComponent implements OnInit {

  @Input() valuesFromHome;

  model: any ={};

  constructor() { }

les valeurs sont ensuite utilisables dans le **register.component.html** du fait du **@Input**

exemple de liste récupérée depuis l’api via home.component:

 <div class="for-group">

    <label for="favorite-value">Votre valeur préférée</label>

    <select name="" id="favorite-value" class="form-control">

        <!--\*\*\*\*\*\*\*\*[value]-> pour option select-option-value -->

        <!--\*\*\*\*\*\*\* "value"-> valeur de la boucle \*ngFor -->

        <!--\*\*\*\*\*\*\*\* values -> passé à app-regiser avec :

              \*\*<app-register [valuesFromHome] = "values"></app-register> -->

      <option \*ngFor="let value of valuesFromHome" [value]="value">{{value.name}}</option>

    </select>

  </div>

Passage de paramètres de l’enfant vers le parent

Ici on envoie un **event** en passant par un @**Output** dans **l’enfant**

On veut renvoyer un cancel venant de **register.component.html** vers le parent

  @Output() cancerRegister = new EventEmitter();

Qui a été importé

import { Component, OnInit,Input, Output, EventEmitter } from '@angular/core';

dans la fonction cancel on déclenche l’event

  cancel(){

    //envoi de l'event avec la valeur false pour valider le cancel

    this.cancelRegister.emit(false);

    console.log('cancelled');

  }

Enfin dans le [**home.component.html**](http://home.component.html)on lie le cancel à l’enfant **register.component**

        <app-register [valuesFromHome] = "values" (cancelRegister)="cancelRegisterMode($event)"></app-register>

Il faut finalement créer la méthode dans le **home.component.ts** pour passer la valeur de l’enfant vers le parent en l’occurrence

Récapitulatif :

* Parent🡪 enfant : Déclarer @Input dans le fichier **enfant .ts**

Utiliser la variable importée du parent dans le **.Html** : *{{value.name}}*

* Enfant🡪 parent : Déclarer @Output entant qu’**EventEmitter** dans le fichier **enfant .ts**
  + - ***Exemple pour un Cancel envoyé depuis l’enfant***
* @Output() cancelRegister = new EventEmitter();

Dans la fonction qui emet l’**Event**

this.cancelRegister.emit(false);

Dans le **parent** **.Html** appel de l’enfant et récupération de l’**Event**:

<app-register [valuesFromHome] = "values" (cancelRegister)="cancelRegisterMode($event)"></app-register>

Dans le **parent .ts** déclaration de la fonction pour récupérer l’**Event**

  cancelRegisterMode(registerMode: boolean){

    this.registerMode = registerMode;

  }