Pokretanje same aplikacije

U korenu foldera nalaze se sledeci folderi: gameMicroservice, scoreMicroservice, userMicroservice, View

Pokrenuti SQL Server

U svakom od mikroservisa zameniti string za konekciju - `DatabaseConnection`:

- \gameMicroservice\Game**appsettings.json**
- \scoreMicroservice\Score**appsettings.json**
- \userMicroservice\User**appsettings.json**

Uci u svaki folder pojedinacno putem terminala u koren foldera Game, Score i User i izvrsiti komandu **dotnet ef database update** ona ce pokupiti sta se nalazi u Migrations folderu i napraviti strukturu baze. Ukoliko Migration folder ne postoji ili je prazan ili pri promenom modela baze izvrisiti komandu pre update : **dotnet ef migrations add initialMigration**

Mikroservisi se pokrecu svaki od njih nezavisno od drugog. Otvoriti Command Promt i zatim preko terminala ici do foldera koji sadrzi *Program.cs* fajl koji je i stratni fajl svakog mikroservisa.

- \gameMicroservice\Game
- \scoreMicroservice\Score
- \userMicroservice\User

Tako da otvoriti 3 terminala i preko terminala uci u svaki folder od gore navedenih. Zatim u sva 3 terminala kada se nalazimo u gore napomenutim direktorijuma radimo **dotnet run** koja ce pokrenuti svaki mikroservis.

Sada nakon njihovog pokretanja ne bi bilo lose da se proba da se kontaktira svaki preko obicnog browsera koji ce biti koriscen za testiranje kako bi odobrili “nesigurnu” konekciju. U stvari ovde je problem sto su mikroservisi postavljeni da budu **https** a kako sertifikat nije postavljen od strane reputabilne kompanije browser nas pita da to potvrdimo, svakako testiranje je lokalno tako da je prihvatljivo.

- https://localhost:5101/ - User mikroservis
- https://localhost:5201/ - Score mikroservis
- https://localhost:5301/ - Game mikroservis

Pokretanje frontend aplikacije - Angular
Otvoriti jos jedan terminal i uci u folder: **\View\Angular\game-app** zatim uraditi **ng run**
Pored ng run komande isto je moguce uraditi i sam build i nakon toga moze aplikacija sama preko generisanih html, css i js fajlova da se pokrene bez potrebe za pokretanjem angular “servera”.

Opis Strukture Mikroservisa

Svaki mikroservis je veoma slicno radjen sa izmenama pojedinih parametra koji su nam potrebni

Program.cs - startni fajl, generisan takav bez izmena
Startup.cs - sadrzi dodatno setovanje nekih specificnih stavri za nasu aplikaciju, izgenerisan. Pored izgenerisanog dela dodate su po neke stvari.

U ConfigureServices klasi :

- **services.AddScoped** -> Scoped objects are the same within a request, but different across different requests.
- **services.AddDbContext** -> prosledjujemo nas string za bazu
- **services.AddCors** -> omogucavaju nam da se konektujemo preko frontenda na nas API. ovde su dozvoljeni svi da imaju pristup, ali zbog bezbednosti bi trebalo odobriti samo odredjeni domen u produkcijskoj veriziji, vise info : https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
- **services.AddAutoMapper** -> info https://www.pragimtech.com/blog/blazor/using-automapper-in-asp.net-core/

U `Configure` klasi:
Samo dodajemo nas `app.UseCors("MyPolicy");`

launchSettings.json

U ovom fajlu postoji `applicationUrl` polje gde postavljamo na kom portu hocemo da se pokrece nasa app.

Model/

U ovom folderu se nalaze modeli nase baze, svaki model predstavlja 1 tabelu u SQLu

Data/

Sadrzi nacin povezivanja na bazu i slanja i dobijanja info od SQL-a
I_NAZIV_Repo.cs - interfejs
_NAZIV_Context.cs - kontekst da se povezemo na bazu, specificiramo na koju tabelu, odnosno na koji nas model da se povezemo
Sql_NAZIV_Repo.cs - implementira nas interfejs i vrsi obradu sa bazom

Controllers/

Sadrzi kontrolere, ovde definisemo nase APIje kao i sta su input podaci za apije, i sta i koje kodove i kada vracaju, kao i definicija putanja

```
[Route("api/game")]
```

Definise putannju ovde se odnosi na localhost:5301/api/game

```
[HttpGet]
```

Kaze da ispod se nalazi funkcija koja prima ovaj GET request i treba da ga odradi…

```
IMapper mapper
```

Sluzi nam za mapiranje DTO i Model objekata.

```
[HttpDelete("{id}")]
```

Kaze da je putanja **localhost:5301/api/game/ID_BROJ_PARAMETAR**
NPR : **localhost:5301/api/game/1** -> brise elemenat iz baze sa IDjem 0

```
[HttpGet("all")]
```

localhost:5301/api/game/all

```
 [HttpPut]
```

Nema nista specijalno da se poveze na _repository

```
_mapper.Map(gameUpdateDto, gameModelFromRepo);
```

Mapper mu kaze da uveze DTO koji dobije od korisnika sa Modelom iz nase baze. Ovim on preslika novodobijene vrednosti iz DTO-a u nas model. Nakon ovoga je potrebno da upamtimo promenu kako bi bila zabelezena.

```
_repository.SaveChanges();
```

Vraca `return NoContent();`
Sto govori da je 204 stausni kod. 200 kodovi govore klijentu (android app-u) da je sve proslo kako treba, noContent posto ne vraca nikakvu vrednost.

Dtos/

https://docs.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5

Klase koje nam sluze da specificiramo strukturu podataka koju treba da dobijemo od klijenta (angulara) i da posaljemo nazad

Profiles/

Specificiramo nas AutoMapper

Kako koristimo nase DTO objekte da definisemo sta dobijamo i saljemo klijentskoj aplikaciji (angular u nasem slucaju) a kako su klase modela prikazi tabele i polja u nasoj bazi ovde kazemo c# kako da mapira stvari. U sustini kazemo da odredjeni DTO predstavlja odredjeni model u bazi. DTO objekti ne moraju da imaju sva polja u bazi.

Opis Strukture Angulara

Angular je aplikacija od jedne strane, putem svog router komponente stvara iluziju o vise stranica. U principu samo menja GUI ali se sve izvrsava u jednoj strani, zato ne postoji “refresh” i povecava UX.

Bitni fajlovi :
/package.json definise skripte i neke dependencije koji se koriste u razvoju aplikacije

/src/index.html
-> startni fajl aplikacije,

```
<link rel="preconnect" href="https://fonts.gstatic.com">
  <link
href="https://fonts.googleapis.com/css2?family=Barlow+Semi+Condensed:wght@
100&display=swap" rel="stylesheet">
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstra
p.min.css" rel="stylesheet"
integrity="sha384-giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTfAAs
rekwKmP1" crossorigin="anonymous">
```

```
  <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/umd/popper.min
.js"
integrity="sha384-q2kxQ16AaE6UbzuKqyBE9/u/KzioAlnx2maXQHiDX9d4/zp8Ok3f+M7D
Pm+Ib6IU" crossorigin="anonymous"></script>
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/js/bootstrap.
min.js"
integrity="sha384-pQQkAEnwaBkjpqZ8RU1fF1AKtTcHJwFl3pblpTlHXybJjHpMYo79HY3h
Ii4NKxyj" crossorigin="anonymous"></script>
  <script src="https://code.jquery.com/jquery-3.5.1.min.js"
integrity="sha256-9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0="
crossorigin="anonymous"></script>
```

ovde sam uvezao dodatno link i script tagove koji nam dovlace potrebne “biblioteke” za css i js

Ispod `<app-root></app-root>` dodao modal i script

Ovde app-root tag je tag koji prikazuje stvari iz nase app komponente koja je ujedno i glavna komponenta

-> i na samom kraju pod `<script>` su ispisane funkcije koje pozivamo u nase komponente...

Svaka komponenta ima fajlove za HTML, CSS i TS (gde je logika komponente), svaka komponenta se kodira kao takva i povezuje se sa drugim komponentama i cini aplikaciju

/src/app - app-root komponenta - defaultno nema puno stvari

app.module.ts - ovde imamo imports, providers, declarations i bootstrap : definisemo koji fajl, koja komponenta i koji servis ima koju ulogu u nasoj app.

app-routing.module.ts - ovde definisemo koji URL parametar ce pozvati koju komponentu da prikaze kao root komponentu za tu odredjenu stranicu

/src/app/services

Sadrzi service, odnosno preko njih kontaktiramo nase C# APIje, definisemo sta saljemo mikroservisima, i prosledjujemo nazad dobijene podatke kao repsonse. Ovde neki imaju i neke varijable, te varijable su dostupne uvek kao takve i cuvaju to stanje tokom zivotnog ciklusa naseg app-a (dok korisnik ne zatvori stranu) ili dok ih mi ne promenimo…

Podela je isto kao za mikroservice i za tabele u bazi. Imamo

- game.service.ts
- score.service.ts
- user.service.ts

/src/app/services/components

Foldre sa ostalim komponentama nase aplikacije, naziv aplikacija predstavlja naziv glavne “strane” na sajtu. Ukoliko komponenta ima podkomponente one se nalaze ugnezdene u podfolderima u toj komponenti. Ovo sad je nacin kako sam dizajnirao app, ne mora da bude ovakva struktura foldera.

Kod angulara setanje kroz druge strane obradjuje takozvani RUTER i na osnovu sta postavimo u **app-routing.module.ts** on poziva odredjene komponente za prikaz. Zato se i koristi

```
<a routerLink="/">
```

```
<a routerLink="/game-table-view">
<a routerLink="/game">
<a routerLink="/login" >
```

umesto href atributa za promenu stranica.

Podatak o tome da li je user logovan ili ne se cuva u `activeUser` u user.service.ts komponenti

Login Komponenta

U TS konstruktoru uvozimo UserService, Router, GameService klase tako da mozemo da uticemo na njihova polja.
U konstruktoru proveravamo ako je user logovan da ga prebacimo na GAME stranu...

U HTML fajlu ima formu za logovanje i 2 dugmeta.

```
[(ngModel)]="this.usr"
```

Takozvano “binding” sto povezuje varijable iz TS fajla komponente da budu povezane sa vrednoscu ovog INPUT polja
https://www.w3schools.com/angular/angular_databinding.asp
Tako da u TS fajlu imamo varijable kao atribute klase

```
usr: string = '';
pwd: string = '';
```

Tako da na svaku promenu input polja i varijable se menjaju i dobijaju i gube vrednosti..

```
<button (click)="onLogin()" id="loginBtn" class="btn
btn-dark">Login</button>
```

Ovde se definise CLICK event koji poziva funkciju onLogin iz naseg TS fajla.

U TS fajlu na pocetku

```
declare const sendMessageUserModal: any;
```

Definise ovu funkciju, ta funkcija nam se nalazi u View/Angular/game-app/src/index.html fajlu dole u <script> tagu. Tako definisanu funckiju mozemo da koristimo onda TS fajlu. Ova funkcija

nam sluzi ovde da napravi MODAL odnosno DIJALOG koji treba da prikaze nekakav tekst korisniku.

```
 onLogin(): void {
    this.userService.loginUser(this.usr, this.pwd).subscribe((data)=>{
      this.onApiLoginOrRegisterActionResult(data)
    }, (err)=>{ this.onApiLoginOrRegisterActionResult("lozinka ili
korisnicko ime netacno!") });
  }
```

Kaze da koristi funkciju loginUser iz naseg UserService klase koja kontaktira c# API i vraca Observable. Tako da moramo da idemo **subscribe** (cekamo od servera da vrati rezultat). Ova funkcija SUBSCRIBE prima 2 argumenta

prvi - da je sve proslo ok (kod 200+)
drugi - da je doslo do neke greske ili drugog statusnog koda

Game komponenta

```
*ngIf="this.gameService.gameActive"
```

Proverava da li je varijabla koja se nalazi u GameService-u pod nazivom gameActive : true ili je false. Ako je true bice prikazano sve sto se nalazi ugnjezdeno pod elementom koji ima ovaj ngIf ako je FALSE nece biti prikazano nista.

```
*ngFor="let item of this.serverData.dataOne"
```

Radi FOR petlju, ide kroz nasu listu koja se zove dataOne, koja se nalazi u this.serverData objektu. Svaki elemenat mozemo da referisemo pod nazivom **item**.

```
{{item.columnOne}}
```

Ispisuje vrednost prve kolone u HTML-u i tako mi dobijamo koja je to vrednost.

```
[ngClass]="{
     'list-group-item-secondary' : item.selectedOne,
     'list-group-item-success' : item.success,
     pointer: item.pointer,
     'list-group-item' : true,
     'list-group-item-action' : true
 }"
```

U zavisnosti od varijabli sa desne strane, klase pod nazivom sa leve strane ce biti aktivne na ovaj HTML elemenat.
Npr. class=”list-grooup-item” ima vrednost uvek TRUE i uvek je aktivna.]

```
(click)="onColumnOneClick(item.columnOne, item.pointer)"
```

Poziva funkciju i prosledjuje joj vrednosti odredjenomg elementa iz nase liste koji je predstavljen kao item.

Game komponenta

Uvlaci u sebe 2 komponente i ima HTML delove svoje...

User Action Btns Component i Score Tab Component su prikazani u HTMLu na sledeci nacin

```
<app-score-tab></app-score-tab>
<app-user-actions-btns></app-user-actions-btns>
```

Nazive ovoga znamo tako da kada napravimo komponentu ona po default-u na pocetku TS fajla ima

```
@Component({
  selector: 'app-game',
```

Gde je selector naziv taga koji mozemo bilo gde u nasoj aplikaciji da postavimo i angular ce da ga prepozna i da vrsi prikaz te komponente. Prikaz komponente podrazumeva HTML + CSS + TS fajlove te komponente.

Ovo je uopsteno kako sta radi u angularu i kako je sta uvezano

Terminal komanda za generisanje angular kompoenti:
ng generate component --skipTests=true NAZIV_KOMPONENTE
Ovo generise html+css+ts fajlove za komponentu i uvezuje je u app.module.ts
--skipTests=true sluzi da izbegnemo da generise testne fajlove…

Mozda da se pohvata angular nije lose pogledati neki video tutorijal poput https://www.youtube.com/watch?v=Fdf5aTYRW0E da pohvatas uopsteno o angularu kako funckionise...