

# **Faza I - Arhitekturni projekat AmongUs - OurWay**

**Osnovne funkcionalnosti:**

* **Prijavljivanje korisnika** - Korisnici ce imati mogucnosti prijavljivanja. Korisnicki nalozi ce sadrzati istoriju odigranih partija kao i mogucnost pregleda istorije jedne partije.
* **Vecinu funkcionalnosti igre AmongUs:**
* **Formiranje predvorja igre** - Korisnici mogu da formiraju predvorje igre koje ima jedinstven kod za prikljucivanje.
* **Pocetak igre** - Nakon sakupljanja igraca u predvorju igra moze otpoceti. Pocetak inicira kreator predvorja.(Igra ne moze poceti ako je u predvorju manje od 4 igraca)
  + - **Uloge igraca** - Postoje 2 vrste uloga:  
       - **Crewmate(clan posade)** - Cilj ove uloge je da zavrsi sve  
       zadatke pre nego sto uljez izvrsi dovoljno ubistava ili da  
       Izbaci uljeza  
       - **Impostor(Uljez)** - Cilj ove uloge je ubijanje clanova   
       Posade i prikrivanje svojih namera.
    - **Akcije igraca tokom igre:** - **Zapocni zadatak** - Clanovi posade mogu da zapocnu   
       zadatak kada se nalaze u njegovoj blizini.  
       - **Ubi** - Uljezi mogu da ubiju clana posade kada se nalaze  
       U njegovoj blizini. Nakon sto su ubijeni, igraci postaju  
       duhovi i mogu da nastave da zavrsavaju zadatke.  
       - **Koristi tunel** - Uljezi mogu na posebnim mestim na  
       koriste pomocne tunele za brze kretanje po mapi.  
       - **Pritisni dugme** - Na mapi postoji dugme cijim pritiskom   
       se saziva sastanak. Tokom sastanka otvaraju se svi   
       komunikacioni kanali i igraci ulaze u debatu kako bi  
       otkrili uljeza.   
       - **Prijavi telo** - U slucaju da igrac pronadje “ubijenog”   
       igraca, telo se moze prijaviti nakon cega se odmah   
       saziva sastanak.  
       - **Glasanje** - Tokom sastanka igraci mogu da glasaju za  
       druge igrace. Igrac sa najvecim brojem glasa se  
       “Izbacuje” iz igre. Izbaceni igraci postaju duhovi.
    - **Kraj igre** - Uslovi pobede clanova posade:  
       - Izbaceni su uljezi.  
       (ili)  
       - Zavrseni zadaci.  
        
        
       - Uslovi pobede uljeza:

- Preostali broj zivih clanova posade je jednak broju   
 preostalih uljeza.

#### 

#### 

#### 

#### 

#### **Angular**

Angular koristi komponente koje predstavljaju templejtske direktive. Sadrzi 2 vrste direktiva:

1) Strukturne direktive: menjaju raspored DOM-a izmenom njegovih elemenata.

2) Atributne direktive: menjaju kako se DOM ponasa i izgled elemenata.

Komponente definisu preglede (**views**), koju su skupovi elemenata na ekranu koje Angular moze da bira i modifikuje prema logici i podacima naseg programa. Komponente koriste **servise (services),** koji pruzaju specificne funkcionalnosti koje nisu direktno povezane sa pregledom/izgledom. Servisi se mogu ubaciti (eng. noun **Injection**) u komponente kao zavisnosti ( eng. **Dependencies**), sto doprinosi modularnosti, povecava upotrebljivost i efikasnost koda.

U angular **aplikacijama**, **View** je predstavljem .**html** templejtima u kombinaciji sa .**css** delovima komponenti. Ako pratimo MVC sablon, View bi definisao mesta u koja mozemo da ubacimo nase podatke. Takodje on pruza metode za registrovanje **callback** kad god je dugme kliknuto. U ovom smislu, View ne bi znao za postojanje **Controller**-a. Kontroler za za postojanje View-a i Modela i povezuje ih. Angular ima drugaciji pristup, nazvan **MVVM pattern (Model-View-ViewModel)**.

**ViewModel** se nalazi u **.ts** fajlu komponenata.

U @Component dekotatoru klase, mozete videti referencu na **View .html** i **.css** fajlove. U ostatku klase, s druge strane, nema reference na View. Umesto nje, ViewModel koji se nalazi u klasi Componente otvara atribute i metode kojima View moze da pristupi. U prevodu, u odnosu na MVC arhitekturu, zavisnost je obrnuta.

**ViewModel** ne zna da View postoji, ali pruza API slican Modelu koji moze da se koristi od strane View-a. Posledica toga je da View mora da zna za postojanje ViewModela i njegovih metoda. Razlog za obrtanje zavisnosti je smanjenje ponavljanja koda. Podaci su dvosmerno spregnuti izmedju View-a i ViewModel-a. Podaci u View-u su uvek sinhronizovani sa podacima u ViewModelu.

* Nesto krace o NgRX biblioteci i zasto koristimo ngrx store i ngrx entity za ocuvanje state-a (redux principi dobre arhitekture aplikacija).

***LUKA KAD PREGLEDAS OVO NAPISANO NA SRPSKI JAVI MI NA MESSENGER AKO TRB IZMENE, ISPOD IMAS ORIGINAL NA ENG.***

#### **Angular**

Angular uses components that are directives with templates. There are two kinds of directives in Angular.

**Structural directives**: they alter the DOM’s layout by replacing its elements.

**Attributive directives**: it changes how the DOM behave and the appearance of the element

#### 

#### **Components define *views*, which are sets of screen elements that Angular can choose among and modify according to your program logic and data.**

#### **Components use *services*, which provide specific functionality not directly related to views. Service providers can be *injected* into components as *dependencies*, making your code modular, reusable, and efficient.**

In Angular applications, the View lives inside the .html templates and the .css style sheets.

If you were to follow the MVC pattern, the View would define slots into which the data could be inserted. It would also provide methods for registering a callback whenever a button is clicked. In this respect, the View would remain completely ignorant of the Controller. The Controller would actively fill the values and register callback methods with the View. Only the Controller would know about both the View and the Model and link the two together.

Angular takes a different approach, called the MVVM pattern.

The ViewModel lives in the .ts files of the components.

In the @Component decorator of the class, you can see the reference to the View .html and .css files. In the rest of the class, on the other hand, there is no reference to the View whatsoever. Instead, the ViewModel, contained in the Component class, exposes properties and methods that can be accessed by the View. This means that, compared to the MVC architecture, the dependency is reversed.

The ViewModel has no knowledge of the View but provides a Model-like API that can be used by the View.This means that the View needs to know about the ViewModel and its methods. The reason for reversing the dependency is the reduction of boilerplate code. There is a two-way data binding between the View and the ViewModel. The data in the View is always in sync with the data in the ViewModel.

Html - view

.ts - controller

Services - model



