Web aplikacija za estimaciju i planiranje temeljeno na tehnici poker planiranja

Pleš, Manuel

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:963112

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-21**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

WEB APLIKACIJA ZA ESTIMACIJU I PLANIRANJE TEMELJENO NA TEHNICI POKER PLANIRANJA

Diplomski rad

Manuel Pleš

Osijek, 2022.

SADRŽAJ

1. UVOD 1

1.1. Zadatak diplomskog rada 1

2. TEHNIKA POKER PLANIRANJA I PREGLED POSTOJEĆIH RJEŠENJA 2

2.1. Tehnika poker planiranja 2

2.2. Pregled postojećih rješenja 3

2.2.1. Scrum Poker Online 3

2.2.2. Planning Poker 3

2.2.3. We agile you – Poker Planning 3

3. KORIŠTENE TEHNOLOGIJE 4

3.1. JavaScript 4

3.2. TypeScript 4

3.3. HTML 4

3.4. CSS 5

3.5. Angular 5

3.6. Node.js 7

3.7. Express 7

3.8. Socket.IO 7

3.9. RxJS 8

3.10. PrimeNG 8

3.11. ngx-charts 8

4. PROCES RAZVOJA APLIKACIJE 9

4.1. Poslužitelj 9

4.1.1. Događaj *room:create* 11

4.1.2. Događaj *room:join* 13

4.1.3. Događaj *room:vote* 15

4.1.4. Događaj *room:checkIfExists* 16

4.1.5. Događaj *room:showResults* 17

4.1.6. Događaj *room:newGame* 17

4.1.7. Događaj *room:setOptions* 18

4.1.8. Događaj *user:changeName* 19

4.1.9. Događaj *room:setTasks* 20
4.1.10. Događaj *room:setCurrentTask* 21
4.1.11. Događaj *room:assignStoryPoints* 22
4.1.12. Događaj *disconnect* 23
4.2. Klijentska aplikacija 25
4.2.1. Struktura aplikacije 25
4.2.2. Servis *GameService* 28
4.2.3. Komponenta *HomeComponent* 30
4.2.4. Komponenta *RoomComponent* 32
5. IZGLED I NAČIN RADA APLIKACIJE 38
5.1. Početna stranica 38
5.2. Virtualna soba 39
6. ZAKLJUČAK 47
LITERATURA 48
SAŽETAK 49
ABSTRACT 50

1. UVOD

U današnje vrijeme većina poslova se svodi na rad u timu, a nije drugačije ni u IT tvrtkama. Kada su u pitanju veliki projekti koji zahtijevaju širok opseg znanja i poslova, voditeljima tima postaje teško procijeniti bitne faktore kao što su očekivano vrijeme trajanja projekta i složenost određenih zadataka. Time dolazi do potrebe za raznim tehnikama procjene koje uzimaju u obzir mišljenja ostalih članova tima i na taj način doprinose bržim i točnijim odlukama. Jedna od tih tehnika je tehnika poker planiranja i u ovom diplomskom radu opisan je postupak izrade web aplikacije za istu. Web aplikacija rješava prethodno spomenuti problem procijene bitnih faktora nekog projekta. Također uklanja potrebu za fizičkim kontaktom članova tima prilikom postupka procjene na način da omogućava stvaranje virtualne sobe i pruža potrebne alate za provođenje glasanja spomenutom tehnikom.

U prvom poglavlju navedene su i kratko objašnjene postojeće web aplikacije koje rješavaju ovaj problem. Drugo poglavlje sadrži opis tehnologija korištenih za izradu cjelokupnog rješenja. Treće poglavlje opisuje proces razvoja aplikacije i podijeljeno je na dva djela. Prvi dio opisuje poslužiteljsku aplikaciju i način na koji komuniciraju poslužitelj i klijentska aplikacija. Drugi dio opisuje način rada klijentske aplikacije zajedno s njenom strukturom. U četvrtom odnosno zadnjem poglavlju je prikazan izgled web aplikacije. Svaki element je prikazan i objašnjen pomoću slike.

1.1. Zadatak diplomskog rada

Potrebno je napraviti web aplikaciju koja će omogućiti primjenu agilne metode estimacije i planiranja koja se temelji na poker planiranju. Potrebno je implementirati unos zadatka i mogućnost glasanja u stvarnom vremenu kao i prikaz rezultata. Isto tako, potrebno je omogućiti pristup glasanju drugim korisnicima. Kao tehnologija koristi će se Angular programsko okruženje.

2. TEHNIKA POKER PLANIRANJA I PREGLED POSTOJEĆIH RJEŠENJA

U ovom poglavlju objašnjeno je što je to tehnika poker planiranja te su navedena i opisana neka od postojećih rješenja za estimaciju i planiranje temeljeno na istoj.

2.1. Tehnika poker planiranja

Poker planiranje, također poznato kao *scrum* poker je tehnika u obliku igre koju razvojni timovi koriste kako bi procijenili potreban trud za rješavanje određenih zadataka. Ova tehnika funkcionira na način da voditelj tima pročita i pobliže objasni korisničku priču svome timu. Korisnička priča je neformalno objašnjenje nekakve nove značajke koju je potrebno implementirati. Svakom članu tima je prethodno dodjeljen špil karata u kojem svaka karta na sebi ima prikazanu jednu vrijednost. Vrijednost na karti može označavati bodove priče, dane ili neku drugu jedinicu procjene. Nakon što je vođa tima pročitao opis korisničke priče, članovi tima mogu postaviti pitanja i prokomentirati zadatak kako bi riješili sve nejasnoće. Nakon toga svaki član tima odabire jednu kartu iz svog špila. Kada je svaki član tima odabrao po jednu kartu, svi okreću karte i pokazuju ih u isto vrijeme. Veća vrijednost na kartici označava da je zadatak za tog člana tima složeniji i da mu treba više vremena. Treba držati na pameti da svaki član odabire karticu koja predstavlja idealnu vrijednost iz njegove perspektive. Nisu svi članovi tima jednako iskusni i ne bi svima isto trebalo da implementiraju značajku spomenutu u korisničkoj priči. Ukoliko se vrijednosti na okrenutim karticama poklapaju korisničkoj priči se dodjeljuje izglasana vrijednost. Ukoliko su vrijednosti na kartama dosta različite pokreće se rasprava u kojoj svako iznosi svoje argumente i zatim se ponovno glasa.

Ovu tehniku planiranja osmislio je James Grenning 2002. godine jer je smatrao da je metoda *Wideband Delphi* koja se tada koristila uzimala previše vremena i imala dosta ograničenja. Metoda se proslavila nakon što ju je Mike Cohn predstavio u svojoj knjizi Agilna estimacija i planiranje (engl. *Agile Estimating and Planning*) [1].

Prednosti ove tehnike su te što se mišljenje svakog člana tima uzima u obzir i svaki član tima nakon nekog vremena skupi dovoljno iskustva te može na osnovu prethodno izglasanih zadataka dati ispravnu procjenu.

2.2. Pregled postojećih rješenja

2.2.1. Scrum Poker Online

Web aplikacija koja omogućuje brzo stvaranje virtualne sobe u koju je zatim moguće pozvati ostale korisnike putem URL-a ili QR koda [2]. U istoj je zatim moguće glasati i prikazati rezultate glasanja. Ova aplikacija savršeno prikazuje osnovni princip rada svih postojećih rješenja zasnovanih na tehnici poker planiranja. Aplikacija opisana u ovom diplomskom radu sadrži sve funkcionalnosti ove web aplikacije uz neke dodatne kao što je unos predmeta glasanja.

2.2.2. Planning Poker

Web aplikacija koja ima sve mogućnosti kao i prethodno navedena aplikacija, ali uz njih još pruža mogućnost unosa predmeta glasanja, uvoz istih i integraciju s alatom Jira [3]. Rezultati glasanja se spremaju i moguće ih je u bilo kojem trenutku pregledati, isprintati i preuzeti. Sobi mogu pristupiti i gledatelji odnosno korisnici koji ne glasaju. Na ovaj način članovi tima za koje je bitno da imaju uvid u situaciju mogu prisustvovati glasanju bez da na njega utječu. U usporedbi s aplikacijom opisanom u ovom diplomskom radu ova web aplikacija sadrži dodatne funkcionalnosti kao što je integracija s vanjskim alatima, ispis i izvoz rezultata te prisustvovanje glasanju bez utjecaja na isto.

2.2.3. We agile you – Poker Planning

Jednostavna web aplikacija koja sadrži sve osnovne funkcionalnosti vezane za tehniku poker planiranja [4]. Sadrži mjerač vremena koji obavještava korisnike koliko još vremena imaju za glasati te ima mogućnost uvoza predmeta glasanja. Rezultati glasanja se spremaju te se u bilo kojem trenutku mogu promijeniti ili preuzeti. Također je ponuđene opcije za glasanje moguće urediti po vlastitom izboru kako bi se aplikacija mogla koristiti u različite svrhe. Od svih navedenih postojećih rješenja, ova aplikacija je po funkcionalnosti najsličnija aplikaciji opisanoj u ovom diplomskom radu.

3. KORIŠTENE TEHNOLOGIJE

U ovom poglavlju su navedene i ukratko opisane tehnologije korištene pri izradi cjelokupnog programskog rješenja.

3.1. JavaScript

Programski jezik koji podržava objektno orijentirani, deklarativni i imperativni stil programiranja. Omogućava stvaranje objekta bez da se prvo definira njegova klasa. Pokreće se na strani klijenta i time omogućava definiranje ponašanja web stranice kod pojave nekakvog događaja. Uglavnom je poznat po tome što osigurava interakciju s web stranicama, ali koristi se i u mnogim okruženjima koja ne uključuju web preglednike kao što su Node.js, Apache CouchDB i Adobe Acrobat [5]. Prvu verziju JavaScript-a napravio je Brendan Eich i od tada je ažurirana tako da zadovoljava ECMAScript specifikacije opisane u ECMA-262 standardu. ECMAScript standard je osmišljen kako bi osigurao funkcioniranje web stranica na različitim web preglednicima [6].

Iako se u ovom diplomskom radu ne koristi JavaScript, opisan je jer se korišteni TypeScript nadograđuje na njega.

3.2. TypeScript

Programski jezik otvorenog koda razvijen od strane Microsofta 2012. godine. Bazira se na programskom jeziku JavaScript i poseban je po tome što u njega uvodi mogućnost korištenja tipova podataka koji opisuju varijable i objekte. Na taj način uvodi podršku za čvršću integraciju sa korištenim uređivačem koda i omogućuje lakše pronalaženje i ispravljanje grešaka te razvojnim programerima u znatnoj mjeri olakšava snalaženje prilikom čitanja koda. TypeScript kod se prilikom prevođenja pretvara nazad u JavaScript kod što znači da se TypeScript može pokrenuti gdje god se može pokrenuti JavaScript [7]. U ovom diplomskom radu je korištena TypeScript verzija 4.6.3.

3.3. HTML

HTML (engl. *HyperText Markup Language*) je temelj svake web stranice koji definira značenje i strukturu sadržaja. Pojam hipertekst se odnosi na veze kojima se web stranice povezuju međusobno. HTML koristi „označavanje“ kako bi istaknuo sadržaj poput slika i teksta u web pregledniku. Za to se koriste posebni elementi kao što su: <img>, , <p>, , <ul>,

i mnogi drugi. Kao što je vidljivo u navedenim primjerima ti elementi se od ostatka teksta razlikuju po tome što se nalaze unutar posebnih zagrada i nisu osjetljivi na velika i mala slova, ali prema konvenciji se uvijek pišu malim slovima [8].

3.4. CSS

CSS (engl. *Cascading Style Sheets*) je jezik za stilizaciju kojim se određuje kako će izgledati elementi dokumenta napisanog u HTML-u ili u XML-u. On određuje kako će se renderirati elementi na zaslonu ili nekom drugom mediju. Jedan je od temeljnih jezika i standardiziran je diljem web preglednika prema W3C specifikacijama [9]. W3C(eng. *World Wide Web Consortium)* je međunarodna zajednica u kojoj organizacije članice, zaposlenici i javnost rade zajedno kako bi razvili web standarde [10].

3.5. Angular

Razvojna platforma izrađena na programskom jeziku TypeScript koja uključuje razvojni okvir temeljen na komponentama pomoću kojeg je moguće izraditi web aplikacije. Sadrži skup biblioteka koje pokrivaju veliki broj značajki kao što je kontrola formi, komunikacija klijent-poslužitelj i još mnogo toga [11]. Napravljen je 2015. godine kao redizajn AngularJS-a od strane Google-a i od tada je aktivno održavan. Arhitektura pravilno napravljene Angular aplikacije temeljena je na komponentama koje olakšavaju ponovno korištenje, čitanje i održavanje koda.

Komponente su temeljni elementi za izgradnju i svaka od njih ima vlastiti predložak i vlastitu funkciju. Svaka pravilno napravljena komponenta poštuje načelo enkapsulacije. Angular aplikacija sadrži stablo komponenti i svaku je komponentu moguće opet iskoristiti bilo gdje u aplikaciji [12]. Isto tako je moguće iz tog stabla izbaciti komponentu, zamijeniti ju novom ili ju samo izmijeniti jer nisu usko vezane jedna za drugu. Testiranje je moguće na razini aplikacije, ali i na razini komponenti.

Na slici 3.1. prikazan je primjer koda Angular komponente koja omogućuje unos imena i prezimena korisnika, dodavanje istog pritiskom na gumb te prikaz svih dodanih korisnika. Na slici 3.2. prikazan je kod HTML predloška te iste Angular komponente. Prezentacijski dio, logički dio i dio za stilizaciju se pišu u odvojene dokumente radi lakše preglednosti. U ovom diplomskom radu korištena je Angular verzija 13.2.

```ts
import { Component, OnInit } from '@angular/core';
interface User {
  name: string;
  surname: string;
}

@Component({
  selector: 'app-users',
  templateUrl: './users.component.html',
  styleUrls: ['./users.component.css'],
})
export class UsersComponent implements OnInit {
  users: User[];

  name: string;
  surname: string;

  constructor() {
    this.users = [];
    this.name = '';
    this.surname = '';
  }

  ngOnInit(): void {}

  addUser() {
    //Checks if input exists
    if (!this.name || !this.surname) {
      return;
    }
    //Adds user to users array
    this.users.push({
      name: this.name,
      surname: this.surname,
    });
  }
}
```

Sl. 3.1. Primjer koda Angular komponente za dodavanje i prikaz korisnika

```html
<h1>Current users</h1>
<p *ngFor="let user of users">{{ user.name + " " + user.surname }}</p>
<label>Name</label>
<input [(ngModel)]="name" />
<label>Surname</label>
<input [(ngModel)]="surname" />
<button (click)="addUser()">Add user</button>
```

Sl. 3.2. Primjer koda HTML predloška Angular komponente za dodavanje i prikaz korisnika

3.6. Node.js

Platforma otvorenog koda koja omogućuje razvojnim programerima stvaranje raznih alata i aplikacija na strani poslužitelja koristeći programski jezik JavaScript. Ima široku primjenu kod web aplikacija stvarnog vremena kao što je aplikacija opisana u ovom diplomskom radu. Baš zato što se koristi programski jezik JavaScript ne mora se gubiti puno vremena na prilagođavanje drugom programskom jeziku kada se piše programski kod sa strane klijenta i sa strane poslužitelja. Alat *Node package manager* (npm) omogućava pristup mnogim gotovim rješenjima. Node je dostupan na mnogim operacijskim sustavima i puno pružatelja web hostinga ga podržava. Ima aktivnu zajednicu razvojnih programera i dobro je dokumentiran [13]. U ovom diplomskom radu se koristi kako bi se izradio poslužitelj s kojim će klijenti komunicirati.

3.7. Express

Najpopularniji Node razvojni okvir koji omogućava definiranje rukovatelja za određene HTTP zahtjeve. Omogućava podešavanje postavki web aplikacije kao što su korišteni port za povezivanje i lokacije predložaka koji se koriste za renderiranje odgovora na zahtjev. Može se integrirati u alate za renderiranje prezentacijskog dijela kako bi se generirao odgovor koji sadrži podatke u predlošku [14].

3.8. Socket.IO

Biblioteka koja omogućava dvosmjernu komunikaciju između klijenta i poslužitelja. Napravljena je pomoću WebSocket protokola koji osigurava dvosmjerni (engl. *duplex*) kanal za komunikaciju između poslužitelja i web preglednika. Informacije se u dvosmjernom kanalu mogu prenositi u oba smjera istovremeno [15].

Ukoliko se ne može uspostaviti WebSocket veza, koristi se *HTTP Long Polling,* a to je osnovna tehnika komuniciranja za web aplikacije u stvarnom vremenu. U toj tehnici klijent šalje poslužitelju zahtjev, a poslužitelj taj zahtjev drži na čekanju sve dok ne dobije nove informacije koje zatim šalje nazad klijentu kao odgovor. Nakon što klijent primi informaciju od poslužitelja, šalje novi zahtjev i postupak se ponavlja.

Prilikom prekida veze Socket.IO će se nastojati ponovno povezati jer u sebi ima mehanizam koji periodično provjerava status veze. Paketi koji su poslani poslužitelju od strane klijenta dok je veza bila prekinuta se spremaju i šalju čim se veza ponovno uspostavi.

3.9. RxJS

Biblioteka za reaktivno programiranje. Reaktivno programiranje je koncept asinkronog programiranja koji se bavi tokovima podataka i propagacijom promjena. Ova biblioteka se koristi za rukovanje asinkronim programima i programima koji su temeljeni na događajima. Omogućava pristup *Observable*, *Observer, Subject* i još mnogim drugim tipovima podataka zajedno sa operatorima koji pomažu kod rukovanja asinkronim događajima [16]. Temeljni koncepti kojima se omogućava vođenje asinkronih radnji su:

- *Observable* – predstavlja ideju kolekcije budućih vrijednosti i događaja koju je moguće pozvati
- *Observer* – kolekcija *callback* funkcija koje osluškuju vrijednosti dobivene od strane *Observable*-a.
- *Subscription* – predstavlja izvođenje *Observable*-a i uglavnom se koristi za prekid izvođenja
- Operatori – funkcije za rukovanje kolekcijama koristeći operacije kao što su *map*, *filter*, *reduce* i slično
- *Subject* – je isto što i *EventEmitter* i jedini je način višestrukog emitiranja vrijednosti ili događaja višestrukim *Observer*-ima

3.10. PrimeNG

Kolekcija Angular komponenti otvorenog koda koje se koriste za izradu korisničkog sučelja. Razvijena je od strane tvrtke PrimeTek i prvi puta se pojavljuje 2016. godine. Sastoji se od velikog broja gotovih komponenti kao što su tablice, navigacijske trake, polja za unos i slično. Korištenjem kolekcije PrimeNG ubrzava proces izrade korisničkog sučelja.

3.11. ngx-charts

Razvojni okvir otvorenog koda za korištenje i izradu grafikona. Jedinstven je jer koristi Angular za renderiranje i animiranje SVG elemenata. Sadrži gotove grafikone spremne za korištenje kao što su linijski, površinski, plošni i još mnogi drugi. Pruža mogućnost izmjene dizajna grafikona koristeći CSS [17].

4. PROCES RAZVOJA APLIKACIJE

U ovom poglavlju je detaljno opisan proces izrade aplikacije. Važno je napomenuti da se radi o aplikaciji u stvarnom vremenu. Kako bi članovi tima mogli vidjeti tko je prisutan, kada je glasao i koliku je vrijednost odabrao potrebno je napraviti klijentsku aplikaciju koja će im omogućiti interakciju i poslužitelja koji će o svakom novom događaju obavijestiti sve klijente.

4.1. Poslužitelj

Poslužiteljska aplikacija realizirana je koristeći već objašnjeni Node.js i Express razvojni okvir. Osim njih uz pomoć *Node Package Manager*-a instaliran je alat *nodemon* koji omogućava da se poslužitelj automatski ponovno pokrene svaki puta kada mu se u uređivaču koda spreme promjene uklanjajući time potrebu za ručnim ponovnim pokretanjem. Nakon toga je omogućeno korištenje TypeScript-a i instalirana biblioteka Socket.IO koja omogućuje komunikaciju između poslužitelja i klijenta u stvarnom vremenu.

Na slici 4.1. prikazan je jednostavna implementacija poslužitelja koji prati nadolazeće veze na portu 3000. Konstanta *app* predstavlja instancu *Express* objekta koji se predaje *createServer()* metodi koja zatim vraća instancu poslužitelja. Taj se poslužitelj zatim dodjeljuje konstanti *io* na način da se preda kao argument konstruktoru klase *Server* zajedno s dodatnim postavkama. Sada se može pristupiti metodi *on()* koja dodaje funkciju za osluškivanje. Funkcija za osluškivanje (engl. *Event listener)* je funkcija koja čeka da se dogodi nekakav događaj te se zatim pokreće i omogućava pristup podacima tog događaja kako bi se s njima manipuliralo po potrebi. Instanca poslužitelja može emitirati samo jedan događaj, a to je *connection* i on se emitira kada se netko poveže s poslužiteljem. Metodom *on()* se prati kada će se pojaviti taj događaj i kada se pojavi ona se pokreće i u sebi kao argument sadržava instancu *socket-a* koji predstavlja ostvarenu vezu koja je pokrenula događaj.

```
import express, { Express } from "express";
import { createServer } from "http";
import { Server } from "socket.io";
import { ISocket } from "./models/ISocket";

const app: Express = express();
const server = createServer(app);
const io = new Server(server, {
  /* options */
});

io.on("connection", (socket: ISocket) => {
  console.log("User connected");
  console.log("New connection established, socket id is: ", socket.id);
  console.log("Socket rooms: ", socket.rooms);
});

server.listen(3000, () => {
  console.log(`Listening on port 3000`);
});
```

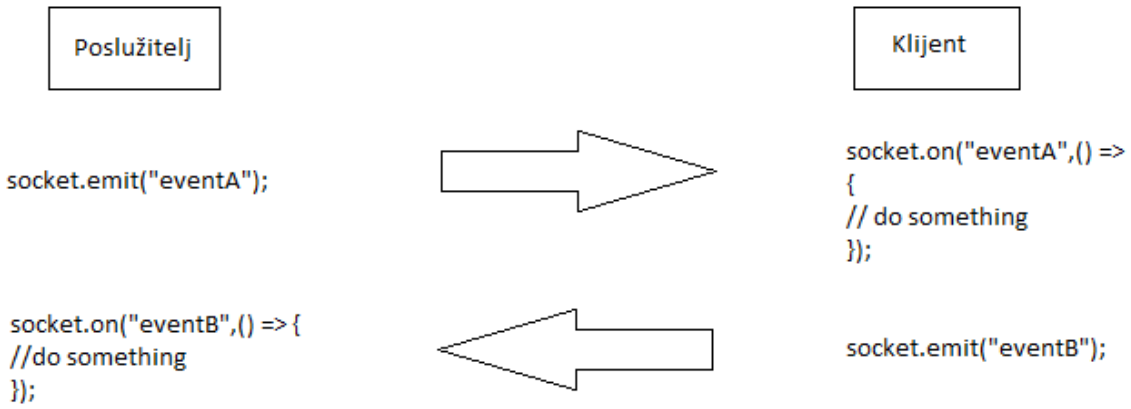
Sl. 4.1. Primjer implementacije poslužitelja

Klasa *Socket* je osnovna klasa pomoću koje će poslužitelj komunicirati s klijentima. Osim toga, ova klasa također omogućava stvaranje virtualne sobe, te ulazak i izlazak iz nje. Svaka nova veza je instanca *socket*-a i kao takva sadrži atribut *id* koji predstavlja identifikator od 20 nasumičnih znakova. Prilikom stvaranja, svaki *socket* stvara i ulazi u svoju vlastitu sobu kojoj je identifikator identičan kao i od *socket*-a (Slika 4.2.).

```
User connected
New connection established, socket id is:  GJ_LEZNBcsXNY0eOAAAB
Socket rooms:  Set(1) { 'GJ_LEZNBcsXNY0eOAAAB' }
```

Sl. 4.2. Ispis identifikatora *socket*-a i identifikatora njegove pripadajuće sobe

Kada je *socket* stvoren pomoću njega je moguće slati ili primati podatke, stvarati sobe ili im se pridružiti i izvoditi sve ostale radnje potrebne za komunikaciju. Komunikacija između poslužitelja i klijenta u ovom diplomskom radu temeljena je na emitiranju događaja i osluškivanju te reagiranju na iste. Klijent osluškuje kada će se pojaviti događaj A, kada ga poslužitelj emitira klijent će ga detektirati i manipulirati s njime po potrebi. Isto tako klijent može emitirati događaj B koji će poslužitelj zatim detektirati (Slika 4.3.).



Sl. 4.3. Prikaz komunikacije poslužitelj-klijent

Unutar funkcije koja osluškuje događaj *connection* prikazanoj na slici 4.1. implementirane su sve ostale funkcije potrebne za potpunu funkcionalnost aplikacije. Sve funkcije koje će se navesti i objasniti u nastavku služe za osluškivanje događaja od strane klijenta. Radi jednostavnosti će se u nastavku svaku funkciju identificirati po događaju kojeg osluškuju npr. događaj *room:create* označava funkciju socket.on("room:create").

4.1.1. Događaj *room:create*

Pojavom ovog događaja započinje se postupak stvaranja sobe. Važno je napomenuti da se soba stvara pomoću metode *join()*. Ova metoda omogućuje da se *socket* pridruži postojećoj sobi ili izradi novu sobu ukoliko ona ne postoji i zatim joj se pridruži. Slika 4.4. prikazuje sve pomoćne objekte koji su napravljeni kako bi se moglo pratiti stanje soba, korisnika, glasova i slično.

```
let users: { [key: string]: User[] } = {};
let votes: { [key: string]: string[] } = {};
let results: { [key: string]: Result[] } = {};
let options: { [key: string]: string[] } = {};
let tasks: { [key: string]: any } = {};
let currentTask: { [key: string]: any } = {};
```

Sl. 4.4. Pomoćni objekti za praćenje stanja soba

Svi objekti na slici 4.4. predstavljaju parove ključeva i vrijednosti gdje su ključevi svih objekata imena soba, a vrijednosti su različite. Objekt *users* za vrijednost ima polje svih korisnika koji su

trenutno u nekoj sobi. Korisnik je prikazan sučeljem *User* koje je prikazano na slici 4.5.. Svaki korisnik ima korisničko ime i identifikator.

```
export interface User {
  username: string;
  userId: string;
}
```

Sl. 4.5. Sučelje *User*

Kada bi postojala soba *testRoom* polju njenih korisnika bi se pristupilo koristeći izraz *users['testRoom']*.

Objekt *votes* za vrijednost ima polje koje sadrži identifikatore svih korisnika u istoj sobi koji su glasali. Na ovaj način će se u klijentskoj aplikaciji opisanoj u poglavlju 4.2. provjeravati koji su korisnici glasali.

Objekt *results* za vrijednost ima polje koje sadrži rezultate glasanja. Rezultat je predstavljen sučeljem *Result* koje je prikazano na slici 4.6.. Sučelje *Result* ima iste atribute kao i sučelje *User* s tim da ima dodatni atribut koji predstavlja vrijednost koju je korisnik odabrao. Korisnici i rezultati glasanja su odvojeni kako bi se izbjeglo slanje izglasanih vrijednosti sve dok glasanje ne završi.

```
export interface Result {
  username: string;
  userId: string;
  voteValue: string;
}
```

Sl. 4.6. Sučelje *Result*

Objekt *options* za vrijednost ima polje koje sadrži opcije glasanja za tu sobu odnosno karte koje će biti dodijeljene svakom korisniku.

Objekt *tasks* za vrijednost ima polje koje sadrži zadatke odnosno korisničke priče za koje je potrebno glasati. Zadatak se sastoji od opisa i identifikatora. Uz ovo ide i objekt *currentTask* koji sadrži trenutnu korisničku priču za koju se glasa u pojedinoj sobi.

Pojavom događaja *room:create* dobiva se pristup imenu sobe poslanog od strane klijenta. Provjerava se postoji li već soba s tim imenom u objektu *users* i ako postoji klijentu se koristeći *callback* funkciju daje do znanja da je soba zauzeta. Spomenuta *callback* funkcija je funkcija koja je predana drugoj funkciji kao argument, te je zatim pozvana u toj funkciji. U ovom slučaju funkcija je u klijentskoj aplikaciji predana kao argument funkcije koja emitira događaj *room:create*, ona se zatim po potrebi poziva u funkciji koja osluškuje taj isti događaj. Ovo je izvrstan način za provjeru kod rada s asinkronim operacijama kao što je ova. Kod poziva *callback* funkcije korišten je poznati uzorak u kojem prvi argument funkcije predstavlja objekt u slučaju greške, a drugi objekt u slučaju uspjeha. S ovim se objektima kasnije rukuje u klijentskoj aplikaciji. Ukoliko ime sobe ne postoji, u već spomenutim objektima se pod ključem koji predstavlja ime sobe dodjeljuju početne vrijednosti (Slika 4.7.).

```
socket.on("room:create", function (roomName: string, callback) {
  if (roomName in users) {
    callback({ type: "error", message: "Room name is taken." }, null);
    return;
  }
  users[roomName] = [];
  votes[roomName] = [];
  results[roomName] = [];
  options[roomName] = ["1", "2", "3", "5", "8", "13"];
  tasks[roomName] = [];
  currentTask[roomName] = "";
  console.log(`Room "${roomName}" created.`);
  callback(null, { type: "success", message: `Room "${roomName}" created.` });
});
```

Sl. 4.7. Funkcija za osluškivanje događaja *room:create*

4.1.2. Događaj *room:join*

Prilikom pojave ovog događaja dobiva se pristup imenu sobe odnosno *roomName* i korisničkom imenu odnosno *username*. Funkcija provjerava postoji li soba u objektu *users* i ukoliko ne postoji pomoću *callback* funkcije obavještava klijentsku aplikaciju. Isto tako provjerava ima li mjesta u sobi. Maksimalni dopušteni broj ljudi u sobi određen je konstantom *maxUsers*. Ukoliko su sve prethodne provjere prošle, *socket* koristeći već spomenutu metodu *join()* ulazi u sobu te ukoliko je ulazak uspješan, u objekt *users* te sobe se dodaje korisnik s korisničkim imenom dobivenim s klijentske aplikacije i vlastitim identifikatorom dobivenim iz instance *socket*-a. U rezultate sobe se zatim dodaje rezultat za korisnika koji se također sastoji od korisničkog imena, identifikatora, ali ima još atribut *voteValue* koji predstavlja vrijednost za koju je korisnik glasao. Zatim se

korisniku koristeći *callback* funkciju šalju opcije glasanja, sve korisničke priče i trenutna korisnička priča. Ostalim korisnicima u sobi se emitira događaj *room:user-joined* zajedno s poljem korisnika u sobi i poljem identifikatora korisnika koji su glasali *votes* (Slika 4.8.).

```
socket.on(
  "room:join",
  function (roomName: string, username: string, callback) {
    if (!(roomName in users)) {
      console.log(
        `Socket "${socket.id}" tried to join a room that doesn't exist.`
      );
      callback({ error: "404" }, null);
      return;
    }
    if (users[roomName].length >= maxUsers) {
      callback({ message: "Room is full.", type: "error" }, null);
      console.log(`Socket "${socket.id}" tried to join a room that is full.`);
      return;
    }
    socket.join(roomName);

    if (socket.rooms.has(roomName)) {
      users[roomName].push({
        username: username,
        userId: socket.id,
      });
      results[roomName].push({
        userId: socket.id,
        username: username,
        voteValue: "",
      });
      socket.roomName = roomName;
      callback(
        null,
        options[roomName],
        tasks[roomName],
        currentTask[roomName]
      );
      io.to(roomName).emit(
        "room:user-joined",
        users[roomName],
        votes[roomName]
      );
    }
  }
);
```

Sl. 4.8. Funkcija za osluškivanje događaja *room:join*

Biblioteka Socket.IO također nudi mogućnost slanja podataka svim korisnicima u sobi koristeći instancu poslužitelja *io* na način prikazan na slici 4.9..

```
io.to("testRoom").emit("notification", {
  notification: "notification content",
});
```

Sl. 4.9. Funkcija za osluškivanje događaja *room:join*

Metoda *to()* se poziva na instanci poslužitelja i predaje joj se ime sobe, zatim se poziva metoda *emit()* koja za prvi argument prima ime događaja koji se želi emitirati i nakon toga podatke koji se žele poslati. Funkcija koja osluškuje ovaj događaj nalazi se u klijentskoj aplikaciji.

4.1.3. Događaj *room:vote*

Pojavom ovog događaja na poslužitelju se uz ime sobe dobiva i *voteValue* odnosno vrijednost za koju je korisnik glasao. U polju rezultata te sobe pronalazi se korisnik kojem je identifikator jednak identifikatoru *socket*-a. Zatim se u polju *votes* pronalazi indeks elementa koji je jednak identifikatoru *socket*-a. Ukoliko je primljena vrijednost *voteValue* prazan *string* to znači da je korisnik poništio odabir vrijednosti i ukoliko je prethodno pronađen spomenuti indeks elementa, taj se element izbacuje iz polja. U polju se nalaze identifikatori korisnika koji su glasali, ako identifikatora nema to znači da korisnik nije glasao. Ukoliko je ipak primljena neka vrijednost u polje *votes* se dodaje identifikator korisnika te se njegova odabrana vrijednost sprema u rezultate. Na kraju se svim korisnicima u sobi emitira događaj *room:user-voted* zajedno sa poljem identifikatora korisnika koji su glasali (Slika 4.10.).

```
socket.on("room:vote", function (roomName: string, voteValue: string) {
  let user: any;
  if (results[roomName]) {
    user = results[roomName].find((user: any) => user.userId === socket.id);
  }

  let voteIndex;
  if (votes[roomName]) {
    votes[roomName].forEach((userId: any, index: number) => {
      if (userId === socket.id) {
        voteIndex = index;
      }
    });
    if (voteValue === "") {
      //if user sends '' then remove him from votes
      if (typeof voteIndex !== "undefined") {
        votes[roomName].splice(voteIndex, 1);
      }
    } else {
      if (typeof voteIndex !== "undefined") {
        votes[roomName][voteIndex] = socket.id;
      } else {
        votes[roomName].push(socket.id);
        console.log(`Socket "${socket.id}" pushed to votes.`);
      }
    }
    if (user) {
      user.voteValue = voteValue;
      console.log("Vote added to user: ", user);
    }
    //Send current room votes to all users in room
    io.to(roomName).emit("room:user-voted", votes[roomName]);
  }
});
```

Sl. 4.10. Funkcija za osluškivanje događaja *room:vote*

4.1.4. Događaj *room:checkIfExists*

Pojavom ovog događaja provjerava se dostupnost imena poslanog s klijentske aplikacije. Također se provjerava popunjenost sobe i ovisno o rezultatima koristeći *callback* funkcije odgovor šalje klijentskoj aplikaciji. Ova funkciji se koristi kako bi se omogućilo stvaranje ili priključivanje sobi s naslovne stranice na web aplikaciji (Slika 4.11.).

```
socket.on("room:checkIfExists", function (roomName: string, callback) {
  if (!(roomName in users)) {
    callback({ message: "Room doesn't exist.", type: "error" }, null);
    return;
  }
  if (users[roomName].length >= maxUsers) {
    callback({ message: "Room is full.", type: "error" }, null);
    console.log(`Socket "${socket.id}" tried to join a room that is full.`);
    return;
  }
  callback(null, { message: "Room exists." });
});
```

Sl. 4.11. Funkcija za osluškivanje događaja *room:checkIfExists*

4.1.5. Događaj *room:showResults*

Pojavom ovog događaja rezultati za sobu s dobivenim imenom se emitiraju zajedno s događajem *room:resultReveal* svim korisnicima u toj sobi (Slika 4.12.).

```
socket.on("room:showResults", function (roomName: string, callback) {
  if (results[roomName]) {
    console.log(`Showing results to room "${roomName}".`);
    io.to(roomName).emit("room:resultReveal", results[roomName]);
    return;
  }
  console.log(`[Error]: Results not found for room "${roomName}".`);
  callback({ message: "Results not found.", type: "error" }, null);
});
```

Sl. 4.12. Funkcija za osluškivanje događaja *room:showResults*

4.1.6. Događaj *room:newGame*

Kada se pojavi ovaj događaj polje koje sadrži identifikatore korisnika koji su glasali se ponovno postavlja te se prolazi kroz polje rezultata za sobu s dobivenim imenom i vrijednost svakog rezultata se ponovno postavlja tako da bude prazno. Zatim se trenutni zadatak odnosno korisnička priča za tu sobu postavlja na idući u nizu ukoliko on postoji. Svim korisnicima u sobi se zatim emitira događaj *room:newGameCreated* zajedno s poljem korisnika u sobi, praznim poljem izglasanih vrijednosti i identifikatorom trenutne korisničke priče (Slika 4.13.).

```
socket.on("room:newGame", function (roomName: string, callback) {
  if (users[roomName]) {
    votes[roomName] = [];
    results[roomName].forEach((result: any) => {
      result.voteValue = "";
    });
    console.log(`Results for room "${roomName}" reset.`);
    //Switch to next task
    if (tasks[roomName]) {
      let currentTaskIndex = tasks[roomName].findIndex(
        (task: any) => currentTask[roomName] === task.id
      );
      if (
        currentTaskIndex >= 0 &&
        currentTaskIndex < tasks[roomName].length - 1
      ) {
        currentTask[roomName] = tasks[roomName][currentTaskIndex + 1].id;
      } else {
        currentTask[roomName] = null;
      }
    }
    io.to(roomName).emit(
      "room:newGameCreated",
      users[roomName],
      votes[roomName],
      currentTask[roomName]
    );
    return;
  }
  callback({ message: "Room not found.", type: "error" }, null);
});
```

Sl. 4.13. Funkcija za osluškivanje događaja *room:newGame*

4.1.7. Događaj *room:setOptions*

Ova funkcija za zadatak ima postaviti opcije za glasanje odnosno karte za korisnike u sobi. Od klijentske aplikacije se dobije ime sobe i opcije za glasanje u obliku polja. Opcije se spremaju u objekt *options*, a zatim se svim korisnicima u sobi emitira događaj *room:optionsChanged* zajedno s poljem koje predstavlja opcije za tu sobu (Slika 4.14.).

```
socket.on(
  "room:setOptions",
  function (roomName: string, voteOptions: string[], callback) {
    if (users[roomName] && voteOptions) {
      options[roomName] = voteOptions;
      console.log(`[${roomName}]: changed vote options to [${voteOptions}]`);
      io.to(roomName).emit("room:optionsChanged", options[roomName]);
      return;
    }
    callback(
      {
        message: "Error occurred while setting options",
        type: "error",
      },
      null
    );
  }
);
```

Sl. 4.14. Funkcija za osluškivanje događaja *room:setOptions*

4.1.8. Događaj *user:changeName*

Ova funkcija omogućava korisniku da promjeni svoje korisničko ime. Pojavom događaja *user:changeName* prolazi se kroz polje korisnika i polje rezultata te se pronalaze indexi korisnika i rezultata kojima je identifikator isti kao identifikator *socket*-a. Zatim se u spomenutim poljima na pronađenim indeksima postavlja korisničko ime na ono dobiveno iz klijentske aplikacije. Svim korisnicima u sobi se potom emitira događaj *room:user-joined* zajedno sa poljem koje sadržava trenutne korisnike u sobi i poljem koje sadržava izglasane vrijednosti (Slika 4.15.). Spomenuti događaj je ponovno emitiran jer sadrži svu potrebnu funkcionalnost i nema potreba za emitiranjem novog događaja za kojeg bi na klijentskoj aplikaciji bilo potrebno implementirati novu funkciju za osluškivanje.

```
socket.on(
  "user:changeName",
  function (roomName: string, name: string, callback) {
    if (users[roomName]) {
      let userIndex = users[roomName].findIndex(
        (user: any) => user.userId === socket.id
      );
      let resultIndex = users[roomName].findIndex(
        (user: any) => user.userId === socket.id
      );
      if (userIndex >= 0 && resultIndex >= 0) {
        users[roomName][userIndex].username = name;
        results[roomName][userIndex].username = name;
        io.to(roomName).emit(
          "room:user-joined",
          users[roomName],
          votes[roomName]
        );
        callback(null, {
          message: "Display name changed successfully.",
          type: "success",
        });
        return;
      }
      callback(
        {
          message: "User not found.",
          type: "error",
        },
        null
      );
      return;
    }
    callback(
      {
        message: "Error occurred while changing username.",
        type: "error",
      },
      null
    );
  }
);
```

Sl. 4.15. Funkcija za osluškivanje događaja *user:changeName*

4.1.9. Događaj *room:setTasks*

Ova funkcija omogućava dodavanje popisa korisničkih priča koje će proći kroz postupak glasanja. Pojavom događaja *room:setTasks* dobiva se pristup imenu sobe, polju koje sadrži korisničke priče i identifikatoru trenutno aktivne korisničke priče. Zatim se u objekt *tasks* sprema dobiveno polje, a u objekt *currentTask* identifikator trenutno aktivne korisničke priče. Svim korisnicima u sobi se zatim emitira događaj *room:tasksChanged* zajedno sa poljem koje predstavlja popis korisničkih priča i identifikatorom trenutno aktivnog zadatka (Slika 4.16.).

```
socket.on(
  "room:setTasks",
  function (
    roomName: string,
    roomTasks: any[],
    currentTaskId: string,
    callback
  ) {
    if (users[roomName] && roomTasks) {
      tasks[roomName] = roomTasks;
      currentTask[roomName] = currentTaskId;
      console.log(
        `[${roomName}]: changed room tasks to [${JSON.stringify(roomTasks)}]`
      );
      io.to(roomName).emit(
        "room:tasksChanged",
        tasks[roomName],
        currentTask[roomName]
      );
      callback(null, {
        message: "Tasks updated",
        type: "success",
      });
      return;
    }
    callback(
      {
        message: "Error occurred while setting tasks",
        type: "error",
      },
      null
    );
  }
);
```

Sl. 4.16. Funkcija za osluškivanje događaja *room:setTasks*

,

4.1.10. Događaj *room:setCurrentTask*

Pojavom ovog događaja dobiva se pristup imenu sobe i identifikatoru trenutno aktivne korisničke priče *currentTaskId* od strane klijenta. U objekt *currentTask* se sprema dobiveni identifikator te se svim korisnicima u sobi emitira događaj *room:currentTaskChanged* zajedno sa spomenutim identifikatorom (Slika 4.17.).

```
socket.on(
  "room:setCurrentTask",
  function (roomName: string, currentTaskId: string, callback) {
    if (users[roomName]) {
      currentTask[roomName] = currentTaskId;
      console.log(
        `[${roomName}]: changed room current task to [${currentTaskId}]`
      );
      io.to(roomName).emit("room:currentTaskChanged", currentTask[roomName]);
      callback(null, {
        message: "Current task updated",
        type: "success",
      });
      return;
    }
    callback(
      {
        message: "Error occurred while setting tasks",
        type: "error",
      },
      null
    );
  }
);
```

Sl. 4.17. Funkcija za osluškivanje događaja *room:setCurrentTask*

4.1.11. Događaj *room:assignStoryPoints*

Svakoj korisničkoj priči je potrebno nakon glasanja dodijeliti određeni broj bodova priče. Ova funkcija omogućava upravo to. Pojavom događaja *room:assignStoryPoints* provjerava se postoje li uopće korisničke priče u sobi te ako postoje pronalazi se indeks korisničke priče dobivene od strane klijentske aplikacije. Ukoliko je indeks pronađen korisničkoj priči u objektu *tasks* na tom indeksu se postavlja vrijednost bodova priče na onu dobivenu od klijentske aplikacije odnosno *storyPoints*. Ukoliko indeks nije pronađen vraća se objekt s odgovarajućom porukom (Slika 4.18.).

```
socket.on(
  "room:assignStoryPoints",
  function (
    roomName: string,
    currentTaskId: string,
    storyPoints: number,
    callback
  ) {
    if (!tasks[roomName]) {
      callback(
        {
          message: "Tasks not found.",
          type: "error",
        },
        null
      );
      return;
    }
    let currentTaskIndex = tasks[roomName].findIndex(
      (task: any) => currentTaskId === task.id
    );
    if (currentTaskIndex >= 0) {
      tasks[roomName][currentTaskIndex].storyPoints = storyPoints;
      io.to(roomName).emit(
        "room:tasksChanged",
        tasks[roomName],
        currentTask[roomName]
      );
    } else {
      callback(
        {
          message: "Task not found",
          type: "error",
        },
        null
      );
      return;
    }
  }
);
```

Sl. 4.18. Funkcija za osluškivanje događaja *room:assignStoryPoints*

4.1.12. Događaj *disconnect*

Kada neki od korisnika prekine vezu s poslužiteljem pojavljuje se događaj *disconnect*. Ova funkcija zatim pomoću identifikatora *socket*-a uklanja korisnika iz objekta *votes*, *users* i *results*. Također se provjerava sadrži li soba iz koje je korisnik izašao barem jednog korisnika. Ukoliko je soba prazna ona se uklanja iz svih objekata. Ukoliko soba nije prazna svim preostalim korisnicima

u njoj se emitira događaj *room:user-joined* zajedno sa ažuriranim poljem korisnika i identifikatorima onih koji su glasali (Slika 4.19.).

```
socket.on("disconnect", () => {
  if (socket.roomName) {
    votes[socket.roomName].forEach((userId: any, index: number) => {
      if (userId === socket.id) {
        votes[socket.roomName!].splice(index, 1);
      }
    });
    users[socket.roomName].forEach((object: any, index: number) => {
      if (object.userId === socket.id) {
        users[socket.roomName!].splice(index, 1);
      }
    });
    results[socket.roomName].forEach((object: any, index: number) => {
      if (object.userId === socket.id) {
        results[socket.roomName!].splice(index, 1);
      }
    });
    if (users[socket.roomName].length < 1) {
      delete users[socket.roomName];
      delete votes[socket.roomName];
      delete results[socket.roomName];
      delete options[socket.roomName];
      delete tasks[socket.roomName];
      delete currentTask[socket.roomName];
      return;
    }
    //Send the updated list of users and votes to current users
    io.to(socket.roomName).emit(
      "room:user-joined",
      users[socket.roomName],
      votes[socket.roomName]
    );
  }
});
```

Sl. 4.19. Funkcija za osluškivanje događaja *disconnect*

4.2. Klijentska aplikacija

Klijentska aplikacija realizirana je pomoću Angular programskog okvira. Napravljena aplikacija je *single-page* aplikacija što znači da se u pregledniku učitava samo jedna web stranica kojoj se pri interakciji s korisnikom dinamički mijenja sadržaj. To znači da je korisnicima omogućeno korištenje web stranice bez da se stranica svaki puta nanovo učitava sa servera. Samim time ubrzava se rad stranice i korisnicima poboljšava iskustvo. Ovo naravno ima i svoje negativne strane kao što je lošija optimizacija web stranice (engl. *Search engine optimization*) za tražilice i otežano praćenje rada stranice.

4.2.1. Struktura aplikacije

Pokretanje Angular aplikacije započinje u datoteci *main.ts* koja učitava korijenski modul koji se prema konvenciji naziva *AppModule*. Modul je mehanizam za grupiranje direktiva, komponenti, servisa i svih ostalih povezanih implementacija kako bi zajedno činili aplikaciju. Klasu u Angularu je moguće pretvoriti u modul dodavanjem *@NgModule* dekoratera u nju. Na slici 4.20. prikazan je korijenski modul aplikacije bez izraza koji definiraju putanje sa kojih se uvoze navedene stavke.

```
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    RoomComponent,
    VoteComponent,
    CopyToClipboardComponent,
    UsernameDialogComponent,
    NonExistingRoomDialogComponent,
    OptionsDialogComponent,
    VotingResultsComponent,
    TaskDialogComponent,
  ],
  imports: [
    BrowserModule,
    BrowserAnimationsModule,
    AppRoutingModule,
    CardModule,
    InputTextModule,
    DividerModule,
    ButtonModule,
    ReactiveFormsModule,
    ToastModule,
    TooltipModule,
    DropdownModule,
    NgxChartsModule,
    TableModule,
    FormsModule,
    InputTextareaModule,
  ],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Sl. 4.20. Korijenski modul aplikacije

Dekorater *@NgModule* može primiti vrijednosti:

- *imports* – polje uvezenih modula koje se koriste unutar trenutnog modula
- *declarations* – polje komponenti koje se nalaze u modulu
- *exports* – polje komponenti modula i servisa koje se izvoze odnosno daju na korištenje ostalim modulima ukoliko uvezu ovaj modul
- *bootstrap* – ovdje se definira korijenska komponenta odnosno komponenta koja se pokreće kod pokretanja modula, iako se radi o polju u većini slučajeva će se raditi samo o jednoj komponenti

Građevni blokovi Angular aplikacije su komponente. Klasa se pretvara u komponentu kada joj dodamo dekorater *@Component*. Najčešće vrijednosti koje on prima su:

- *selector* – CSS selektor koji omogućava identifikaciju komponente u HTML predlošku
- *templateUrl* – relativna ili apsolutna putanja do HTML predloška za Angular komponentu
- *styleUrls* – polje relativnih ili apsolutnih putanja do datoteka koje sadrže CSS kod za primjenu u komponenti

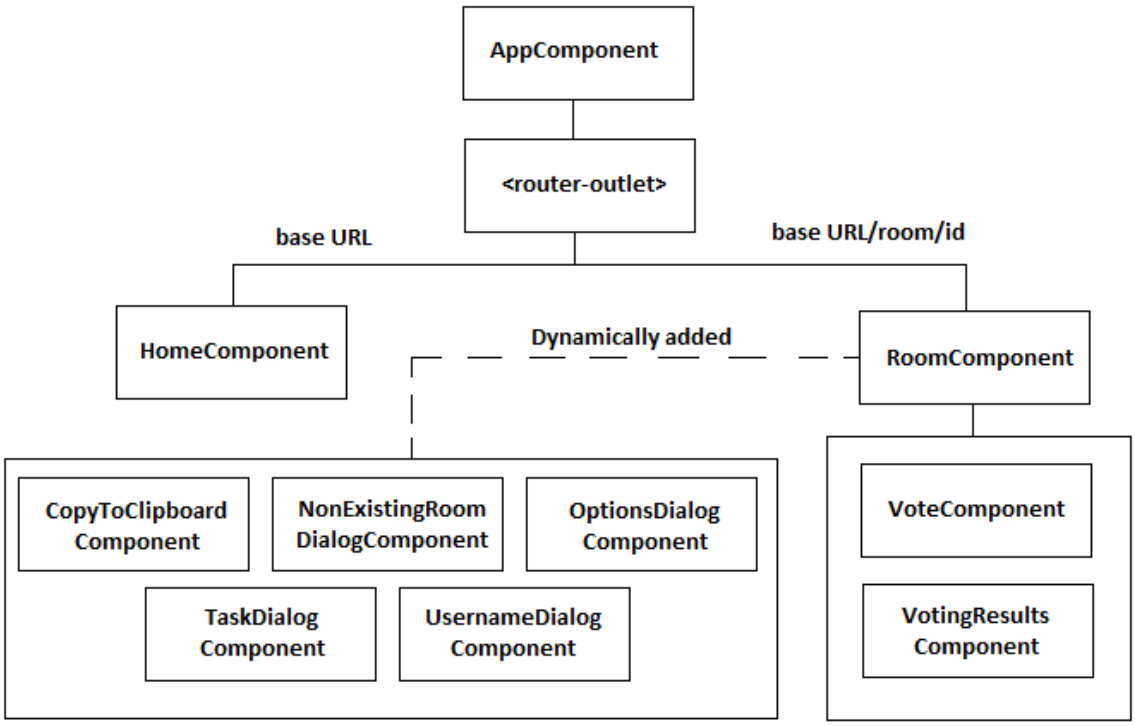
Pokretanjem korijenskog modula pokreće se komponenta *AppComponent* koja u predlošku sadrži oznake <router-outlet></router-outlet>. Ove oznake služe kao *placeholder* kojeg će Angular dinamički popuniti ovisno o trenutnoj putanji. Na slici 4.20. u polju *imports* nalazi se *AppRoutingModule*.

AppRoutingModule je modul u kojem se definiraju putanje do određenih komponenti. Na ovaj način Angular zna s kojim sadržajem popuniti stranicu. Komponente i njihove putanje definirane su u konstanti *routes*. Kod osnovnog URL-a i kod URL-a koji ne postoji unutar aplikacije učitat će se *HomeComponent*, a kod URL-a *room/id* gdje je *id* ime sobe učitat će se *RoomComponent* (Slika 4.21.).

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './pages/home/home.component';
import { RoomComponent } from './pages/room/room.component';

const routes: Routes = [
  {
    path: '',
    component: HomeComponent,
  },

  {
    path: 'room/:id',
    component: RoomComponent,
  },
  {
    path: '**',
    redirectTo: '',
  },
];

...
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

Sl. 4.21. Prikaz *AppRoutingModule*-a

Kao što je prethodno spomenuto u HTML predlošku komponente *AppComponent* dinamički će se izmjenjivati sadržaj ovisno o putanji. Naslovnu stranicu predstavlja komponenta *HomeComponent*, a virtualnu sobu komponenta *RoomComponent*. Unutar komponente *RoomComponent* nalaze se sve ostale komponente ili su dinamički dodane pri interakciji s korisnikom. Struktura odnosno stablo komponenti prikazano je na slici 4.22..



Sl. 4.22. Stablo komponenti

4.2.2. Servis *GameService*

Servis u Angularu je klasa s dobro definiranom svrhom u koje se dodaje *@Injectable* dekorater zajedno s pripadajućim meta podacima. Moguće ga je koristiti u bilo kojoj drugoj komponenti zahvaljujući injekciji ovisnosti (engl. *Dependency injection*). *Dependency injection* je oblikovni obrazac u programiranju u kojem klasa dobiva ovisnosti iz vanjskih izvora umjesto da ih stvara. Injektiranje ovisnosti se u Angularu vrši tako da se konstruktoru neke klase preda argument tipa te ovisnosti. Time ovisnost postaje vidljiva u toj komponenti.

Glavni servis klijentske aplikacije je *GameService* i u njemu se nalazi sva funkcionalnost potrebna za komunikaciju s poslužiteljem opisanim u poglavlju 4.1.. U njemu se uz pomoć biblioteke RxJS manipulira podacima. Na slici 4.23. prikazani korišteni objekti predstavljaju tokove podataka gdje *users$* predstavlja polje korisnika u sobi, *votes$* polje identifikatora korisnika koji su glasali, *tasks$* polje korisničkih priča, *currentTask$* identifikator trenutno aktivne korisničke priče i *options$* polje opcija za glasanje odnosno karti koje će biti dodijeljene svakom korisniku. Svi oni su dobiveni iz odgovarajućih *Subject* ili *ReplaySubject* objekata koji su i zapravo *Observable i*

Observer i imaju mogućnost poziva metode *next()* koja će svakom od prethodno navedenih *Observable*-a proslijediti vrijednost predanu kao argument.

```
public usersSubject: ReplaySubject<any> = new ReplaySubject(1);
public votesSubject: Subject<any> = new Subject();
public optionSubject: Subject<String[]> = new Subject();
public taskSubject: ReplaySubject<any> = new ReplaySubject(1);
public currentTaskSubject: ReplaySubject<any> = new ReplaySubject(1);
public notificationSubject: Subject<any> = new Subject();

public users$ = this.usersSubject.asObservable();
public votes$ = this.votesSubject.asObservable();
public tasks$ = this.taskSubject.asObservable();
public currentTask$ = combineLatest([
  this.tasks$,
  this.currentTaskSubject.asObservable(),
]).pipe(
  map(([tasks, currentTask]) => {
    let currentTaskObject = tasks.find(
      (task: any) => task.id === currentTask
    );
    if (currentTaskObject !== -1) {
      return currentTaskObject;
    }
  })
);

public options$ = this.optionSubject.asObservable().pipe(
  map((options: String[]) =>
    options.map((x) => {
      return {
        value: x,
        selected: false,
      };
    })
  )
);

private voteOptions: String[] = [];

public get clonedVoteOptions() {
  return Object.assign([], this.voteOptions);
}
```

Sl. 4.23. Korišteni objekti u servisu *GameService*

GameService u sebi sadrži instancu klase *Socket* iz *Socket.IO-client* biblioteke koju koristi za komunikaciju s poslužiteljem. Ova instanca klase *Socket* isto kao i ona na poslužitelju sadrži metodu za osluškivanje događaja *on()* i metodu za emitiranje događaja *emit()*. Servis također sadrži *notificationSubject$* koji se koristi za prikaz notifikacija s poslužitelja u obliku poruka na zaslonu.

Metode koje servis daje na korištenje komponentama biti će opisane u nastavku zajedno s komponentom u kojoj se koriste.

4.2.3. Komponenta *HomeComponent*

HomeComponent je komponenta koja predstavlja naslovnu stranicu aplikacije. Unutar nje se pokreće postupak povezivanja s poslužiteljem te se pretplaćuje (engl. *Subscribe*) na *notificationSubject* iz servisa *GameService*. Svaka obavijest koja se dobije od strane poslužitelja se zatim prikazuje pomoću *MessageService*-a biblioteke PrimeNG u obliku poruke na zaslonu. Ova komponenta omogućava stvaranje sobe koristeći *createRoom()* funkciju iz servisa (Slika 4.24.). Navedena funkcija poslužitelju šalje događaj *room:create* te koristeći *callback* funkciju dobiva povratnu informaciju u obliku *success* i *error* objekta. Komponenta s obzirom na povratnu informaciju usmjerava korisnika na komponentu *RoomComponent* koja pokreće postupak pridruživanja napravljenoj sobi ili mu prikazuje poruku s opisom greške.

```
createRoom(roomName: string, callback: any) {
  this.socket.emit('room:create', roomName, (error: any, success: any) => {
    if (error) {
      callback(error, null);
      return;
    }
    if (success) {
      callback(null, success);
    }
  });
}
```

Sl. 4.24. Funkcija *createRoom()*

Komponenta također omogućava korisniku da se pridruži postojećoj sobi, ali na način da provjeri postoji li soba kojoj se korisnik želi pridružiti, te ako postoji sprema korisnikovo ime u lokalnu memoriju i usmjerava ga na komponentu *RoomComponent* u kojoj se zatim poziva funkcija *joinRoom()* iz servisa (Slika 4.25.). Ova funkcija pokreće postupak pridruživanja sobi tako što poslužitelju šalje događaj *room:join* te ukoliko nije došlo do greške preko *callback* funkcije dobiva od poslužitelja potrebne podatke.

```
joinRoom(roomName: string, displayName: string, callback: any) {
  this.socket.emit(
    'room:join',
    roomName,
    displayName,
    (err: any, options: string[], tasks: any[], currentTaskId: string) => {
      if (err) {
        callback(err.error, null);
        this.notificationSubject.next(err.error);
        return;
      }
      if (options) {
        callback(null, options);
        this.voteOptions = options;
        this.optionSubject.next(options);
      }
      if (tasks) {
        this.taskSubject.next(tasks);
      }
      if (currentTaskId) {
        this.currentTaskSubject.next(currentTaskId);
      }
    }
  );
}
```

Sl. 4.25. Funkcija *joinRoom()*

Ako se korisnik uspješno pridružio sobi, svim korisnicima u sobi poslan je događaj *room:user-joined* koji pokreće funkciju *onRoomJoin()* koja od poslužitelja dobiva polje korisnika u sobi i polje identifikatora korisnika koji su glasali (Slika 4.26.). Dobivene vrijednosti su zatim proslijeđene odgovarajućim objektima.

```
onRoomJoin() {
  this.socket.on('room:user-joined', (users: any, votes: any) => {
    if (users) {
      this.usersSubject.next(users);
    }
    if (votes) {
      this.votesSubject.next(votes);
    }
  });
}
```

Sl. 4.26. Funkcija *onRoomJoin()*

Na slici 4.27. prikazana je implementacija opisane komponente.

```
ngOnInit(): void {
  this.gameService.connect();
  this.errorSubscription = this.gameService.notificationSubject
    .asObservable()
    .pipe(
      tap((notification) => {
        if (notification) {
          this.messageService.add({
            severity: notification.type,
            summary: notification.message,
          });
        }
      })
    )
    .subscribe();
}

joinRoomPressed() {
  let values = this.formGroup.value;
  this.gameService.checkIfRoomExists(
    values.joinRoomName,
    (error: any, success: any) => {
      //if room doesnt exist returns error object from server, else returns success
      if (error) {
        this.gameService.notificationSubject.next(error);
        return;
      }
      if (success) {
        localStorage.setItem('pp:username', values.displayName);
        this.router.navigate([`room/${values.joinRoomName}`]);
      }
    }
  );
}

createRoomPressed() {
  let values = this.formGroup.value;
  this.gameService.checkIfRoomExists(
    values.createRoomName,
    (error: any, success: any) => {
      //if room doesnt exist returns error object from server, else returns success
      if (success) {
        this.gameService.notificationSubject.next({
          message: 'Room name is taken.',
          type: 'error',
        });
        return;
      }
      if (error) {
        this.gameService.createRoom(
          values.createRoomName,
          (error: any, success: any) => {
            if (error) {
              this.gameService.notificationSubject.next(error);
              return;
            }
            if (success) {
              localStorage.setItem('pp:username', values.displayName);
              this.router.navigate([`room/${values.createRoomName}`]);
            }
          }
        );
      }
    }
  );
}
```

Sl. 4.27. Implementacija opisane funkcionalnosti komponente *HomeComponent*

4.2.4. Komponenta *RoomComponent*

Ova komponenta predstavlja virtualnu sobu i sve što korisnici vide kada uđu u nju. Korisnici do sobe dolaze na dva moguća načina. Prvi način je da naprave sobu ili joj se pridruže koristeći

naslovnu stranicu, a drugi način je putem URL veze. Kada se ova komponenta stvori iz trenutnog URL-a se izdvaja ime sobe i šalje poslužitelju koristeći metodu *joinRoom()* te se ovisno o odgovoru poslužitelja korisniku prikazuje sadržaj. Ukoliko korisnik nije prethodno bio na stranici, a sobi želi pristupiti preko URL veze, od korisnika se prvo zahtjeva da unese korisničko ime. Implementacija opisane funkcionalnosti prikazana je na slici4.28..

```
ngOnInit(): void {
  this.gameService.connect();
  this.roomName = this.route.snapshot.url[1].path;
  this.gameService.options$
    .pipe(
      tap((options) => {
        this.voteOptions = options;
      })
    )
    .subscribe();
  //Username check
  this.username = localStorage.getItem('pp:username') ?? '';
  //Listener for connection
  this.gameService.onConnect((connected: any) => {
    if (connected) {
      if (this.username && this.username.trim() !== '') {
        this.connect();
        this.gameService.joinRoom(
          this.roomName,
          this.username,
          (error: any, options: any) => {
            if (error) {
              this.loggedIn = false;
              return;
            }
            if (options) {
              this.loggedIn = true;
            }
          }
        );
      } else {
        this.showUsernameDialog();
      }
    }
  });
}
```

Sl. 4.28. Implementacija procesa za ulazak korisnika u sobu

Nakon što je korisnik ušao u sobu i poslužitelj poslao sve potrebne podatke, ti su podaci prikazani koristeći ostale komponente. Popis trenutnih korisnika u sobi prikazan je u obliku kartica koristeći komponentu *VoteComponent*.

Vote Component je komponenta koja prima polje korisnika u sobi i polje identifikatora korisnika koji su glasali. Zatim prikazuje korisnike u obliku kartica od kojih svaka ima odgovarajuće ime korisnika. Također omogućava razlikovanje korisnika koji su glasali od onih koji nisu te prikaz rezultata glasanja (Slika 4.29.).

```
<div class="voting-content">
  <div class="user" *ngFor="let user of users">
    <div class="card" [class.card-selected]="userVoted(user.userId)">
      <b> {{ user.voteValue ? user.voteValue : "" }} </b>
    </div>
    <span class="username"
      ><b>{{ user.username }}</b></span
    >
  </div>
</div>
```

Sl. 4.29. HTML predložak *VoteComponent* komponente

Karte s mogućim vrijednostima za glasanje (Slika 4.30.) kao i funkcije koje to omogućavaju nalaze se unutar *RoomComponent* komponente.

```
<!--Voting options-->
<div class="vote-content">
  <div *ngIf="cardsRevealed" class="voting-results">
    <app-voting-results
      (voteAvg)="assignStoryPoints($event)"
    ></app-voting-results>
  </div>
  <ng-container *ngIf="!cardsRevealed">
    <div class="lower-right">
      <p-button
        pTooltip="Vote options"
        tooltipPosition="top"
        icon="pi pi-cog"
        (click)="showOptionsDialog()"
        styleClass="p-button p-button-lg"
      ></p-button>
    </div>
    <div class="vote-options">
      <div
        *ngFor="let option of voteOptions; let indexOfCard = index"
        class="card"
        [class.card-selected]="option.selected"
        (click)="cardSelected(indexOfCard)"
      >
        <b> {{ option.value ? option.value : "" }} </b>
      </div>
    </div>
  </ng-container>
</div>
```

Sl. 4.30. HTML kod za prikaz karata za glasanje

Svaka soba kada se napravi ima zadane vrijednosti na kartama za glasanje. Za promjenu tih vrijednosti koristi se *OptionsDialogComponent* komponenta.

OptionsDialogComponent komponenta pojavljuje se u obliku modalnog prozora i implementirana je koristeći modul *DynamicDialogModule* iz PrimeNG biblioteke. Kako bi se modalni prozor mogao koristiti u komponenti *RoomComponent* je u meta podatke dekoratora *@Component* pod *providers* potrebno dodati *DialogService*. Ova komponenta omogućava korisniku unos vlastitih vrijednosti koje će biti prikazane u obliku karata. Vrijednosti se unose jedna za drugom i odvojene su zarezom. Komponenta zatim provjerava korisnikov unos i ako je sve u redu omogućava korisniku da pritisne na gumb koji poslužitelju šalje unesene vrijednosti u obliku polja. Također nudi mogućnost odabira neke od zadanih vrijednosti u obliku padajućeg izbornika. Vrijednosti se poslužitelju šalju koristeći *setOptions()* metodu iz servisa nakon čega poslužitelj vraća svim korisnicima nove vrijednosti karata za glasanje uz pomoć *onOptionsChange()* metode. HTML predložak komponente prikazan je na slici 4.31..

```
<form [formGroup]="formGroup" (submit)="hideDialog()">
  <div class="form-item">
    <p-dropdown
      [options]="voteOptions"
      formControlName="selectedVoteOptions"
      optionLabel="name"
      optionValue="value"
      appendTo="body"
      (onChange)="onOptionSelect()"
    ></p-dropdown>
  </div>

  <div class="form-item" *ngIf="formGroup.value.selectedVoteOptions === 1">
    <input
      placeholder="A,B,C,D"
      pInputText
      formControlName="customVoteOptions"
    />
    <label class="form-hint"
      >Maximum is 15 CSV values, 3 character per value.</label
    >
  </div>

  <div class="button-container">
    <button [disabled]="!formGroup.valid" pButton label="Save"></button>
  </div>
</form>
```

Sl. 4.31. HTML predložak *OptionsDialogComponent* komponente

Kada korisnik odabere jednu od ponuđenih karata poslužitelju se šalje vrijednost odabrane karte uz pomoć funkcije *vote()* iz servisa *GameService.* Svi korisnici zatim uz pomoć funkcije

onUserVote() dobivaju od poslužitelja novo polje identifikatora korisnika koji su glasali koje zatim komponenta *VoteComponent* koristi za prikazivanje istih. Kada svi korisnici glasaju, pritiskom na gumb *Show cards* koje je korisnicima vidljivo karte za glasanje se skrivaju i prikazuju se rezultati glasanja u obliku dijagrama i dodatnih informacija. Za ovo se koristi komponenta *VotingResultsComponent*.

VotingResultsComponent je komponenta koja uzima rezultate glasanja, računa prosjek i uzima vrijednost iz mogućih vrijednosti za glasanje koja je najbliža prosjeku. Zatim ju prikazuje zajedno s grafikonom i mogućim vrijednostima za glasanje od kojih svaka vrijednost uz sebe ima postotak korisnika koji su za tu vrijednost glasali (Slika 4.32.). Ova komponenta zatim šalje izglasanu vrijednost komponenti *RoomComponent* koja ju šalje poslužitelju.

```
<div class="container">
  <span class="avg"
    ><label>Average:</label> <b>{{ result }}</b></span
  >
  <ngx-charts-advanced-pie-chart
    [view]="view"
    [scheme]="scheme"
    [results]="data"
    [label]="label"
  >
  </ngx-charts-advanced-pie-chart>
</div>
```

Sl. 4.32. HTML predložak *VotingResultsComponent* komponente

Trenutno aktivna korisnička prikazana je također unutar komponente *RoomComponent*. Za upravljanje korisničkim pričama koristi se komponenta *TaskDialogComponent*.

TaskDialogComponent komponenta pojavljuje se u obliku modalnog prozora i omogućava dodavanje, brisanje i uređivanje korisničkih priča. Također omogućava izmjenu redoslijeda njihovog pojavljivanja i postavljanje trenutno aktivne korisničke priče. Koristi se kopija korisničkih priča na kojoj se prate promjene koje je korisnik napravio. Ukoliko korisnik nije napravio nikakve izmjene već je samo postavio neku drugu priču kao aktivnu poslužitelju će se poslati događaj *room:setCurrentTask* koji će zatim korisnicima poslati identifikator nove trenutno aktivne korisničke priče. Ukoliko je korisnik izmijenio korisničke priče na način da je dodao novu, obrisao postojeću ili promjenio njihov redoslijed, poslužitelju se šalje događaj *room:setTasks* koji svim korisnicima šalje novi popis korisničkih priča.

Korištenjem prethodno navedenih komponenti ostvarena je osnovna funkcionalnost aplikacije. Ostale komponente koje će se navesti u nastavku napravljene su kako bi korisniku poboljšale iskustvo korištenja aplikacije.

CopyToClipboardComponent je komponenta koja pojavljuje u obliku modalnog prozora i korisniku olakšava dijeljenje URL veze sobe u kojoj se nalazi na način da spremi URL kod korisnika u međuspremnik (engl. *clipboard*) kada korisnik pritisne na predviđeni gumb. Zatim ga korisnik može zalijepiti i poslati nekome. Ovaj modalni prozor poziva se iz komponente *RoomComponent*. Na slici 4.33. prikazan je HTML predložak komponente.

```
<div class="container">
  <input
    readonly="readonly"
    pInputText
    class="p-inputtext-sm url"
    [value]="currentUrl"
  />
  <p-button (onClick)="hideDialog()" label="Copy to clipboard"></p-button>
</div>
```

Sl. 4.33. HTML predložak *CopyToClipboardComponent* komponente

NonExistingRoomDialogComponent je komponenta koja korisnika obavještava da ne postoji soba kojoj se pokušao pridružiti. Ova komponenta se u obliku modalnog prozora poziva iz komponente *RoomComponent* kada se od poslužitelja dobije povratna informacija da soba nije pronađena te korisniku u obliku poruke govori isto.

UsernameDialogComponent je komponenta koja omogućava korisniku da promjeni korisničko ime. Također se u obliku modalnog prozora poziva iz komponente *RoomComponent* i nakon što korisnik unese novo željeno korisničko ime šalje poslužitelju događaj *room:changeName* koristeći funkciju *changeName()* iz servisa. Poslužitelj zatim šalje nazad popis korisnika sa ažuriranim korisničkim imenom. Na slici 4.34. prikazan je HTML predložak komponente.

```
<div class="container">
  <input pInputText [formControl]="username" placeholder="Name" />
  <p-button
    [disabled]="!formValid()"
    (onClick)="hideDialog()"
    label="Save"
  ></p-button>
</div>
```

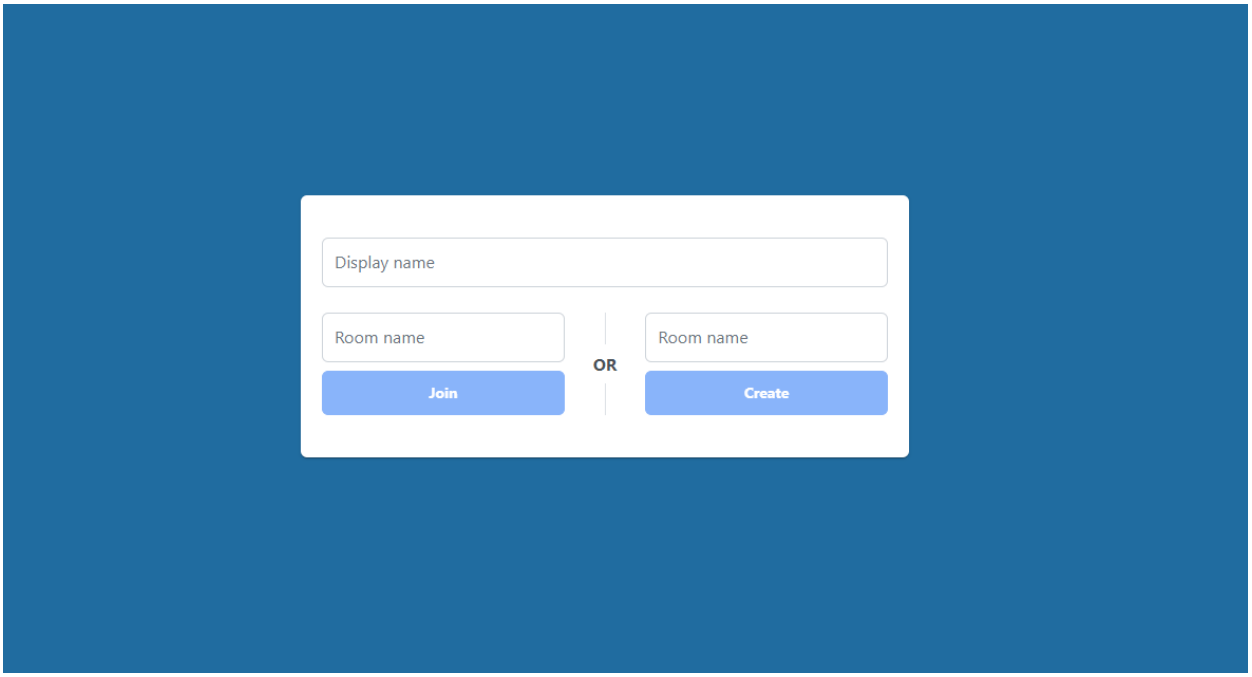
Sl. 4.34. HTML predložak *UsernameDialogComponent* komponente

5. IZGLED I NAČIN RADA APLIKACIJE

U nastavku će se objasniti način rada web aplikacije uz pomoć slika korisničkog sučelja.

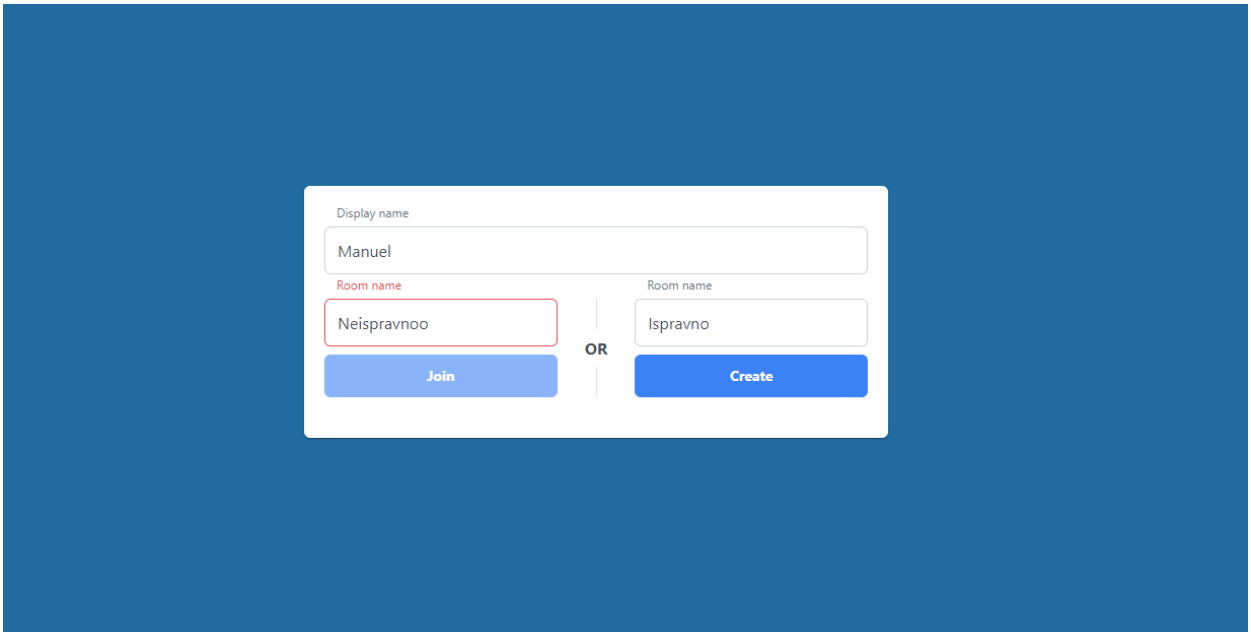
5.1. Početna stranica

Prilikom dolaska na stranicu prikazuje se početna stranica koja sadrži polje za unos korisničkog imena te polje za unos imena sobe koju se želi stvoriti ili kojoj se želi pridružiti (Slika 5.1.).



Sl. 5.1. Izgled početne stranice

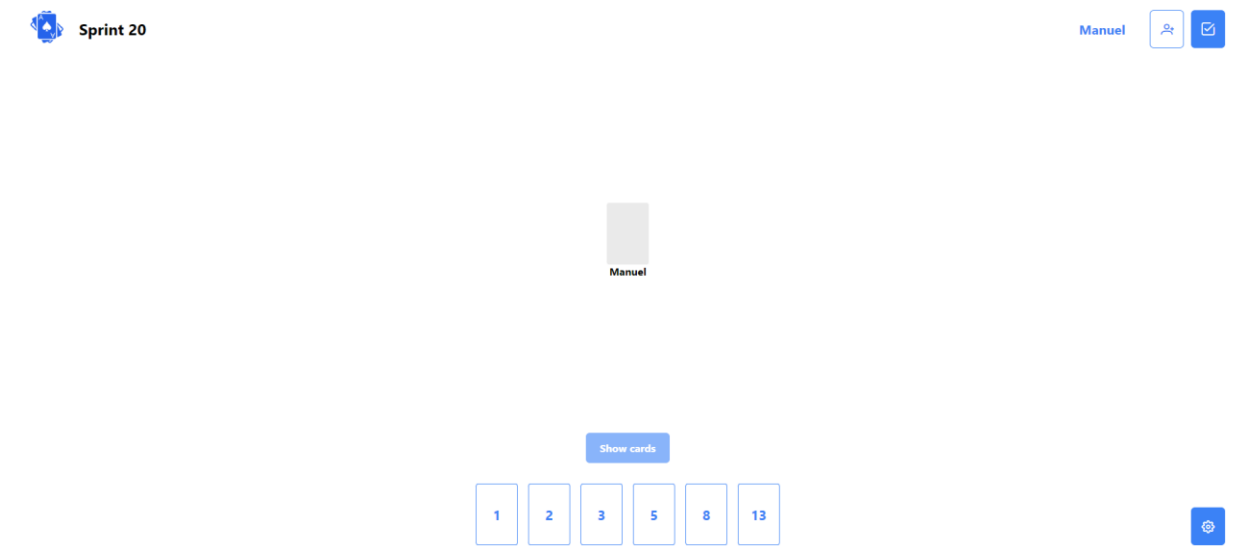
Polja će obavijestiti korisnika ukoliko je unos neispravan. Neispravan unos je prazno polje i korisničko ime koje sadrži više od 9 znakova ili ime sobe koje sadrži više od 10 znakova. Kada korisnik želi napraviti sobu, unosi željeno korisničko ime i u polje za unos iznad gumba *„Create“* unosi ime sobe. Ukoliko je unos ispravan gumb *„Create“* će biti aktivan. Isto vrijedi i za pridruživanje postojećoj sobi samo što korisnik ime sobe unosi iznad gumba *„Join“*. Primjer ispravnog i neispravnog unosa prikazan je na slici 5.2..



Sl. 5.2. Primjer ispravnog i neispravnog unosa

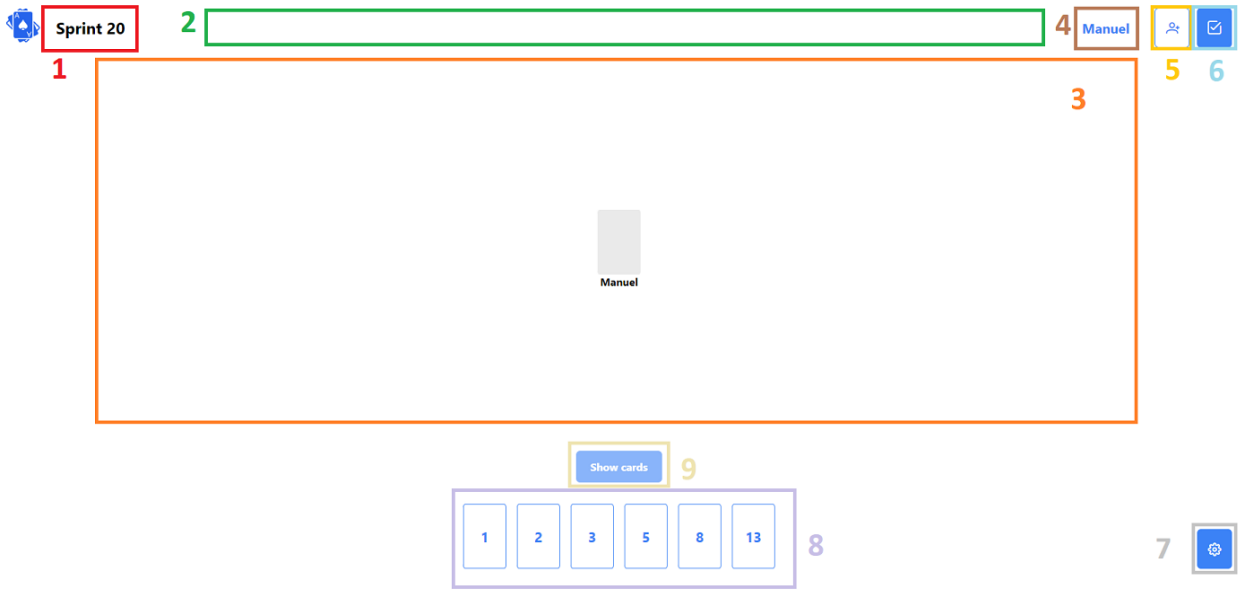
5.2. Virtualna soba

Kada korisnik napravi sobu ili se priključi postojećoj dobiva pristup korisničkom sučelju prikazanom na slici 5.3..



Sl. 5.3. Izgled korisničkog sučelja unutar sobe

Na slici 5.4. brojem su označeni pojedini elementi od kojih se sastoji korisničko sučelje. Svaki pojedini element biti će opisan u nastavku.



Sl. 5.4. Označeni elementi korisničkog sučelja

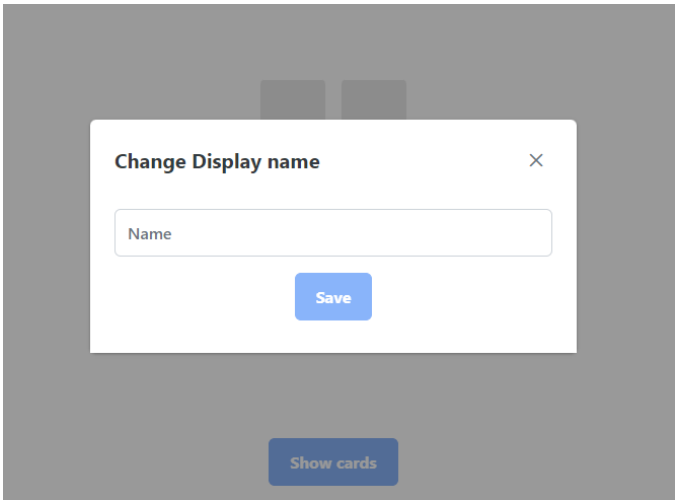
Element označen brojem 1 predstavlja naziv sobe i za njega nije potrebno posebno objašnjenje.

Element označen brojem 2 predstavlja opis trenutno aktivne korisničke priče i vidljiv je svim korisnicima. Ukoliko nema dodanih korisničkih priča ne prikazuje se ništa.

Element označen brojem 3 predstavlja trenutne korisnike u sobi te prikazuje koji od njih je glasao, a koji nije. Na slici 5.5. prikazana su dva korisnika u sobi, Manuel i Ivan. Manuel je glasao, a Ivan nije.

Sl. 5.5. Razlika u karti korisnika koji je glasao i onoga koji nije

Element označen brojem 4 predstavlja gumb s korisničkim imenom. Pritiskom na ovaj gumb otvara se modalni prozor prikazan na slici 5.6..

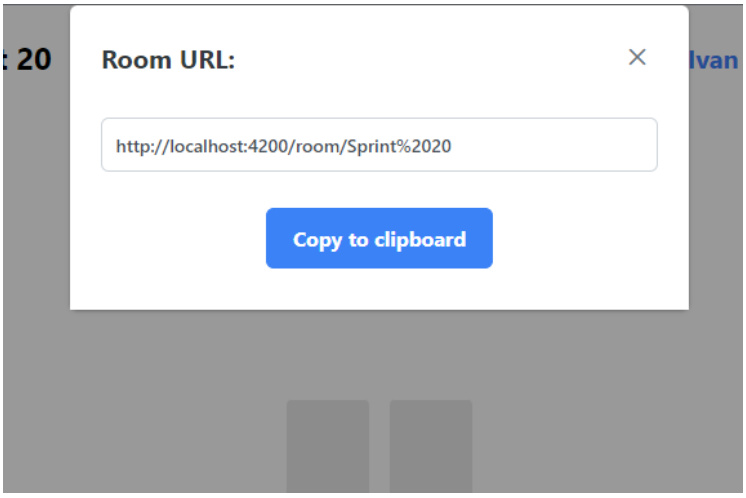


Sl. 5.6. Modalni prozor za promjenu korisničkog imena

Ovaj prozor omogućava korisniku unos novog korisničkog imena te ako je unos ispravan gumb *„Save“* će biti aktivan. Unos je ispravan ako zadovoljava iste kriterije kao i unos za korisničko ime na početnoj stranici. Također se na isti način korisnika obavještava ukoliko je unos neispravan.

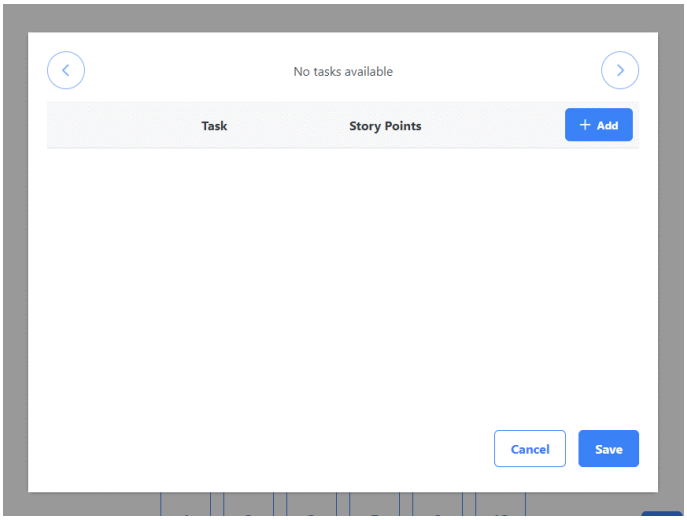
Element označen brojem 5 predstavlja gumb s kojim se otvara modalni prozor koji pomaže pri pozivanju korisnika u sobu (Slika 5.7.). Ovaj modalni prozor prikazuje URL za pristup sobi te

ukoliko korisnik pritisne na gumb *„Copy to clipboard“* korisniku se u međuspremnik pohranjuje URL za pristup sobi koji zatim može prosljeđivati.

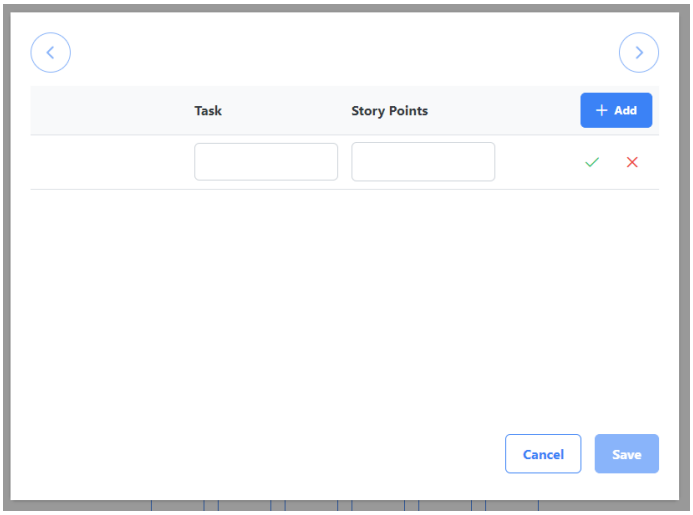


Sl. 5.7. Modalni prozor za kopiranje URL-a za pristup sobi

Element prikazan brojem 6 predstavlja gumb kojim se otvara modalni prozor za upravljanje korisničkim pričama (Slika 5.8.). U njemu se nalazi tablica koja u sebi sadrži sve korisničke priče, te gumb *„ Add“* za dodavanje novih priča. Pritiskom na gumb u tablicu se dodaje novi red s dva polja za unos od kojih lijevo služi za unos opisa korisničke priče, a desno za unos bodova korisničke priče (Slika Sl. 5.9.).

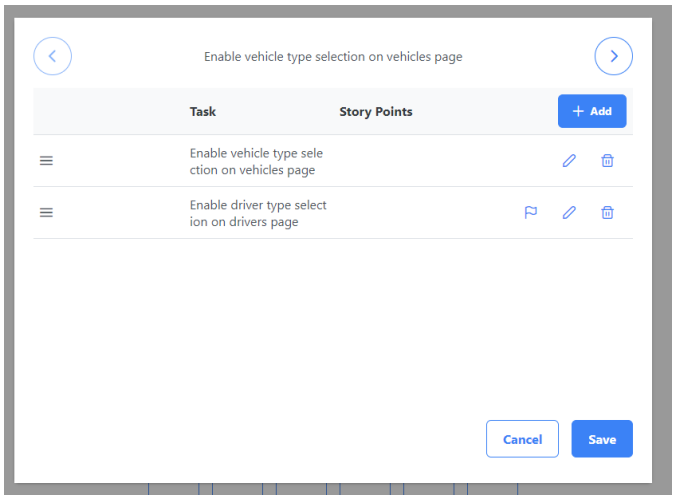


Sl. 5.8. Modalni prozor za rukovanje korisničkim pričama



Sl. 5.9. Dodavanje korisničke priče

Svaka korisnička priča u tablici ima gumb za uređivanje koji korisniku omogućava izmjenu opisa ili bodova priče i gumb za brisanje priče. Također sadrži gumb sa ikonom zastave kojim je moguće postaviti odabranu priču kao trenutno aktivnu priču. Gumbovi na vrhu prozora služe za prebacivanje na prethodnu i slijedeću korisničku priču kako bi korisniku olakšalo promjenu u slučaju velikog broja priča. Redoslijed pojavljivanja korisničkih priči moguće je mijenjati tako da se pritisne na ikonu s lijeve strane, zadrži i povuče na željeno mjesto (Slika 5.10.).



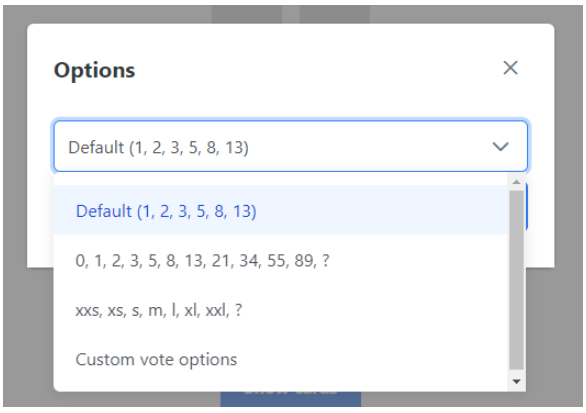
Sl. 5.10. Popis korisničkih priča

Bitno je napomenuti da će se promjene spremiti i biti vidljive ostalim korisnicima tek kada se pritisne dugme *„Save“*. Ukoliko korisnik nije napravio nikakve izmjene, gumb će biti neaktivan. Za zatvaranje prozora i poništavanje izmjena koristi se gumb *„Cancel“*.

Element označen brojem 7 predstavlja gumb kojim se otvara modalni prozor za promjenu ponuđenih karata za glasanje (Slika 5.11.).

Sl. 5.11. Modalni prozor za izmjenu ponuđenih karata za glasanje

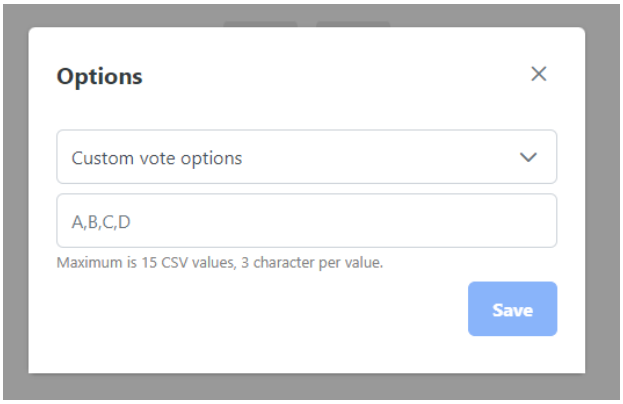
Ovaj prozor sadrži padajući izbornik koji nudi neke predefinirane vrijednosti ili omogućava korisniku unos vlastitih (Slika 5.11.).



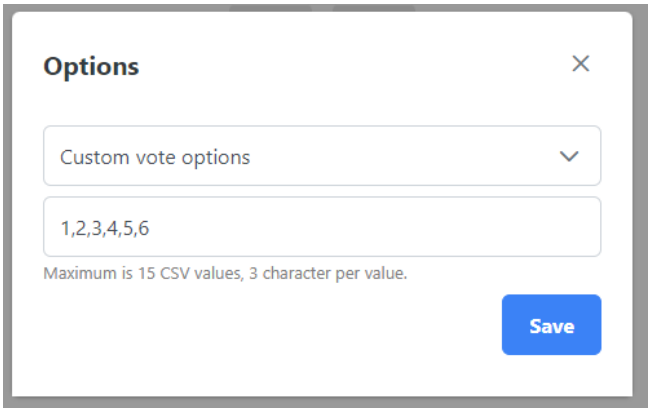
Sl. 5.11. Prikaz padajućeg izbornika

Ukoliko korisnik odabere *„Custom vote options“* ispod padajućeg izbornika pojavljuje se polje za unos vlastitih vrijednosti (Slika 5.12.). Ovo polje obavještava korisnika ako unos nije ispravan. Vrijednosti se unose jedna za drugom s tim da su odvojene zarezom kao što je prikazano na slici 5.13.. Maksimalni broj vrijednosti koji se može unijeti je 15 i svaka se vrijednost može sastojati

od maksimalno 3 znaka. Ukoliko je unos ispravan gumb „*Save*“ će biti aktivan i pritiskom na njega će se svim korisnicima promijeniti ponuđene karte za glasanje.



Sl. 5.12. Prikaz polja za unošenje vlastitih vrijednosti karata

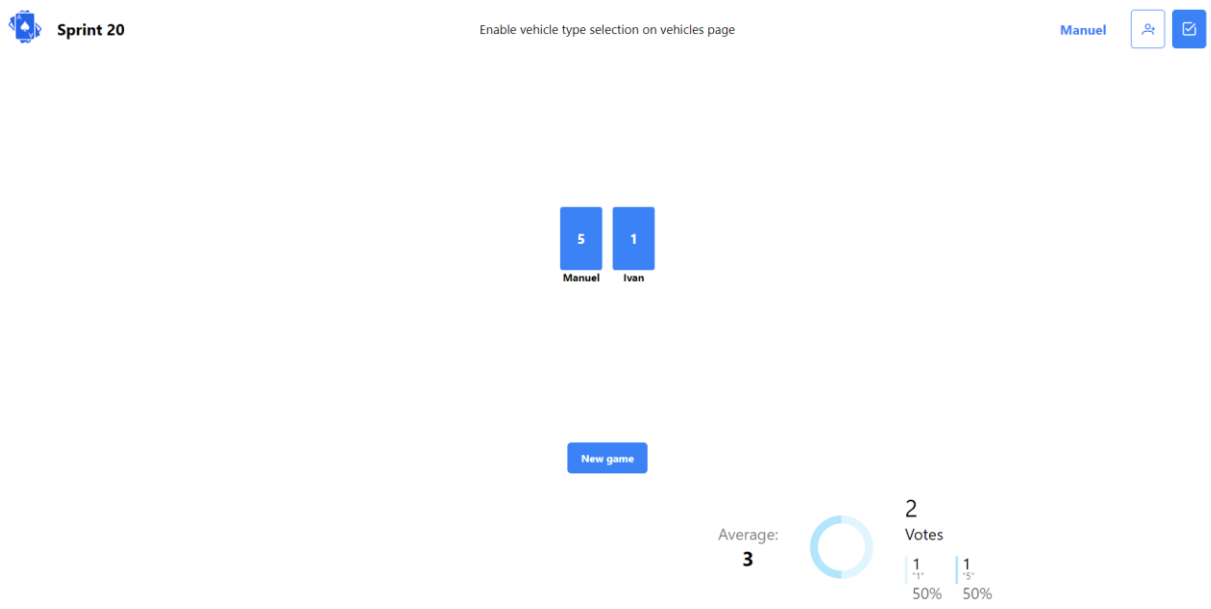


Sl. 5.13. Prikaz ispravnog unosa vrijednosti karata

Element označen brojem 8 predstavlja ponuđene karte koje služe za glasanje. Pritiskom na jednu od karata korisnik je glasao (Slika 5.14.). Isto tako ukoliko korisnik opet pritisne na istu kartu njegovo glasanje se poništava i gleda se kao da nije glasao.

Sl. 5.14. Prikaz karte s vrijednosti „1“ nakon što ju je korisnik odabrao

Element označen brojem 9 predstavlja gumb za prikaz rezultata glasanja. Pritiskom na ovaj gumb skrivaju se karte za glasanje i na njihovom mjestu se pojavljuje grafikon koji prikazuje vrijednosti karata koje su izglasane zajedno s pripadajućim brojem korisnika koji je odabrao pojedinu vrijednosti. Također se prikazuje prosječna vrijednost svih glasova na način da je zaokružena na najbližu ponuđenu. Kartice s imenima korisnika sada pokazuju i pripadajuću vrijednost za koju je taj korisnik glasao. Opisano je prikazano na slici 5.15..



Sl. 5.15. Prikaz rezultata glasanja

Gumb „*Show cards*“ potom je zamijenjen je gumbom „*New game*“ koji započinje novo glasanje.

6. ZAKLJUČAK

Koristeći Angular programsko okruženje realizirana je web aplikacija koja timovima omogućava korištenje tehnike poker planiranja. Time je korisnicima olakšan proces donošenja odluka koje se tiču složenosti određenih poslova te je uklonjena potreba za fizičkim kontaktom i predmetima potrebnim za provođenje postupka odlučivanja temeljenog na tehnici poker planiranja.

Koristeći *Node.js* platformu napravljena je jednostavna poslužiteljska aplikacija s kojom klijentska aplikacija komunicira. Opisan je način komunikacije između poslužiteljske i klijentske aplikacije koristeći biblioteku Socket.IO. Poslužiteljska aplikacija odgovara na zahtjev svakog korisnika i pohranjuje sve informacije vezane za sve postojeće sobe. Time je olakšana distribucija podataka između korisnika. Poslužiteljska aplikacija bi se uvelike mogla poboljšati uvođenjem autentifikacije korisnika i određene arhitekture koja bi omogućila lakšu nadogradnju. Klijentska aplikacija napravljena je pomoću više manjih komponenti od kojih se svaka može ponovno iskoristiti. Implementirana je mogućnost stvaranja virtualne sobe u koju je zatim moguće pozvati ostale korisnike. Potrebni predmeti za postupak odlučivanja spomenutom tehnikom zamijenjeni su jednostavnim i intuitivnim korisničkim sučeljem. Izgled i funkcionalnost aplikacije su prikazani i objašnjeni slikama. Testiranjem aplikacije s timom razvojnih programera utvrđena je njena funkcionalnost kojom su zadovoljene potrebe ovog rada.

LITERATURA

[1] M. N. Adams, »A brief overview of planning poker,«[Mrežno]. Dostupno na: https://www.atlassian.com/blog/platform/a-brief-overview-of-planning-poker. [Pokušaj pristupa 6 Lipanj 2022.].

[2] Anonimno, »Scrum Poker Online,« [Mrežno]. Dostupno na: https://www.scrumpoker-online.org/en/. [Pokušaj pristupa 5 Lipanj 2022.].

[3] Anonimno, »Planning Poker,« [Mrežno]. Dostupno na: https://www.planningpoker.com/. [Pokušaj pristupa 5 Lipanj 2022.].

[4] Anonimno, »We agile you - Poker Planning,« [Mrežno]. Dostupno na: https://planningpokeronline.com/. [Pokušaj pristupa 5 Lipanj 2022.].

[5] Anonimno, »JavaScript,« [Mrežno]. Dostupno na: https://developer.mozilla.org/en-US/docs/Web/JavaScript. [Pokušaj pristupa 5 Lipanj 2022].

[6] Anonimno, »ECMA international,« [Mrežno]. Dostupno na: https://262.ecma-international.org/5.1/. [Pokušaj pristupa 5 Lipanj 2022.].

[7] Anonimno, »TypeScript,« [Mrežno]. Dostupno na: https://www.typescriptlang.org/. [Pokušaj pristupa 5 Lipanj 2022.].

[8] Anonimno, »HTML: HyperText Markup Language,« [Mrežno]. Dostupno na: https://developer.mozilla.org/en-US/docs/Web/HTML. [Pokušaj pristupa 5 Lipanj 2022.].

[9] Anonimno, »CSS: Cascading Style Sheets,« [Mrežno]. Dostupno na: https://developer.mozilla.org/en-US/docs/Web/CSS. [Pokušaj pristupa 5 Lipanj 2022.].

[10] Anonimno, »About W3C,« [Mrežno]. Dostupno na: https://www.w3.org/Consortium/. [Pokušaj pristupa 5 Lipanj 2022.].

[11] Anonimno, »What is Angular?,« [Mrežno]. Dostupno na: https://angular.io/guide/what-is-angular. [Pokušaj pristupa 5 Lipanj 2022.].

[12] Anonimno, »Description,« [Mrežno]. Dostupno na: https://angular.io/api/core/Component. [Pokušaj pristupa 5 Lipanj 2022.].

[13] Anonimno, »Introducing Node,« [Mrežno]. Dostupno na: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction. [Pokušaj pristupa 6 Lipanj 2022.].

[14] Anonimno, »Introducing Express,« [Mrežno]. Dostupno na: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction#introducing_express. [Pokušaj pristupa 6 Lipanj 2022.].

[15] Anonimno, »Introduction,« [Mrežno]. Dostupno na: https://socket.io/docs/v4/#what-socketio-is. [Pokušaj pristupa 6 Lipanj 2022.].

[16] Anonimno, »Introduction,« [Mrežno]. Dostupno na: https://rxjs.dev/guide/overview. [Pokušaj pristupa 10 Lipanj 2022.].

[17] Anonimno, »Introduction,« [Mrežno]. Dostupno na: https://swimlane.gitbook.io/ngx-charts/. [Pokušaj pristupa 10 Lipanj 2022.].

SAŽETAK

Tema ovog diplomskog rada je web aplikacija za primjenu agilne metode estimacije i planiranja koja se temelji na poker planiranju. Cilj rada je omogućiti korisnicima stvaranje virtualne sobe, pozivanje drugih sudionika u sobu te im pružiti korisničko sučelje za međusobnu interakciju. Za potrebe rada napravljene su poslužiteljska i klijentska aplikacija Poslužiteljska aplikacija realizirana je pomoću Node.js platforme, a klijentska pomoću programskog okvira Angular. Za komunikaciju između njih koristi se biblioteka Socket.IO.

Ključne riječi: Angular, NodeJs, poker planiranje, SocketIO, Web aplikacija

ABSTRACT

WEB APPLICATION FOR ESTIMATION AND PLANNING BASED ON POKER PLANNING TECHNIQUE

The topic of this thesis is a web application for the application of agile methods of assessment and planning based on poker planning. The aim of this paper is to enable users to create virtual rooms, invite other participants to the room and provide them with an user interface for interacting with each other. For the needs of the thesis, a server and client application were created. The server application was implemented using the Node.js platform, the client application was implemented using Angular framework. The Socket.IO library is used for communication between them.

Keywords: Angular, NodeJs, poker planning, SocketIO, Web application