

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 215

**SUSTAV ZA PODRŠKU INTERAKTIVNIM APLIKACIJAMA I
IGRAMA NA VIDEOZIDU**

Mihael Rodek

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 215

**SUSTAV ZA PODRŠKU INTERAKTIVNIM APLIKACIJAMA I
IGRAMA NA VIDEOZIDU**

Mihael Rodek

Zagreb, lipanj 2023.

**SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Zagreb, 10. ožujka 2023.

DIPLOMSKI ZADATAK br. 215

Pristupnik: **Mihael Rodek (0036518478)**

Studij: Računarstvo

Profil: Programsко инженерство и информациони системи

Mentor: doc. dr. sc. Tomislav Jaguš

Zadatak: **Sustav za podršku interaktivnim aplikacijama i igrama na videozidu**

Opis zadatka:

Napredak tehnologije i pad cijena velikih ekrana posljednjih godina povećao je dostupnost videozidova u raznim prostorima. Većina ovih uređaja koristi se primarno u informativne ili promidžbene svrhe iako bi ih se, uz odgovarajuću programsku podršku moglo upotrijebiti ih kao ekrane za razne interaktivne aplikacije ili igre. U okviru ovog diplomskog rada potrebno je osmislit i razviti sustav za podršku izvođenja interaktivnih aplikacija i jednostavnim računalnim igrama koje se pokreću na videozidu, a upravljaju preko mobilnog uređaja. Potrebno je proučiti dostupnu literaturu, slična rješenja i potrebne tehnologije te predložiti vlastitu arhitekturu sustava i opis potrebnih programskih sučelja te ostvariti sve dijelove sustava koji će omogućiti korištenje različitih aplikacija i igara na videozidu ili sustavu računala s projektorom. Potrebno je istražiti i osmislit obrazovne mogućnosti sustava, kao i mogućnosti suradnje ili natjecanja više istovremenih korisnika te elemente igrifikacije koji bi se mogli ugraditi u sustav, u cilju povećanja motivacije ili interesa korisnika za navedene aplikacije. U sklopu diplomskog rada potrebno je ostvariti nekoliko testnih aplikacija ili igara kako bi se cijeli sustav mogao isprobati u praksi.

Rok za predaju rada: 23. lipnja 2023.

SADRŽAJ

| | |
|--|-----------|
| 1. Uvod | 1 |
| 2. Pregled rada | 2 |
| 2.1. Pozadina i motivacija | 2 |
| 2.2. Pregled postojećih rješenja | 3 |
| 3. Dizajn sustava i zahtjeva | 8 |
| 3.1. Pregled zahtjeva | 8 |
| 3.1.1. Funkcionalni zahtjevi | 8 |
| 3.1.2. Nefunkcionalni zahtjevi | 9 |
| 3.2. Prijedlog arhitekture sustava | 10 |
| 4. Implementacija programskog rješenja | 13 |
| 4.1. Upravljanje projektom | 13 |
| 4.2. Implementacija komponenata sustava | 21 |
| 4.2.1. Poslužiteljska aplikacija | 21 |
| 4.2.2. Baza podataka i migracije | 29 |
| 4.2.3. Mobilna aplikacija | 32 |
| 4.2.4. Web aplikacija | 44 |
| 5. Edukacijski i igrifikacijski aspekti sustava | 51 |
| 5.1. Elementi igrifikacije | 51 |
| 5.2. Edukacijski elementi | 52 |
| 6. Diskusija i analiza | 54 |
| 6.1. Limitacije i izazovi | 54 |
| 6.2. Poboljšanja i nadogradnje sustava | 55 |
| 7. Zaključak | 56 |
| Literatura | 57 |

1. Uvod

Napredak tehnologije i pad cijena velikih ekrana posljednjih godina povećao je dostupnost videozidova u raznim prostorima. Nekad vrlo premijerni i malobrojni uređaji danas su postali sve dostupniji i rašireniji, a mogućnosti koje oni pružaju sve veće. Trenutačno se ovi uređaji prvenstveno koriste u informativne i promotivne svrhe, no uz odgovarajuću programsku podršku imaju potencijal da se transformiraju u korisnički interaktivna sučelja i sustave.

Međutim, unatoč velikom potencijalu, realizacija takvih interaktivnih videozidova ostaje nedovoljno istražena i iskorištena. Taj primjer izražen je i na Fakultetu Elektrotehnike i Računarstva, gdje postoji sustav videozida, ali njegov potencijal u velikoj mjeri ograničen i neiskorišten. Upravo to stvara priliku za dizajn i razvoj sustava za podršku interaktivnim aplikacijama i igrama na videozidu odakle potiče i sama ideja rada. Cilj ovog rada je transformirati videozidove s tradicionalne, informativne upotrebe te ponuditi dizajn i implementaciju sustava koji će ih pretvoriti u korisnički interaktivne sustave za korištenje.

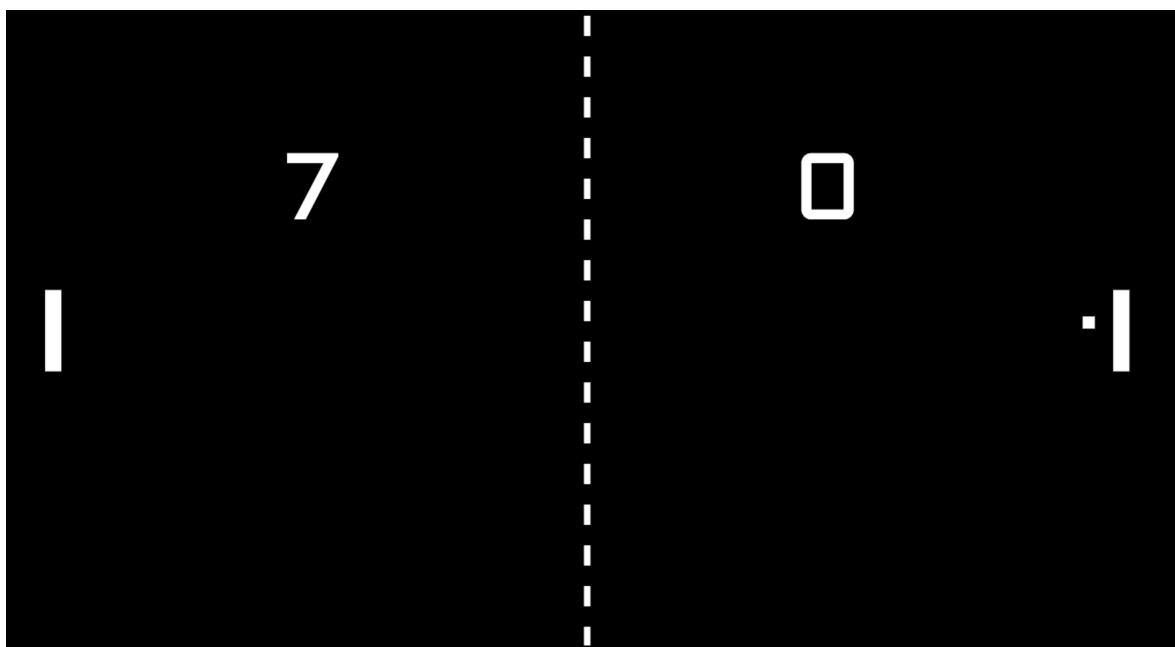
Ovaj rad je podijeljen u nekoliko dijelova, od kojih se svaki fokusira na određeni aspekt samog projekta. Početna poglavlja predstavljaju opsežan pregled rada, opisuju pozadinu i pregled postojećih rješenja, dok se nadalje zadire dublje u sam dizajn i implementaciju samog sustava. Kroz predloženo rješenje nastoji se unijeti inovativna i interaktivna dimenzija sustava koja će korisnike potaknuti na sudjelovanje i pružiti im kvalitetnije interakcije s prikazanim sadržajem.

2. Pregled rada

2.1. Pozadina i motivacija

Pojava i povećana dostupnost videozidova posljednjih godina znatno su proširili njihov potencijal. Ovi nekoć ne tako popularni uređaji danas su postali sveprisutna i pristupačna tehnologija, s mogućnostima daleko izvan konvencionalnih informativnih i promotivnih primjena.

Unatoč njihovom velikom potencijalu, realizacija interaktivnih videozidova ostaje u velikoj mjeri neistražena i nedovoljno iskorištena. Taj primjer vidljiv je i na Fakultetu Elektrotehnike i Računarstva (u dalnjem tekstu FER) na kojem postoji sustav videozida koji nije u potpunosti iskorišten. Trenutačni sustav služi uglavnom kao veliki reklamni zid koji se koristi u obavještajne i promotivne svrhe s velikim neiskorištenim potencijalom. Izuzetak je aplikacija "FER Pong" koju možemo vidjeti na 2.1.



Slika 2.1: Igra FER Pong

Ova aplikacija predstavlja kopiju popularne retro igre istog imena u kojoj dva igrača s reketom odbijaju lopticu sve do kad jedan na temelju bodova ne pobjadi. Iako sama

aplikacija služi svojoj svrsi, problem je u tome što, s obzirom na trenutnu strukturu koda, postaje vrlo teško održiva. U sklopu mentoriranog predmeta Seminar 2 imao sam za cilj unaprijediti trenutnu aplikaciju te proučiti i primijeniti elemente igrifikacije (engl. *gamification*) nad samim sustavom. Proučavanjem aplikacije bilo je jasno da se trenutna aplikacija ne može poboljšati, već se samo trenutni kod može minimalno modificirati te da postoji potreba za stvaranjem zasebnog sustava. Također, u sklopu seminarinskog rada istražene su metode i načini igrifikacije koji se koriste u suvremenim aplikacijama te će se isti naknadno primijeniti u ovom radu.

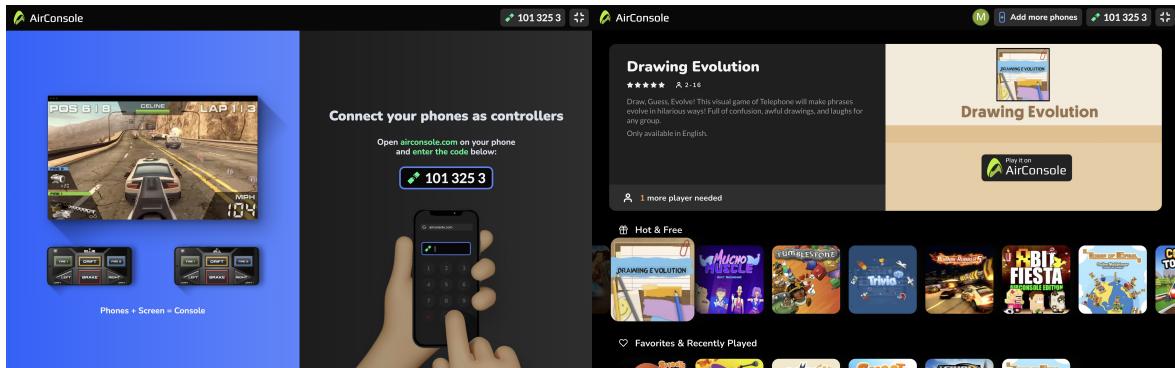
Pošto "FER Pong" predstavlja jedinu interaktivnu aplikaciju koja se pokreće na samome videozidu jasno je da to otvara veliki prostor za dizajn cijelog sustava interaktivnih aplikacija te je upravo iz toga i proizašla motivacija za izradu ovog rada. Cilj sustava je da bude što intuitivniji za korištenje te pruža široku mogućnost više-korisničke interakcije pritom težeći ka poboljšanju slobodnog vremena koje studenti provode na fakultetu uvodeći obrazovne i igrifikacijske elemente, čineći korisničko vrijeme ugodnim, zabavnim i korisnim.

2.2. Pregled postojećih rješenja

Trenutačno na tržištu postoji nekoliko rješenja koja djelomično odgovaraju scenariju videozida. Treba naglasiti da nijedna od tih aplikacija nije implementirana kako bi joj namjera bila isključivo pokretanje na videozidu, već uglavnom kao web aplikacija. Kako bih dobio što bolju inspiraciju za dizajn novog sustava, odlučio sam proučiti postojeća rješenja te načine na koji su oni implementirali određene komponente.

AirConsole

AirConsole[1] je najpoznatija aplikacija i platforma koja omogućuje korisnicima da pretvore svoje pametne uređaje u bežične kontrolere za igranje videoigara na velikom zaslonu, odnosno svojevrsnom videozidu. Sustav je razvijen kako bi omogućio više-korisničko iskustvo igranja, bez potrebe za dodatnim uređajima kao što su igrače konzole ili kontroleri. Jedan od ključnih koncepta AirConsole-a je da korisnici koriste svoje mobilne uređaje kao kontrolere, dok se sama igra prikazuje na većem zaslonu. Sve što je potrebno je da korisnici preuzmu AirConsole aplikaciju na svoje mobilne uređaje i povežu se putem Wi-Fi mreže s istim zaslonom na kojem žele igrati igre. Prikaz web platforme preko koje se korisnik povezuje možemo vidjeti na 2.2, gdje prva slika prikazuje izgled prije prijave, a druga nakon prijave.

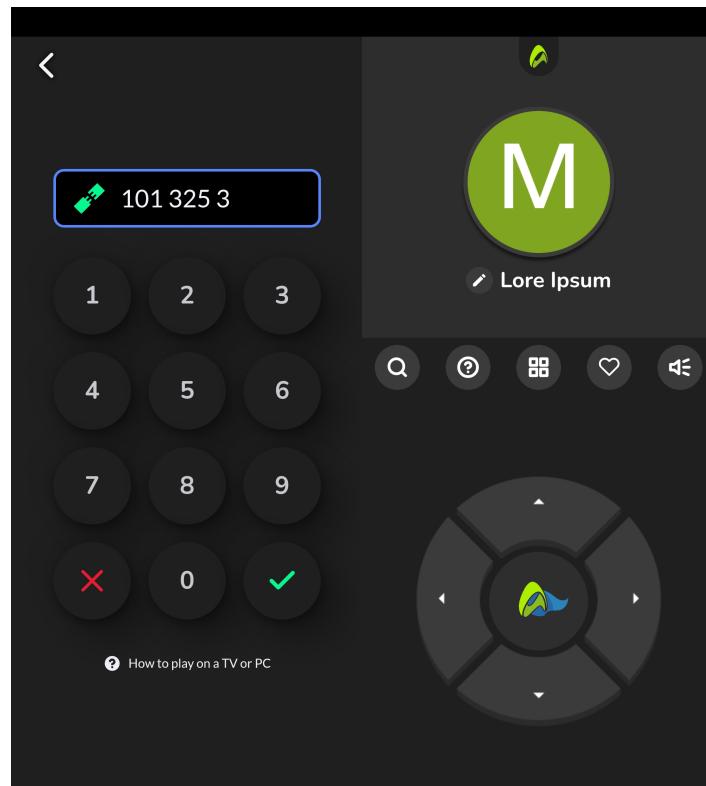


(a) Prije prijave

(b) Poslije prijave - izbornik igara

Slika 2.2: AirConsole Web sučelje

Kako bi korisnik mogao koristiti AirConsole, potrebno je da preuzme aplikaciju na svoj uređaj te zatim unese kod koji je prikazan na zaslonu. Unosom ispravnog koda, korisnik vidi glavno sučelje aplikacije dok se mobilni uređaj automatski pretvara u bežični kontroler za upravljanje igrom kako možemo vidjeti na 2.3.



(a) Prije prijave

(b) Poslije prijave - kontroler

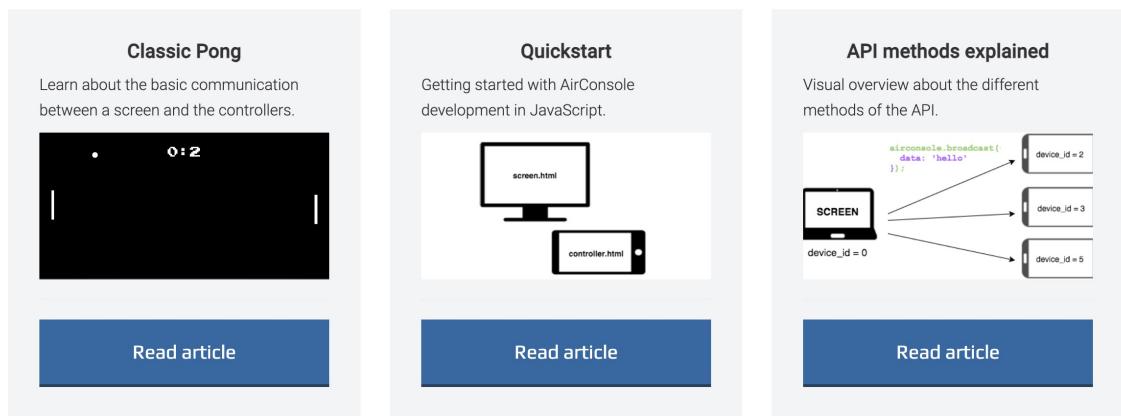
Slika 2.3: AirConsole mobilna aplikacija

Nakon uspješnog povezivanja, korisnici mogu pristupiti raznovrsnom izboru igara koje su dostupne na AirConsole platformi. Te igre su specifično dizajnirane za multiplayer iskustvo i obuhvaćaju različite žanrove kao što su utrke, sportske igre, potezne

strategije, kvizovi i mnoge druge. Igrači koriste svoje mobilne uređaje kao kontrolere, a akcija svih igrača se prikazuje na zajedničkom zaslonu, stvarajući uzbudljivo i interaktivno iskustvo za sve sudionike.

Uz iganje postojećih igara, AirConsole omogućuje i razvijanje vlastitih igra putem AirConsole API-a koji je javno dostupan. API omogućuje korisnicima da kreiraju igru po svojoj želji dajući im na korištenje sve komponente i alate koji bi mogli biti potrebni. Uz to postoje i brojni članci koji objašnjavaju tijek razvoja aplikacije putem API-a, a zanimljivo je da je kao primjer igre za razvoj dan baš Pong. Ovakva pristupačnost stvara AirConsole izrazito modularnim rješenjem za razvoj igara što samo povećava bazu korisnika te broj dostupnih igara na samoj platformi.

Getting started



Slika 2.4: Brojni članci koji objašnjavaju korištenje AirConsole API-a

Jackbox Games Packs

Jackbox Games[2] je razvojni studio specijaliziran za razvoj interaktivnih igara. Tvrta je najpoznatija po svom "The Jackbox Party Pack" serijalu. Svaki paket obično sadrži pet jedinstvenih igara koje se mogu igrati na širokom spektru uređaja, uključujući konzole, računala i mobilne uređaje. Svaki od igrača koristi svoj mobilni uređaj ili tablet kao kontroler, dok se sama igra prikazuje na većem ekranu poput monitora ili televizora. Igre koje razvija Jackbox Games obuhvaćaju širok raspon žanrova od kvizova, crtanja i pisanja do brojnih arkadnih natjecateljskih igara. Kako bi korisnici međusobno mogli igrati igre, potrebno je da barem jedna osoba ima kupljen paket koji se želi igrati, dok se ostale osobe mogu povezati bilo da su na istoj lokaciji ili online putem s udaljene lokacije. Ovo daje veliku prednost u odnosu na AirConsole koji služi isključivo za lokalnu igru. Ipak, u odnosu na AirConsole koji je u potpunosti besplatan, paketi "The Jackbox Party Pack" serijala moraju se zasebno kupovati. Trenutno je izdan paket "Jackbox Party Pack 9" čiji plakat možemo vidjeti na 2.5. Paket je dostupan za

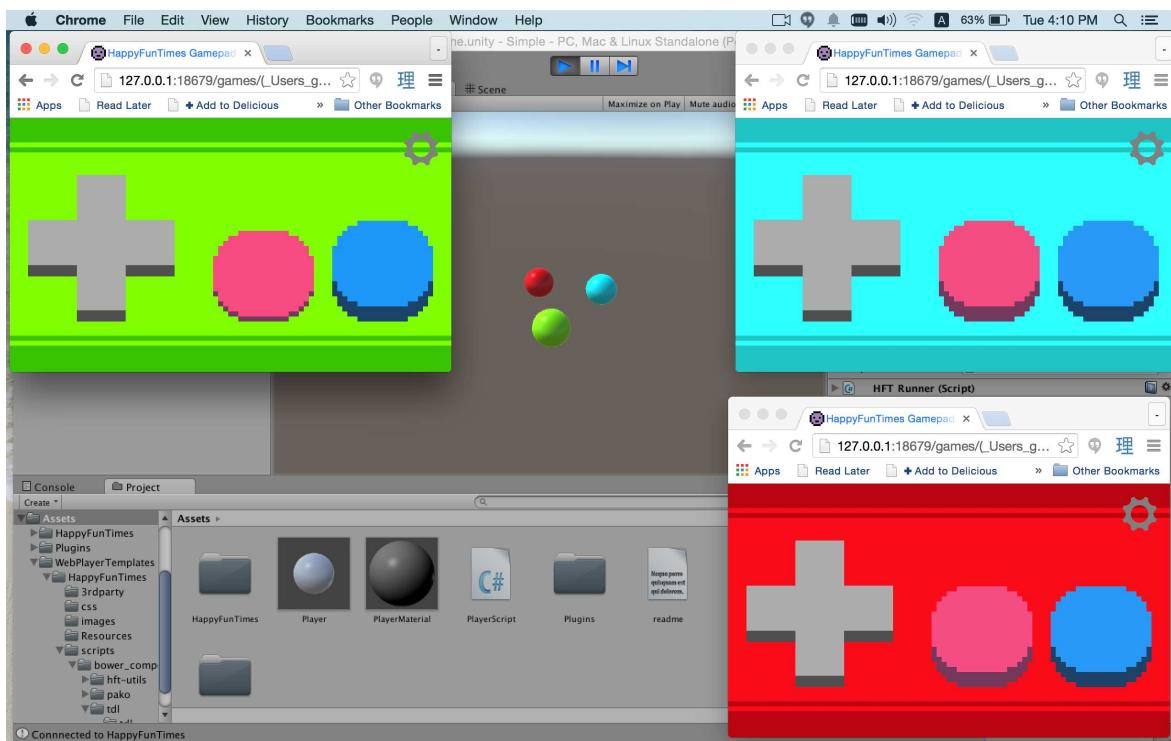
kupnju na svim većim platformama poput Epic Games-a, Playstation Store-a i drugih. S obzirom na to da jedan paket sadrži do pet igara ukupne vrijednosti i preko 30 dolara, dolazi se do zaključka da je takav način igranja poprilično financijski skup posebice ako korisnik želi igrati nekoliko različitih paketa.



Slika 2.5: Paket "Jackbox Party Pack 9" (izvor jackboxgames.com)

HappyFunTimes

HappyFunTimes[3] je bio sustav otvorenog koda (engl. *open source*) za igranje lokalnih višekorisničkih igara koje podržavaju iznimno velik broj igrača. Na određenim igrama ta brojka dosezala je i do 100 igrača koji mogu istovremeno igrati. Sustav je bio iznimno popularan prije desetak godina, no već se par godina ne održava te je zastario. Razvojem modernih tehnologija održavanje i refaktoriranje cijelog sustava je postalo prezahtjevno stoga su autori odlučili ugasiti podršku za sam sustav. Princip igranja je identičan već spomenutim sustavima u kojem se mobilni uređaj koristi kao kontroler, dok web aplikacija predstavlja samu igru. Uz samo igranje, sustav je pružao i detaljnu podršku i dokumentaciju za razvijanje vlastitih igara. Igre su podržavale razvijanje u sustavu Unity ili putem programskog jezika JavaScript. Sljedeća slika 2.6 prikazuje primjer razvijanja igara za HappyFunTimes.



Slika 2.6: Razvijanje igre za HappyFunTimes (izvor happyfuntimes.net)

3. Dizajn sustava i zahtjeva

3.1. Pregled zahtjeva

Dizajniranje svakog složenog sustava započinje definiranjem jasnih i preciznih zahtjeva. Ovi zahtjevi služe kao temelj za razvoj sustava i kao mjerilo za procjenu njegove uspješnosti stoga je izrazito važno da se zahtjevi na sustav pažljivo razmotre i jasno odrede.

U procesu dizajniranja ovog sustava, pažnja je posvećena razumijevanju i artikulaciji svih ključnih funkcionalnih i nefunkcionalnih zahtjeva. Funkcionalni zahtjevi odnose se na konkretnе funkcije koje sustav mora obavljati, dok nefunkcionalni zahtjevi određuju kako se te funkcije unutar sustava moraju izvršavati, često u smislu performansi, sigurnosti, skalabilnosti, pouzdanosti i ostalog.

3.1.1. Funkcionalni zahtjevi

Funkcionalni zahtjevi sustava su sljedeći:

- Svi korisnici mogu pristupiti aplikaciji
- Sustav mora komunicirati u stvarnom vremenu putem WebSocketa
- Sustav mora pružiti korisnicima mogućnost izbora između svijetlog (light) i tamnog (dark) moda korisničkog sučelja
- Korisnik se može ulogirati u sustav
- Korisnik treba moći prilagoditi postavke aplikacije prema svojim preferencijama
- Korisnik se može kretati navigacijskom trakom
- Korisnik se može pridružiti sobi unoseći broj sobe
- Korisnik se može pridružiti sobi skenirajući QR kod sobe
- Korisnik može navigirati različitim ekranima i dijelovima sustava
- Postoji sustav nagrađivanja putem novčića
- Korisnik može pomoći kontrolera upravljati određenom igrom

- Za svaku igru dostupna je statistika i opis igre
- Korisnik može zaraditi novčiće igranjem kviza unutar mobilne aplikacije
- Korisnik treba moći birati različite teme kviza
- Sustav treba omogućiti izbor različitih jezika sučelja
- Korisnik može pregledavati vlastiti profil
- Korisnik može izmijeniti profilnu sliku
- Korisnik može vidjeti listu svojih najboljih rezultata
- Korisnik treba moći prekinuti i ponovno pokrenuti igru

3.1.2. Nefunkcionalni zahtjevi

Nefunkcionalni zahtjevi sustava su sljedeći:

- Sustav treba biti intuitivan i jednostavan za korištenje
- Neispravno korištenje sučelja ne smije ni na koji način obustaviti rad sustava
- Sustav mora biti u stanju brzo obraditi dobivene podatke i isporučiti ih korisniku
- Sustav mora biti skalabilan s obzirom na različite verzije svih komponenata sustava
- Sustav mora podržavati dijakritičke znakove
- Sustav mora podržavati različite načine lokalizacije
- Baza podataka mora biti zaštićena i otporna na pogrešne zahtjeve
- Sustav mora podržavati sigurnu komunikaciju između komponenata kako bi se osigurala privatnost korisnika
- Sustav mora biti dizajniran da minimizira utjecaj na performanse uređaja na kojem se izvodi
- Sustav treba pružiti brzu povratnu informaciju korisnicima na njihove akcije
- Sustav bi trebao biti dizajniran da se može lako održavati i nadograditi
- Sustav mora biti kompatibilan s različitim vrstama uređaja i platformi sljedeći preporučene standarde
- Prilikom skeniranja pogrešnog QR koda sustav ga treba zanemariti

3.2. Prijedlog arhitekture sustava

Predložena arhitektura sustava strukturirana je oko već spomenutih funkcionalnih i nefunkcionalnih zahtjeva, s ciljem pružanja praktičnog i održivog rješenja. Prilikom odabira dizajna arhitekture sustava u obzir su uzete pozitivne i negativne strane već postojećih proučenih sličnih rješenja koja su opisana u poglavlju ranije, a koja su smislena za implementaciju same aplikacije. Ovo poglavlje detaljnije opisuje pojedine komponente koje je potrebno implementirati te služi kao prijedlog rješenja i alata za implementaciju, dok će odabir konkretnе tehnologije za implementaciju biti u sljedećem poglavlju.

Kao središnja komponenta ove arhitekture zamišljena je web aplikacija koja funkcioniра po principu videozida. Ova aplikacija treba biti vizualno privlačna i korisnički intuitivna za korištenje, olakšavajući prezentaciju različitih oblika sadržaja. Njezina namjena je isključivo olakšati korisniku interakciju sa sustavom te reagirati na potrebne unose od strane korisnika. Kao alat za izgradnju same web aplikacije predlaže se koristiti jedan od popularnih modernih okvira za izgradnju korisničkih sučelja kao što su Reacta, Angulara i Vue.jsa.

Uz web aplikaciju potrebno je implementirati i mobilnu aplikaciju. Svrha mobilne aplikacije je da djeluje kao primarno korisničko sučelje pružajući kontrolu nad funkcijama sustava te samu interakciju s videozidom. Mobilnom aplikacijom pojedini korisnik se može ulogirati u sustav, povezati s trenutnim videozidom te upravljati videozidom i ostalim potrebnim funkcionalnostima. Kao okvir za izradu mobilnih aplikacija predlaže se jedan od popularnih nativnih okvira i jezika poput React Native i Fluttera ili okvir namijenjen pojedinoj platformi poput Kotlin i Java (operacijski sustav Android) i Swifta (operacijski sustav iOS).

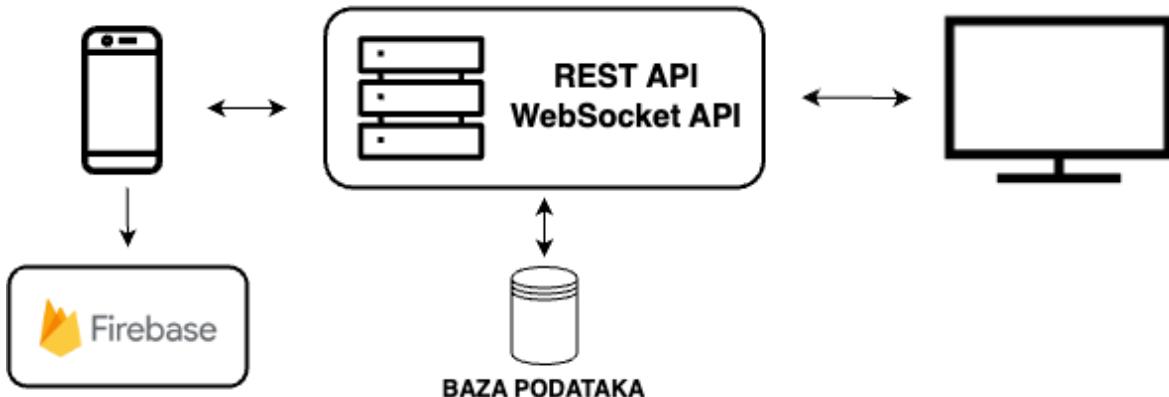
Kako bi se mogla ostvariti komunikacija između web i mobilne aplikacije, potrebno je razviti poslužiteljski servis (engl. *backend*). Cilj poslužiteljskog servisa je djelovati kao posrednik između web i mobilne aplikacije, primajući naredbe od jedne, obrađujući ih te prenoseći na drugu aplikaciju. Pomoću servisa je potrebno izložiti REST pozive za operacije potrebne nad određenim podatcima te WebSocket protokol za komunikaciju u realnom vremenu. Nekoliko jezika koji se predlažu za implementaciju poslužiteljskog servisa su Java, Go i .NET koji zbog svoje popularnosti i gotovih programskih knjižica (engl. *library*) korisniku olakšavaju razvoj servisa.

U sustav je također potrebno uključiti i prikladnu bazu podataka za pohranu relevantnih podataka. Namjena baze je da sadrži informacije o korisničkim profilima te ostalim potrebnim elementima koji su potrebni kako bi sama aplikacija funkcionirala. Baze koje se predlažu za korištenje u sklopu sustava su PostgreSQL, MySQL, AzureSQL i slično. U sklopu određene sigurnosti u sustav je potrebno integrirati i servis za auten-

tifikaciju korisnika. Umjesto implementacije prilagođenog servisa za autentifikaciju, zbog korisničke sigurnosti i olakšavanja razvoja ideja je koristiti jedan od već postojećih servisa poput Keycloak, Firebasea ili AWS Cognitoa. Ti servisi nude robustne sigurnosne značajke i široko se koriste, smanjujući vrijeme razvoja i potencijalne sigurnosne ranjivosti. Servis za autentifikaciju je primarno potreban za mobilnu aplikaciju, budući da web aplikacija i poslužiteljski servis ne zahtijevaju dodatnu autentifikaciju korisnika u trenutačnom rješenju.

Ova arhitektura predstavlja sveobuhvatan i fleksibilan pristup razvoju sustava koji zadovoljava definirane ciljeve. Svaka komponenta ima ključnu ulogu u osiguranju operativnosti sustava, pružajući brzo i responzivno rješenje koje se može prilagoditi i nadograditi prema potrebi. Pošto će istovremeno aplikaciju koristiti svega nekoliko korisnika, njezina skalabilnost nije pretjerano bitna, no brzina i responzivnost su ključne. Korisnički unos treba biti obradiv i prikazan korisniku na vjerni i pouzdan način, bez zastoja. Ovaj aspekt operativnosti je posebno važan za održavanje korisničke interakcije i zadovoljstva. Korištenjem modernih tehnologija i najboljih praksa dizajna aplikacija treba omogućavati visoku razinu performansi i responzivnosti.

Upravo to je jedna od glavnih smjernica kojima se treba voditi prilikom odabira tehnologija za cijeli sustav. Dijagram predložene arhitekture sustava koji potkrjepljuje gore navedene zahtjeve i veze unutar sustava prikazan je na 3.1.



Slika 3.1: Prijedlog arhitekture sustava

Vodeći se definiranim zahtjevima projekta, suvremenim tehnološkim trendovima i osobnim preferencijama, odabrane su odgovarajuće tehnologije i okviri za izgradnju različitih komponenti sustava. Ključni kriteriji koji su utjecali na odabir uključuju performanse, responzivnost i lakoću integracije s ostatkom komponenti sustava.

Za implementaciju poslužiteljskog servisa odabran je programski jezik Go. Go je stvoren s ciljem postizanja iznimnih performansi i efikasnosti, što ga čini optimalnim

za izgradnju backend servisa koji treba biti brz i responzivan. Za izradu web aplikacije, odabran je React, biblioteka dizajnirana za efikasnu izgradnju korisničkih sučelja. React pruža mogućnost brze izrade dinamičkih korisničkih sučelja, omogućujući stvaranje impresivnih korisničkih iskustava. Kotlin, kao moderni programski jezik posebno je dizajniran za razvoj Android aplikacija, a pruža mogućnost brzog i jednostavnog razvoja mobilne aplikacije koja ispunjava sve postavljene zahtjeve. Za pohranu podataka koristit će se AzureSQL, baza podataka koja podržava relacijski model podataka. AzureSQL nudi visoke performanse i sigurnost, a u isto vrijeme pruža fleksibilnost i jednostavnost korištenja. Kada je u pitanju autentifikacija korisnika, odabran je Firebase. Firebase nudi širok spektar značajki za razvoj, uključujući robustan sustav za autentifikaciju. Ovaj skup tehnologija omogućit će izgradnju responzivnog i učinkovitog sustava, spremnog da zadovolji sve postavljene zahtjeve, dok u isto vrijeme daje fleksibilnost za buduća proširenja i nadogradnje.

4. Implementacija programskog rješenja

Na planiran tijek razvoja aplikacije i njezinih komponenti utjecalo je više faktora, između kojih su najveći utjecaj imala već postojeća rješenja te zadani funkcionalni i nefunkcionalni zahtjevi sustava. Postojeća gotova rješenja poput Fer Pong ili AirConsole sustava dala su najbolji mogući uvid u samu ideju i način podjele različitih komponenata te je implementacija najvećim djelom bila zasnivana na njima.

Sama ideja aplikacije je implementirati videozid na kojem će se prikazivati igre kontrolirane pomoću mobilne aplikacije koja služi kao kontroler. Mobilna aplikacija je jedna od bitnijih komponenata sustava jer služi za navigiranje ekranima, pridruživanje sobama te kontroliranje igara. Sve te akcije istovremeno se prikazuju korisniku na videozidu. Korisnik može igrati određenu igru, no za to mu je potrebno da ima dovoljan broj novčića. Novčići se sakupljaju igranjem kvizova pitalica s ponuđenim odgovorima unutar mobilne aplikacije. Detaljniji opis pojedinih komponenata sa implementacijske i funkcionalne strane bit će opisan u sljedećim poglavljima.

4.1. Upravljanje projektom

U procesu izrade ovog sustava, ključnu ulogu igrali su razni alati i metode upravljanja projektom. Oni su olakšali razvoj i organizaciju samog sustava te bi bez njih, upravljanje projektom bilo znatno teže. Ovo poglavlje će se ukratko dotaknuti pojedinih alata i njihovog utjecaja kod razvoja sustava.

Gitlab Issues

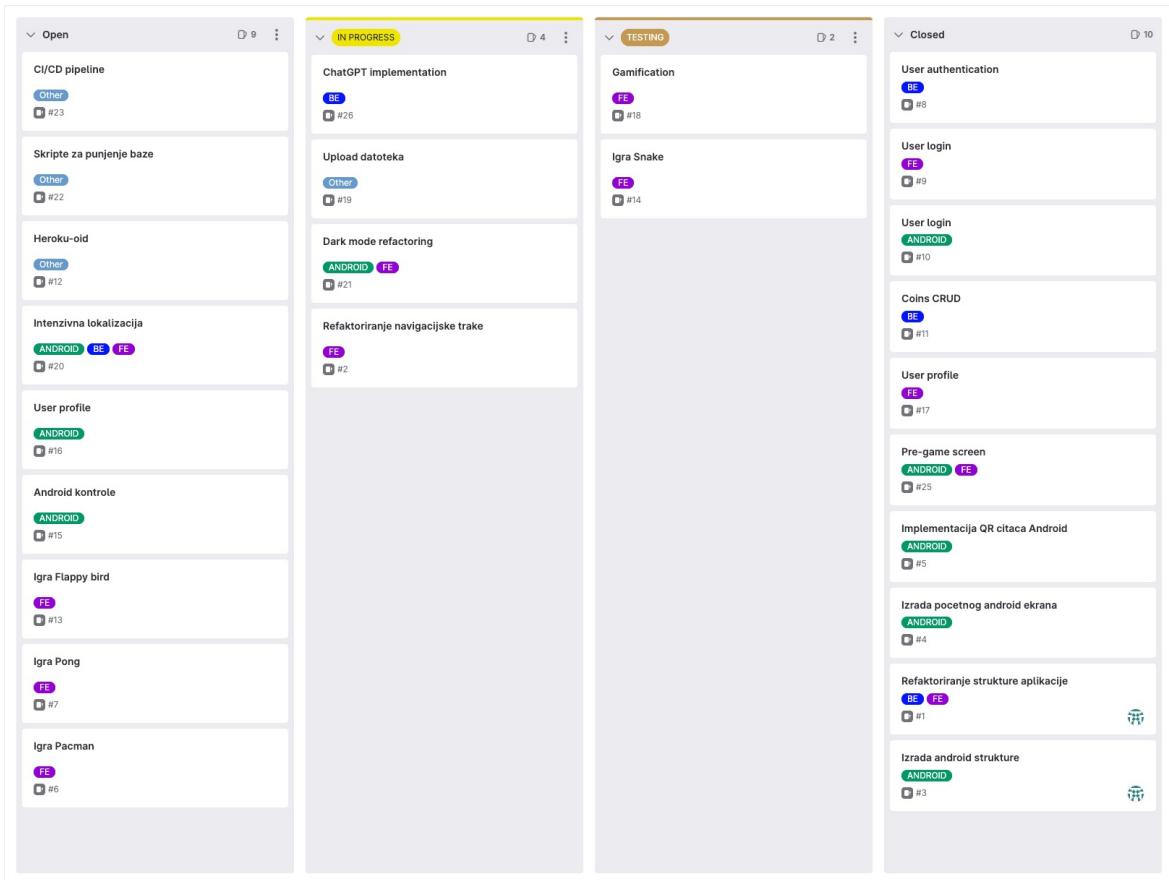
Git je najpoznatiji sustav za upravljanje verzijama koji omogućuje praćenje promjena u datotekama. Ovaj alat je posebno koristan za razvoj softvera, gdje više ljudi često radi na istom projektu. GitLab, platforma koja koristi Git, pruža brojne značajke koje dodatno olakšavaju razvoj i upravljanje softverom. Jedna od tih značajki je i "GitLab Issues". GitLab Issues omogućuje timovima da prate, organiziraju i prioritiziraju rad

na projektu. Svaki zadatak ili problem se može stvoriti kao "issue", odnosno problem, koji se može dodijeliti određenom članu tima, kategorizirati, i pratiti dok se isti ne riješi. Iako je u okviru ovog projekta samo jedna osoba radila na samom sustavu, kategoriziranje problema i raspisivanje zadataka svejedno je uvelike pomoglo prilikom samog razvijanja.

Nakon što su raspisani osnovni zadatci, svakom od zadataka dodijeljena je određena labela ovisno o skupini kojoj zadatak pripada. Labele korištene kod kategorizacije su sljedeće:

1. **BE** - backend, poslužiteljska aplikacija
2. **FE** - frontend, web aplikacija
3. **ANDROID** - mobilna aplikacija
4. **OTHER** - ostalo

Tijek izrade svih zadataka može se pratiti pomoću ploče (engl. *board*) na kojoj su prikazani svi zadaci i njihovi statusi. Status zadataka označava u kojoj je fazi izrade. Na samome početku svi zadaci se vode kao otvoreni (engl. *open*), dok se zatim njihov tok može mjenjati kroz faze u izradi (engl. *in progress*), testiranje (engl. *testing*) te konačno zatvoren (engl. *closed*). Na sljedećoj slici 4.1 moguće je vidjeti ploču zadataka u sklopu izrade ovog rade u fazi izrade projekta gdje su različiti zadaci označeni različitim rednim brojevima i labelama te se nalaze u različitim statusima.



Slika 4.1: GitLab ploča zadataka

Izrazito važna stvar prilikom korištenja Gita je korištenje grana (engl. *branch*). Grane omogućuju programerima da izolirano rade na novim značajkama ili ispravcima bez utjecaja na glavni, stabilni kod koji se često naziva "master" ili "main" grana. Kada se stvori nova grana, stvara se novi niz commitova koji počinje od trenutne točke u kodu i omogućuje paralelni razvoj. Jednom, kada su gotove promjene na određenoj grani, moguće je tu granu ponovno spojiti u glavnu granu. Kako se ne bi koristilo proizvoljno imenovanje grana, pri imenovanju grana sam koristio konvenciju za imenovanje. Tako je svaka grana bila u obliku "**X-ime-featura**", gdje X predstavlja broj zadatka na koji se grana odnosi, a ime kratak proizvoljan opis. Na taj se način svaki puta prilikom spajanja u glavnu granu ta grana povezala s određenim zadatkom te se taj zadatak automatski prebacio u fazu završenog.

Docker compose

Docker je platforma otvorenog koda koja automatizira deployment, skaliranje i izolaciju aplikacija kroz upotrebu kontejnera (engl. *container*). Kontejner je standardizirana jedinica softvera koja omogućuje programerima da izoliraju svoju aplikaciju od okruženja u kojem se izvodi neovisno o operacijskom sustavu. To je iznimno korisno kada se apli-

kacija mora premjestiti s jednog okruženja na drugo - npr. s razvojnog okruženja na testno ili produkcijsko okruženje. Uz to, kontejneri olakšavaju i ubrzavaju podizanje servisa, posebice kod lokalnog postavljanja sustava. Iz tog su razloga Docker kontejneri postali popularan alat za razvoj, deployment i skaliranje današnjih aplikacija.

Docker Compose je alat za definiranje i upravljanje više-kontejnerskim Docker aplikacijama. Omogućuje korisnicima da koriste YAML datoteke za konfiguraciju aplikacijskih servisa i s jednostavnim naredbama možete stvoriti i pokrenuti sve servise iz određene konfiguracije. U ovom projektu, Docker Compose je korišten za pokretanje baze podataka. Docker compose datoteka definira *MSSQL* servis koristi Azure SQL Edge sliku i konfiguriran je za pokretanje Microsoft SQL Server baze podataka kako je vidljivo na sljedećoj slici 4.2.

```
version: '3.9'

services:
  mssql:
    image: mcr.microsoft.com/azure-sql-edge:latest
    environment:
      MSSQL_DB: "videowall"
      MSSQL_SA_PASSWORD: "MyPass@word"
      ACCEPT_EULA: "Y"
    ports:
      - "1433:1433"
```

Slika 4.2: Docker compose

Makefile

Makefile je datoteka koja sadrži niz pravila za automatizaciju procesa izgradnje (engl. *build*), testiranja, čišćenja i drugih zadatka u razvoju softvera. Koristi se za olakšavanje ponovljivih postupaka i upravljanje ovisnosti između različitih dijelova projekta. Makefile sadrži ciljeve (engl. *targets*) i pravila (engl. *rules*). Ciljevi predstavljaju zadatke koji se žele izvršiti, poput izgradnje aplikacije ili pokretanja testova, dok pravila definiraju kako se ti ciljevi ostvaruju. Pravila specificiraju akcije koje se trebaju izvršiti i ovisnosti između ciljeva i izvornih datoteka.

U konkretnom su sustavu korištene dvije Makefile datoteke. Skripta za poslužiteljski

servis[4] ima sljedeće ciljeve:

1. **build** - kompajlira poslužiteljsku aplikaciju za Linux operacijski sustav u direktoriju cmd i koristi vrijednosti verzije definirane u datoteci VERSION
2. **build_local** - kompajlira poslužiteljsku aplikaciju za lokalno izvođenje bez posebnih platformskih ovisnosti
3. **run** - pokreće poslužiteljsku aplikaciju u razvojnog okruženju s određenim opcijama
4. **mod** - preuzima i provjerava sve ovisnosti definirane u go.mod datoteci
5. **clean** - briše generirane izvršne datoteke backend aplikacije
6. **test** - izvršava testove za backend aplikaciju
7. **swagger** - generira Swagger dokumentaciju na temelju definicija u kodu

Skripta za bazu podataka ima sljedeće ciljeve:

1. **build** - kompajlira poslužiteljsku aplikaciju za Linux operacijski sustav u direktoriju cmd i koristi vrijednosti verzije definirane u datoteci VERSION
2. **build_local** - kompajlira poslužiteljsku aplikaciju za lokalno izvođenje bez posebnih platformskih ovisnosti
3. **run_up** - pokreće skriptu za podizanje baze podataka
4. **run_down** - pokreće skriptu za zaustavljanje baze podataka
5. **mod** - preuzima i provjerava sve ovisnosti definirane u go.mod datoteci
6. **clean** - briše generirane izvršne datoteke backend aplikacije

Kako bi se koristili pojedini dijelovi skripte, potrebno je pozicionirati se na ispravno mjesto u terminalu te pozvati jednu od naredba kao naprimjer *make run_up* za podizanje baze podataka ili *make run* za pokretanje poslužiteljskog servisa. Sljedeća slika 4.3 prikazuje kako jedna konkretna Makefile datoteka upravlja poslužiteljskim servisom.

```

.PHONY: build build_local run mod clean test swagger

CURRENT_DIR=$(shell pwd)
SERVICE_NAME=service
VERSION=$(shell cat ${CURRENT_DIR}/VERSION)

build: clean mod
    GOOS=linux GOARCH=amd64 go build -ldflags="-w -s -X 'main.version=$(VERSION)'" -o $(SERVICE_NAME) ./cmd

build_local: clean mod
    go build -ldflags="-w -s -X 'main.version=$(VERSION)'" -o $(SERVICE_NAME) ./cmd

run:
    FW_ENV=dev go run ./cmd --verboseDB=true --config=./data/config.yaml

mod:
    go mod download && go mod verify

clean:
    rm -rf $(SERVICE_NAME)

test:
    go test ./...

swagger:
    go generate ./tools.go

```

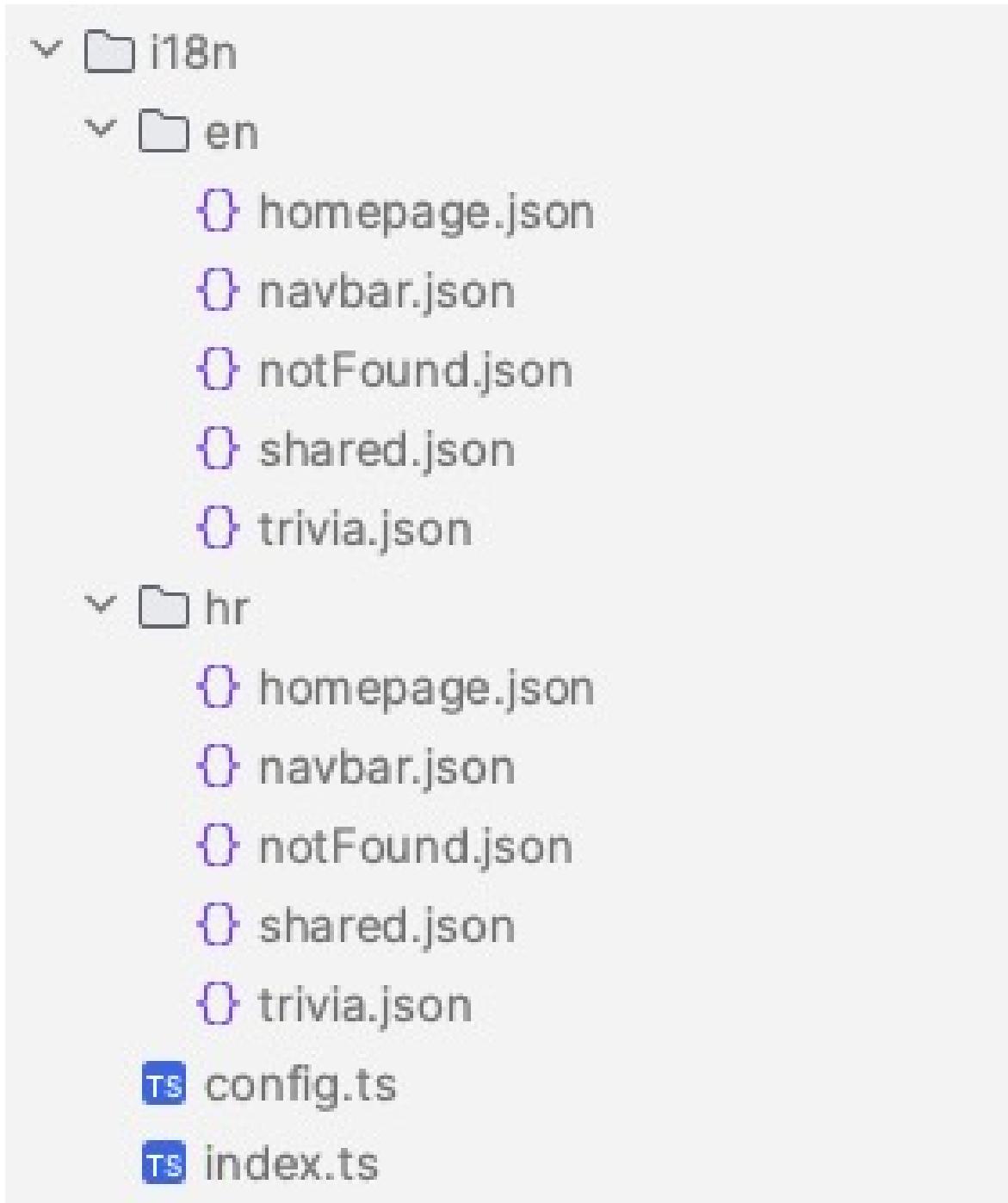
Slika 4.3: Makefile datoteka

Lokalizacija

Lokalizacija je proces prilagodbe softverske aplikacije za specifično geografsko ili kulturno-geografsko tržište. Ovaj proces uključuje prilagodbu softvera tako da odgovara jezičnim, kulturnim i drugim zahtjevima određene ciljane publike. Lokalizacija je posebno važna za aplikacije koje su namijenjene međunarodnoj uporabi jer pomaže korisnicima da bolje razumiju i koriste aplikaciju na način koji je najprirodniji za njih.

U kontekstu ovog projekta, lokalizacija je korištena kako bi se poboljšalo korisničko iskustvo pružajući poruke i sučelje na jeziku koji korisnik najbolje razumije. S poslužiteljske strane ovdje je riječ o prikazivanju poruka, a sa strane videozid aplikacije prikaz samog korisničkog sučelja. To nam omogućuje da se ne koriste hardkodirani dijelovi unutar koda, već su oni izdvojeni u posebne datoteke ovisno o jeziku na kojem se poruka želi prikazati. Za ove potrebe korišten je alat i18n, odnosno biblioteka *i18next*[5] na frontendu i biblioteka otvorenog koda *go-i18n*[6] na backendu.

Na sljedećoj slici može se vidjeti primjer kako je strukturirana lokalizacija sa strane web aplikacije u određenom djelu razvijanja projekta. Za svaki željeni jezik potrebno je stvoriti novi direktorij s imenom jezika, te unutar njega odgovarajuće json datoteke za pravilno prevođenje kako je vidljivo na slici 4.4.



Slika 4.4: Struktura lokalizacije

Na sljedećoj slici 4.5 prikazana je *shared.json* datoteka i način na koji se stvara lokalizacija na engleskom jeziku.

```
{
  "forms": {
    "backButton": "Back",
    "save": "Save",
    "create": "Create"
  },
  "tables": {
    "ordinalNumber": "Number",
    "additionalOptions": "Additional options",
    "cancelSelection": "Cancel"
  },
  "tablePagination": {
    "page": "Page",
    "perPage": "Per page:",
    "total": "Total:",
    "selected": "Selected:"
  },
  "filters": {
    "hideFilters": "Hide filters",
    "showFilters": "Show filters",
    "reset": "Reset",
    "apply": "Apply"
  }
}
```

Slika 4.5: Datoteka shared.json

Ako sad korisnik želi koristiti određenu lokalizacijsku poruku, može joj pristupiti kako je prikazano na sljedećem isječku koda.

```

const { t } = useTranslation('shared');

...
<SubmitInput value={entityExists ? t('forms.save') : t('forms.create')} />
...

```

Sličan stvar vrijedi i kod poslužiteljskog djela aplikacije, samo je način instanciranja i pristupanja lokalizacijskim vrijednostima malo drugačiji.

4.2. Implementacija komponenata sustava

Sustav se sastoji od nekoliko ključnih komponenti čiji arhitekturalni detalji su temeljito opisani u poglavlju "Prijedlog arhitekture sustava". Svaka od ovih komponenata ima svoju specifičnu ulogu i funkcionalnost koja doprinosi cjelokupnom radu sustava. U ovom poglavlju, svaka komponenta će biti detaljno opisana s naglaskom na njene ključne funkcionalnosti, strukturu aplikacije i koda. Opisivanje svake komponente će dati dublji uvid u unutarnji rad i funkcionalnosti sustava. Detaljni opisi će obuhvatiti arhitekturne odluke, obrasce uporabe, i tehnologije koje su korištene u implementaciji svake komponente. Također će se raspravljati o integraciji i međusobnoj interakciji komponenti kako bi se stvorio kohezivan i dobro funkcionirajući sustav. Dodatno, bit će navedeni primjeri koda i pseudokoda kako bi se bolje ilustrirali koncepti i logika implementacije svake komponente.

4.2.1. Poslužiteljska aplikacija

Programski jezik Go[7], poznat i kao Golang, predstavlja jedan od modernih programskih jezika koji se ističe svojim jednostavnim i čistim dizajnom, a izrađen je s naglaskom na efikasnost i performanse. Stvoren je unutar Google-a, s jasnim ciljem adresiranja nekih od problema koji su bili prisutni u drugim jezicima, posebno u pogledu konkurentnog programiranja, jednostavnosti upotrebe i brzine izvršavanja. Go se sve više prepoznaće kao iznimno koristan alat za izgradnju skalabilnih, visoko performansnih poslužiteljskih servisa. Njegova učinkovitost dolazi od optimiziranog upravljanja memorijom i podrške za konkurentno programiranje pomoću rutina (engl. *goroutine*), što omogućava efikasno procesiranje višestrukih zahtjeva istovremeno. Go je također dizajniran da bude jednostavan za učenje i upotrebu, s minimalističkom sintaksom i jasno definiranim pravilima, a u isto vrijeme nudi statičko tipiziranje i brzo vrijeme komplamacije, čime se smanjuje mogućnost pogrešaka i povećava produktivnost programera.

Go ekosustav također nudi veliki broj razvijenih biblioteka otvorenog koda (engl. *open source*) koje su podržane i korištene od velikog djela zajednice. Zbog ove snažne podr-

ške zajednice, Go biblioteke su robusne, dobro održavane i neprestano se unapređuju. Tijekom razvoja neke od bitnijih datoteka koje je vrijedno spomenuti su sljedeće:

- github.com/uptrace/bun - biblioteka koja pruža ORM (Object-Relational Mapping) funkcionalnost za rad s bazama podataka. ORM je tehnika koja omogućava programerima da koriste objekte i metode u programskom jeziku kako bi upravljali podacima u bazi podataka, umjesto da pišu nativne SQL upite. Bun omogućava definiranje modela podataka, povezivanje s bazom podataka i izvršavanje upita koristeći Go kod
- github.com/swaggo/swag - biblioteka koja se koristi za generiranje dokumentacije za API-jeve u formatu Swagger. Swagger je popularni standard za opisivanje RESTful API-ja koji omogućava generiranje dokumentacije, testiranje API-ja i automatsko generiranje klijentskog koda. Swag pojednostavljuje proces generiranja Swagger dokumentacije za Go API-jeve i omogućava programerima da anotiraju svoj Go kod sa Swagger oznakama kako bi definirali putanje, parametre, odgovore i druge informacije o API-ju koje se zatim koriste za generiranje Swagger JSON ili YAML dokumentacije
- github.com/gin-gonic/gin - biblioteka koja olakšava razvoj brzih i učinkovitih web aplikacija. Gin koristi net/http paket iz standardne biblioteke Go-a, ali dodaje mnoge korisne funkcionalnosti i apstrakcije koje pojednostavljaju razvoj web aplikacija. Pruža podršku za routing, middleware, upravljanje zahtjevima i odgovorima, renderiranje HTML-a i mnoge druge funkcionalnosti. Gin je dizajniran da bude brz i ima malu potrošnju memorije, što ga čini pogodnim izborom za razvoj visoko performantnih web aplikacija u Go-u

Integracija ChatGPT-a

OpenAI je istraživačko-razvojni laboratorij koji je osnovan s ciljem promoviranja i razvoja napredne umjetne inteligencije (engl. *Artificial Intelligence*) (**AI**). Osnovana 2015. godine, tvrtka OpenAI je postala jedna od vodećih institucija koja se bavi istraživanjem i razvojem umjetne inteligencije, te istovremeno promovira transparentnost, odgovornost i etičko korištenje ovih tehnologija. Cilj OpenAI-a je stvoriti umjetnu inteligenciju koja je sigurna, korisna i dostupna svima, umjesto da bude ograničena samo na nekoliko organizacija ili interesnih skupina. OpenAI nastoji izgraditi tehnologije koje poboljšavaju ljudske sposobnosti, potiču suradnju između ljudi i strojeva te potiču razvoj socijalno korisnih aplikacija.

Jedan od najpopularnijih proizvoda koje je OpenAI razvio je ChatGPT. Ovaj razgovorni model predstavlja jedan od najnovijih dostignuća u umjetnoj inteligenciji i

omogućuje interakciju s računalom putem prirodnog jezika. ChatGPT koristi duboko učenje i tehnike obrade prirodnog jezika kako bi razumio postavljena pitanja i generirao relevantne odgovore. Nastao kao rezultat kontinuiranog istraživanja i razvoja na polju umjetne inteligencije, ChatGPT je razvijen na temelju GPT-3.5 arhitekture. Taj model treniran je na ogromnom skupu tekstualnih podataka iz različitih izvora, uključujući knjige, članke, web stranice i forume. Ova široka paleta podataka omogućuje ChatGPT-u da bude informiran o mnogim temama i da pruži korisne odgovore na različite upite iz područja kao što su edukacija, istraživanje, tehnička podrška i ostalo. U posljednjih nekoliko mjeseci od kada je ChatGPT postao široko dostupan na tržištu, primjetan je sve veći porast popularnosti umjetne inteligencije, a velike kompanije prepoznaju njihov potencijal i sve više ulazu u njih. Uz sve veće ulaganje u AI tehnologije, očekuje se da će se ovaj sektor nastaviti razvijati i napredovati u budućnosti, a AI će postati još snažniji, pristupačniji i integriran u svakodnevne živote korisnika diljem svijeta.

U sklopu razvoja samog razgovornog modela, OpenAI je također osigurao i ChatGPT API (engl. *Application Programming Interface*). API omogućuje razvijateljima pristup i integraciju ChatGPT-a u svoje aplikacije i sustave. Korištenjem API-ja, razvijatelji mogu koristiti sve benefite razgovornog modela i iskoristiti ga za pružanje poboljšane interakcije s korisnicima.

S obzirom na to da je za vrijeme pisanja ovog rada popularnost raznih generativnih modela na samome vrhuncu, odlučio sam probati integrirati ChatGPT u samu aplikaciju. Pošto je u aplikaciji planiran sustav igrifikacije po principu novčića (engl. *coin operated game*), potrebno je bilo osmisliti na koji način će korisnik sakupljati novčiće koje želi potrošiti. Upravo ovdje dolazimo do ChatGPT-a, koji će poslužiti kao izvor za generiranje pitanja i odgovora iz područja programskog inženjerstva (engl. *software engineering*) na koje će korisnik morati odgovoriti. Važno je napomenuti da je ChatGPT, kao i svi ostali AI modeli, generativni model temeljen na treniranju na velikoj količini podataka, ali kao i svaki model, i on ima svoja ograničenja. Može generirati impresivne odgovore, ali također može biti podložan pogreškama, nejasnoćama i iznošenju netočnih informacija. Stoga je važno biti kritičan prilikom interpretiranja odgovora koje generira ChatGPT i potražiti dodatne izvore i potvrdu informacija ako je potrebno. S obzirom na to da je ovakva integracija u sustav samo pokazna, a da ChatGPT po subjektivnom mišljenju generira smislena pitanja i odgovore, prepostaviti ćemo da je za potrebe ovog rada generativni model točan te se njegovi izvori neće dodatno provjeravati.

Biblioteka go-openai, koju je razvio Sasha Baranov, je neformalni klijent za OpenAI API napisan u programskom jeziku Go. Biblioteka omogućuje interakciju s različitim

OpenAI modelima na jednostavan i prilagodljiv način. Sve što je potrebno za komunikaciju je dohvatiti API Token s ChatGPT stranice kako je prikazano u sljedećem isječku koda.

```
client := openai.NewClient("authorizationToken")
```

Nakon toga sama interakcija s modelom postaje poprilično intuitivna. Sve što je potrebno je odabrati određeni model s kojim se želi komunicirati te mu dati određenu poruku na temelju koje se zatim dobije odgovor. U sljedećem primjeru na slici 4.6 prikazano je kako se na jednostavan način može dohvatiti 3 pitanja iz područja računalnog inžinjerstva te odgovori na njih i format u kojem se očekuje odgovor od samog ChatGPT modela. Ovo nam olakšava to da uvjek možemo prepostaviti format odgovora koji ćemo dobiti te isti možemo pravilno transformirati, parsirati i poslati Android aplikaciji koja zatim ta pitanja prikazuje korisniku.

```
resp, err := client.CreateChatCompletion(
    ctx,
    openai.ChatCompletionRequest{
        Model: openai.GPT3Dot5Turbo,
        Messages: []openai.ChatCompletionMessage{
            {
                Role:    openai.ChatMessageRoleUser,
                Content: "Help me generate a quiz with 3 questions about random fields in software engineering." +
                    "Provide me a questions with up to 3 different answers. Tell me which questions is correct." +
                    "It needs to be in following json format:" +
                    "{\"quiz\": [\" +
                    "\"question\": \"Question\",\" +
                    "\"options\": [\" +
                    "\"Answer 1\",\" +
                    "\"Answer 2\",\" +
                    "\"Answer 3\",\" +
                    \"\"],\" +
                    "\"correctAnswer\": 2\" +
                    \"}]\" +
                    \"\",
            },
        },
    },
)
```

Slika 4.6: Komunikacija s ChatGPT modelom

Komunikacija REST-om

REST(Representational State Transfer)[8] je arhitekturni stil koji definira set ograničenja za izradu mrežnih(internetskih) aplikacija. REST se koristi za izgradnju web servisa koji su lagani, održivi i skalabilni. Izuzetno je popularan u modernim web aplikacijama te se često koristi za izgradnju API-ja koji omogućuju razmjenu podataka između klijenta i servera. Komunikacija REST-om najčešće se odvija putem HTTP protokola, a neke od metoda koje REST koristi uključuju GET, POST, PUT, DE-

LETE, i druge. Web servisi koji prate REST principi nazivaju se RESTful web servisi. U kontekstu ovog poslužiteljskog servisa implementiran je RESTful servis koristeći biblioteku github.com/gin-gonic/gin, koja je jedan od najpopularnijih HTTP web okvira napisanih u Go-u. Sljedeći slika koda 4.7 prikazuje kako su postavljene neke od osnovnih ruta REST API-a.

```
gin.SetMode(gin.ReleaseMode)
router := gin.New()
router.Use(i18n.LocalizationMiddleware())
router.Use(cors.CORS(
    cors.WithAllowOrigins(cfg.Cors.AllowOrigins),
    cors.WithAllowCredentials(cfg.Cors.AllowCredentials)))
router.Use(requestid.New())
router.Use(gin.Recovery())

router.GET(relativePath: "/health", http.HealthCheck())
router.GET(relativePath: "/metrics", gin.WrapH(promhttp.Handler()))
router.GET(relativePath: "/readiness", http.ReadinessCheck(database))
router.GET(relativePath: "/swagger/*any", ginSwagger.WrapHandler(swaggerFiles.Handler, ginSwagger.InstanceName(name: "v1")))

v1Group := router.Group(relativePath: "/api/v1")

v1Group.POST(relativePath: "/user", v1.CreateUser(dbRepository.UserRepository()))
v1Group.PUT(relativePath: "/user/:email/updateCoins", v1.UpdateCoins(dbRepository.UserRepository()))
v1Group.GET(relativePath: "/user/:email", v1.FindUserByEmail(dbRepository.UserRepository()))
v1Group.GET(relativePath: "/user/:email/coins", v1.GetUserCoinsByEmail(dbRepository.UserRepository()))
```

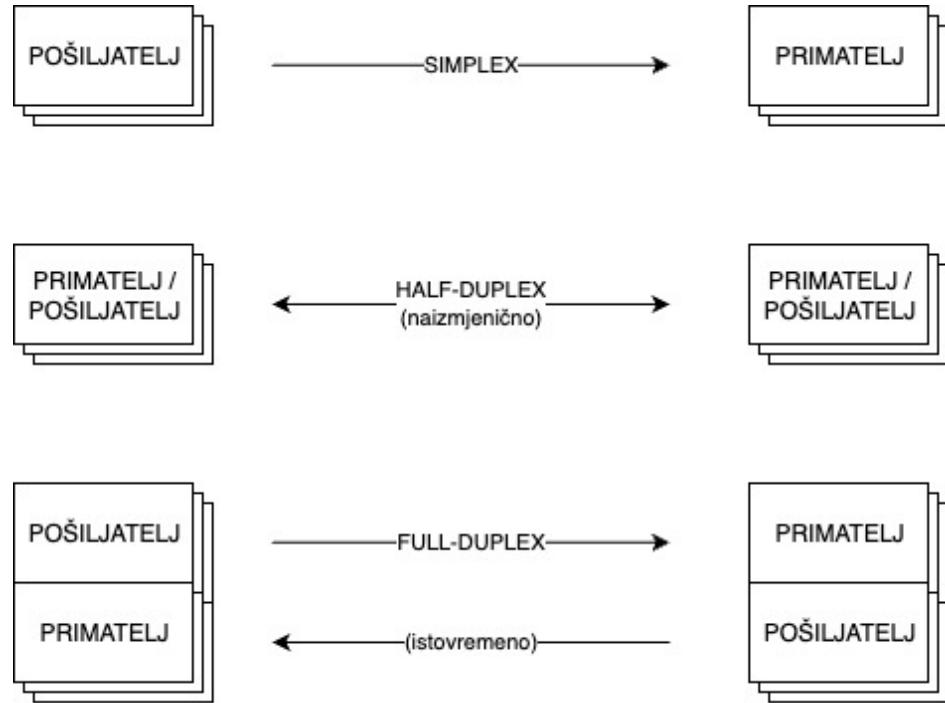
Slika 4.7: REST API na poslužiteljskoj strani

U ovom isječku koda, možemo vidjeti kako su definirane različite rute (endpointi) za upravljanje različitim resursima. Na primjer, ruta "/user" ima metode POST i GET koje se koriste za stvaranje novih korisnika i dohvaćanje postojećih, respektivno, dok se metoda PUT se koristi za ažuriranje korisničkih novljičića. Kroz ovu vrstu strukture, možemo lako definirati i upravljati različitim funkcionalnostima servisa, pružajući intuitivan i učinkovit način za interakciju s klijentskom aplikacijom.

Komunikacija WebSocket-om

Kako bismo mogli objasniti komunikaciju WebSocket-om, prvo je potrebno objasniti što je WebSocket zapravo. WebSocket je komunikacijski protokol koji pruža full-duplex komunikacijske kanale preko jedne TCP veze. Full-duplex komunikacijski kanal omogućuje dvosmjernu komunikaciju između dvije strane simultano. To znači da oba sudionika mogu slati i primati informacije u isto vrijeme, bez potrebe za prebacivanjem između slanja i primanja. Za razliku od HTTP-a, koji je bez stanja i svaka nova veza zahtijeva novi zahtjev i odgovor, WebSocket omogućava dvosmjernu komunikaciju između klijenta i servera bez potrebe za ponovnim uspostavljanjem veze. To ga čini idealnim za situacije gdje je potrebna brza, stvarna interakcija između klijenta i servera, kao što je to slučaj s chat aplikacijama, igranjem igara u stvarnom vremenu,

praćenjem događaja uživo i drugim primjenama. Više informacija o WebSocketu te njegovoj razlici u odnosu na HTTP može se pronaći na [9]. Primjer obične simplex, half-duplex te full-duplex komunikacije i njihova vizualna usporedba vidljiva je na slici 4.8.



Slika 4.8: Primjer simplex, half-duplex i full duplex komunikacije

U slučaju ove aplikacije, komunikacija WebSocketom se odvija u nekoliko faza. Te faze uključuju upravljanje sobama(aktivnim igram) i korisnicima unutar tih soba, a možemo ih podijeliti u nekoliko faza:

- Inicijalizacija WebSocket Hub-a i Handler-a** Dvije najbitnije komponente vezane uz WebSocket su Handler i Hub koji se inicijaliziraju prilikom pokretanja servisa. Handler se koristi za obradu svih dolazećih WebSocket zahtjeva poput stvaranja sobe i pridruživanja sobi. Hub predstavlja centralno mjesto gdje se događaju sve operacije poput registracije novih korisnika, odjave korisnika i slanja poruka svim korisnicima što možemo vidjeti u priloženom kodu na slici 4.9.

```

func (h *Hub) Run() {
    for {
        select {
        case cl := <-h.Register:
            if _, ok := h.Rooms[cl.RoomUUID]; ok {
                r := h.Rooms[cl.RoomUUID]

                if _, ok := r.Clients[cl.UUID]; !ok {
                    r.Clients[cl.UUID] = cl
                }
            }

        case cl := <-h.Unregister:
            if _, ok := h.Rooms[cl.RoomUUID]; ok {
                if _, ok := h.Rooms[cl.RoomUUID].Clients[cl.UUID]; ok {
                    if len(h.Rooms[cl.RoomUUID].Clients) != 0 {
                        h.Broadcast <- &Message{
                            ActionType: "event",
                            Content:    "user left the chat",
                            RoomUUID:   cl.RoomUUID,
                            Username:   cl.Username,
                        }
                    }

                    delete(h.Rooms[cl.RoomUUID].Clients, cl.UUID)
                    close(cl.Message)
                }
            }
        }

        case m := <-h.Broadcast:
            if _, ok := h.Rooms[m.RoomUUID]; ok {

                for _, cl := range h.Rooms[m.RoomUUID].Clients {
                    cl.Message <- m
                }
            }
        }
    }
}

```

Slika 4.9: Upravljanje korisnicima putem Hub-a

2. **Kreiranje sobe** Kada klijent šalje zahtjev za kreiranjem sobe, koristimo *CreateRoom* funkciju koja unosi sobu u Hub i stvara novi zapis o sobi u bazi podataka. Jedna instanca sobe u ovom slučaju predstavlja jednu instancu aktivne igre na koju se korisnici mogu priključiti.

```
func (h *Handler) CreateRoom(repository RoomRepository) gin.HandlerFunc {
    return func(ctx *gin.Context) {

        logger := commonHttp.GetHttpLogger(ctx, methodName: "v1.CreateRoom")

        var roomForm form.CreateRoomForm

        if err := ctx.ShouldBindJSON(&roomForm); err != nil {
            logger.Error().Err(err).Msg( msg: "Failed to unmarshal create room request")
            ctx.AbortWithStatusJSON(http.StatusBadRequest, commonHttp.Error{
                Status: http.StatusBadRequest,
                Error: i18n.LocalizeMessage(ctx, key: "general.requestUnmarshalError"),
            })
            return
        }

        if err := roomForm.ValidateWithContext(ctx); err != nil {
            logger.Error().Err(err).Msg( msg: "Failed to validate create room request")
            ctx.AbortWithStatusJSON(http.StatusBadRequest, commonHttp.Error{
                Status: http.StatusBadRequest,
                Error: i18n.LocalizeMessage(ctx, key: "general.requestValidationErrors"),
            })
            return
        }

        h.hub.Rooms[roomForm.RoomUUID] = &Room{
            ID:      roomForm.RoomUUID,
            Name:    roomForm.Name,
            Clients: make(map[int64]*Client),
        }

        room, err := repository.CreateRoom(ctx, roomForm.MapToEntity())
        if err != nil {
            logger.Error().Err(err).Msg( msg: "Failed to create room")
            ctx.AbortWithStatusJSON(http.StatusInternalServerError, commonHttp.Error{
                Status: http.StatusInternalServerError,
                Error: i18n.LocalizeMessage(ctx, key: "general.internalServerError"),
            })
            return
        }

        logger.Info().Msg( msg: "Room " + room.Name + " with RoomUUID " + strconv.FormatInt(room.UUID, base: 10) + " created")
        ctx.Status(http.StatusCreated)
    }
}
```

Slika 4.10: Stvaranje nove sobe unutar Handler-a

3. **Pridruživanje sobi** Klijent se može pridružiti sobi slanjem *JoinRoom* zahtjeva. Ako soba postoji, klijent se dodaje u sobu i svim ostalim korisnicima u sobi se šalje poruka da se novi korisnik pridružio.
4. **Slanje i primanje poruka** Nakon što je korisnik pridružen sobi, mogu se slati i primati poruke sa strane svih korisnika trenutno pridruženih sobi. Za svaku

poruku, koristi se *Broadcast* kanal na Hub-u za slanje poruke svim preostalim klijentima u sobi.

Struktura WebSocket poruke koja se prenosi je sljedeća:

```
type Message struct {
    ActionType string `json:"actionType"`
    Content     string `json:"content"`
    RoomUUID   int64  `json:"roomUUID"`
    Username   string `json:"username"`
}
```

Websocket poruka sastoji se od četiri polja od kojih svako polje ima svoju svrhu i značaj za komunikaciju između klijenta i poslužitelja:

1. **ActionType** - označava tip akcije koju poruka predstavlja. Ovisno o vrijednosti ovog polja, poruka se može različito tumačiti i obrađivati na poslužiteljskoj strani. Tipovi akcija mogu uključivati primjerice "direction" za smjer, "event" za događaj, "navigate" za navigaciju i drugo, ovisno o specifičnim zahtjevima i funkcionalnostima aplikacije
2. **Content** - sadrži konkretnе podatke ili informacije vezane za akciju. Na primjer, ako je ActionType "direction", Content bi mogao specificirati kojom se točno smjeru korisnik pokušava kretati, konkretnije "UP" za gore, "DOWN" za dolje i ostalo
3. **RoomUUID** - predstavlja identifikator sobe unutar koje se akcija odvija. To omogućuje poslužitelju da zna u kojem kontekstu treba obraditi poruku i kojim korisnicima je potrebno proslijediti ažuriranja
4. **Username** - korisničko ime osobe koja inicira akciju. Omogućuje povezivanje akcija s određenim korisnicima, što može biti korisno za praćenje tko je izvršio koju akciju ili za slanje specifičnih odgovora pojedinim korisnicima

Ovakva poslužiteljska aplikacija omogućuje brzu, efikasnu i sigurnu komunikaciju između komponenata sustava. S ovom je implementacijom podržan pristup podatcima putem REST servisa te komunikacija u stvarnom vremenu putem WebSoketa.

4.2.2. Baza podataka i migracije

Kao baza podataka u ovom sustavu korištena je Azure SQL Edge baza podataka koja razvijatelju daje broje prednosti u pogledu skalabilnosti, dostupnosti i sigurnosti podataka. Za razliku od klasičnih principa za stvaranje sustava ove vrste, ova aplikacija nema samo statičku bazu koja se jednom pokrene, već je ona upravljana uz

pomoć migracija. Migracije su skup akcija koje se koriste za verzioniranje, stvaranje, mijenjanje ili brisanje strukture baze podataka. Kada se promijene struktura ili podaci baze podataka, nova migracija se stvara koja opisuje te promjene. Na taj način, svaka migracija ima svoju verziju, a sve promjene su zabilježene u povijesti. Ovo omogućava lakši razvoj i otklanjanje problema, a ujedno se lako može vratiti na prethodno stanje baze podataka ako je potrebno. Iako u slučaju ovog sustava ovakve migracije nisu bile potrebne, radi preventivnosti i lakšeg upravljanja bazom podataka u budućnosti, ovaj princip primjenjen je od samog početka implementacije.

U ovome sustavu migracije se pokreću preko već objašnjene Makefile skripte koja je prikazana na 4.11.

```
.PHONY: build build_local run_up run_down mod clean

CURRENT_DIR=$(shell pwd)
SERVICE_NAME=service
VERSION=$(shell cat ${CURRENT_DIR}/VERSION)

build: clean mod
    GOOS=linux GOARCH=amd64 go build -ldflags="-w -s -X 'main.version=$(VERSION)' -o $(SERVICE_NAME) ./cmd

build_local: clean mod
    go build -ldflags="-w -s -X 'main.version=$(VERSION)' -o $(SERVICE_NAME) ./cmd

run_up:
    go run ./cmd --up

run_down:
    go run ./cmd --down

mod:
    go mod download && go mod verify

clean:
    rm -rf $(SERVICE_NAME)
```

Slika 4.11: Make skripta za migracije

Jednom kad se u terminalu pokrene jedna od naredbi *make_run_up* ili *make_run_down*, pokreće se sam program koji izvršava migraciju. Program sa slike 4.12 učitava konfiguraciju, otvara vezu s bazom podataka koristeći dobivene podatke iz konfiguracije, te provjerava je li veza uspješna. Zatim, koristeći migracijski paket, otkriva dostupne migracije te ovisno o izabranoj opciji gdje je *up* naredba za nadogradnju, a *down* naredba za vraćanje, program izvršava navedenu naredbu. Ako se dogodi bilo kakva greška tijekom ovog procesa, ona će se zabilježiti u logove i migracija neće uspjeti.

```

func migrationAction(ctx *cli.Context) error {
    logger := log.With().Str(key: "version", version).Logger()
    logger.Info().Msg(msg: "Running migration")

    // global application context
    appCtx, cancel := context.WithCancel(ctx.Context)
    defer cancel()

    cfg, err := loadConfiguration(ctx.String(name: "config"))
    if err != nil {
        logger.Fatal().Err(err).Msg(msg: "Failed to load configuration")
    }

    sqlDB, err := sql.Open(
        driverName: "sqlserver",
        fmt.Sprintf("sqlserver://#{cfg.Username}:#{cfg.Password}@#{cfg.Host}:#{cfg.Port}?database=#{cfg.Database}"),
    )
    db := bun.NewDB(sqlDB, mssqlialect.New(), bun.WithDiscardUnknownColumns())
    if err := db.Ping(); err != nil {
        logger.Fatal().Err(err).Msg(msg: "Failed to connect to the database")
    }
    defer db.Close()

    var migrations = migrate.NewMigrations()
    err = migrations.Discover(migration.GetMigration())
    if err != nil {
        logger.Fatal().Err(err).Msg(msg: "Failed to load migration files")
    }
    migrator := migrate.NewMigrator(db, migrations)

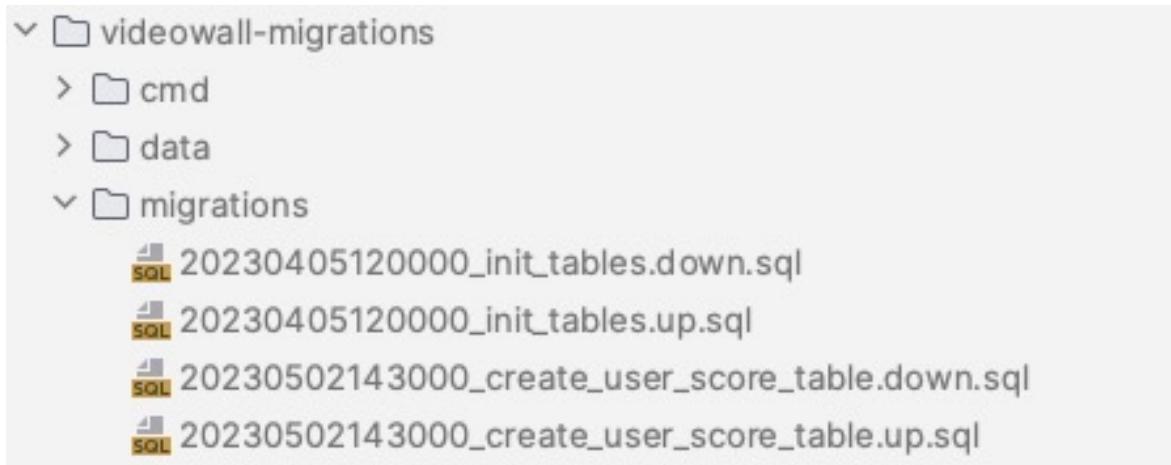
    _ = migrator.Init(appCtx)
    if ctx.Bool(name: "up") {
        logger.Info().Msg(msg: "Applying migration")
        _, err = migrator.Migrate(appCtx)
    } else if ctx.Bool(name: "down") {
        logger.Info().Msg(msg: "Rollback migration")
        _, err = migrator.Rollback(appCtx)
    }
    if err != nil {
        logger.Fatal().Err(err).Msg(msg: "Failed to run migration")
    }

    logger.Info().Msg(msg: "Migration successfully done")
    return nil
}

```

Slika 4.12: Go kod za migracije

Migracijske skripte koje se dodaju moraju biti u formatu koji zadovoljava principe biblioteke koja se koristi. Stoga svi fileovi moraju zadovoljavati standard koji odgovara sljedećem formatu : **YYYYMMDDHHMMSS_naredba.(down/up).sql** te je vidljiv na slici 4.13.



Slika 4.13: Struktura migracijskih skripti

4.2.3. Mobilna aplikacija

Za implementaciju mobilne aplikacije odabran je operacijski sustav Android. Android je trenutno najrašireniji mobilni operacijski sustav na svijetu, što ga čini prirodnim izborom za svaki projekt koji cilja na širok doseg. Iako je fokus trenutno na Android aplikaciji, važno je napomenuti da nije isključena mogućnost razvoja iOS verzije u budućnosti. Ovisno o potrebama i zahtjevima korisnika, projekt bi se mogao proširiti i na iOS platformu, čime bi se dodatno povećala dostupnost aplikacije te broj korisnika.

Kotlin ili Java

Kada je riječ o razvoju softvera za Android platformu, postavlja se pitanje o izboru programskog jezika. Dva su jezika koja dominiraju ovim područjem - Java, najpoznatiji jezik koji se već dugo koristi u svijetu programiranja, i Kotlin, noviji jezik koji je nastao kao svojevrsna zamjena za Javu. Razumijevanje prednosti i nedostataka svakog jezika može voditi do boljih odluka i učinkovitijeg koda. Iz tog sam razloga u sklopu mentoriranog predmeta "Seminar" napravio usporedbu programskih jezika Java i Kotlin. U ovom poglavlju samo ću se djelomično dotaknuti nekoliko koncepcata pisanja koda koje su istražene u sklopu seminarinskog rada te objasniti odabir programskog jezika.

Odmah na prvi pogled jedna od prednosti koja se primjećuje kod pisanja kod je da Kotlin ne zahtijeva korištenje točke sa zarezom na kraju svake linije koda. Od mehanizma koje pruža treba spomenuti integrirani null safety mehanizam koji smanjuje mogućnost NPE-a(NullPointerExceptiona), što je jedna od vrlo čestih pogrešaka u Javi. Kad već spominjemo iznimke (engl. *exceptions*), Kotlin ne koristi provjeravane iznimke (engl. *checked exceptions*), što je u ovom svijetu dosta kontroverzna tema. Razlog za to je više, a u posljednje vrijeme česta je priča kako provjeravane iznimke nisu

dobre i poželjne te je čak pokrenuta i inicijativa da se izuzmu iz novijih verzija Java. Još jedan od izrazito korisnih alata koji Kotlin nudi su funkcije ekstenzije, koje omogućuju proširenje postojećih klasa novim funkcijama bez nasljeđivanja klase. Kotlin ne koristi primitivne tipove podataka poput Java. Umjesto toga, koristi vrijednosti val i var. Varijable se automatski provjeravaju i pretvaraju (engl. *cast*) tamo gdje je to moguće, što omogućuje značajka smart cast. Kotlin nam omogućuje značajku koja je popularna u jezicima poput C++, a koja se naziva preopterećenje operatora (engl. *operator overloading*). To nam omogućuje da funkciju označimo sa operator te tim putem promijenimo značenje određenog operatora. Postoji još mnogo razlika u odnosu Java i Kotlin-a, poput načina formiranja petlji, instanciranja Singleton klase, formiranja podatkovnih klasa i drugih koje su detaljnije objašnjene u samom radu.

Osim navedenih prednosti, Kotlin je postao primarni jezik za razvoj Android aplikacija i iz još nekoliko razloga. Kotlin je iznimno dobro integriran s Android Studiom, najpopularnijim razvojnim okruženjem za Android. Google je uložio puno resursa kako bi osigurao glatku i učinkovitu integraciju Kotlina Android Studia, što povratno rezultira puno efikasnijem razvoju samih aplikacija, a od 2017. godine i službeno je glavni jezik za izradu Android aplikacija. Unatoč svojoj dugoj i uspješnoj povijesti u razvoju Android aplikacija, čini se da Java polako gubi prednost u korist Kotlin-a, zahvaljujući njegovoj modernijoj sintaksi, boljim značajkama i podršci od strane Googlea. Upravo iz tih razloga kao jezik za razvoj ove aplikacije odabran je Kotlin.

Jetpack Compose

Jetpack Compose[10] je moderna, deklarativna biblioteka za izradu korisničkih sučelja (engl. *User Interface*) na Androidu, koju je Google predstavio 2019. godine i od tada je postao standard za izradu Android aplikacija. Ovaj alat donosi potpuno novi način izrade korisničkih sučelja, koji se temelji na deklarativnom stilu programiranja, za razliku od tradicionalnog imperativnog stila kojim se prije koristilo u Android razvoju.

Prije uvodenja Jetpack Compose, izrada korisničkih sučelja na Androidu temeljila se na upotrebi XML-a (Extensible Markup Language). U XML-u, korisničko sučelje se definira statički, unaprijed definiranjem rasporeda elemenata na zaslonu. To znači da su promjene sučelja, poput promjene boje gumba ili prikaza različitih elemenata na temelju stanja aplikacije, morale biti ručno upravljane pomoću programskog koda. XML je omogućavao odvajanje dizajna korisničkog sučelja od logike aplikacije, što je bilo korisno u razvoju Android aplikacija. No, uporaba XML-a za definiranje korisničkih sučelja također je imala nekoliko izazova. Veliki izazov je složenost pisanja i održavanja XML koda. XML kod može postati vrlo velik i složen, posebno u većim projektima, što ga čini teškim za razumijevanje i održavanje. Još jedan problem je

poteškoća vizualizacije kako će korisničko sučelje izgledati samo gledajući XML kod. Iako su razvojni alati poput Android Studija pružali vizualne uredivače za lakšu izradu sučelja, oni nisu uvijek bili potpuno točni u prikazu kako će sučelje izgledati na stvarnom uređaju. Osim toga, XML je zahtijevao pisanje dodatnog "međukoda" za povezivanje korisničkog sučelja i logike aplikacije. Ovaj međukod često se morao ručno ažurirati svaki put kada bi se napravila promjena u korisničkom sučelju ili logici aplikacije. Takav princip pisanja je često dovodio do povećane složenosti koda i mogućih grešaka.

S druge strane, Jetpack Compose koristi deklarativni pristup za izradu korisničkih sučelja. U deklarativnom pristupu, umjesto da programer izravno manipulira korisničkim sučeljem pomoću programskog koda, samo opisuje kako bi sučelje trebalo izgledati na temelju određenog stanja aplikacije. To znači da programer ne mora brinuti o tome kako ažurirati sučelje kada se stanje promjeni - Jetpack Compose automatski obnavlja korisničko sučelje kada se stanje ažurira. Osim što pojednostavljuje izradu korisničkih sučelja, Jetpack Compose također donosi nekoliko drugih prednosti. Njegova deklarativna priroda čini kod čišćim i lakšim za čitanje, što olakšava razumijevanje i održavanje koda. Osim toga, Jetpack Compose je izgrađen da bude kompatibilan s postojećom Android arhitekturom i API-ima, što znači da ga je lako integrirati u postojeće Android projekte. Također, budući da je izgrađen na Kotlinu, omogućuje napredne značajke ovog jezika, poput korutina i referentne transparentnosti.

Više detalja i usporedba između Jetpack Composea i XML-a može se pronaći na [10]. Iz svih ovih razloga, za izradu korisničkog sučelja u ovoj aplikaciji odabran je Jetpack Compose koji omogućava brzi i učinkoviti razvoj, s fleksibilnim, čitljivim i lako održavajućim kodom, čime olakšava razvoj visokokvalitetnih korisničkih sučelja za Android aplikacije.

Vrste kontrolera

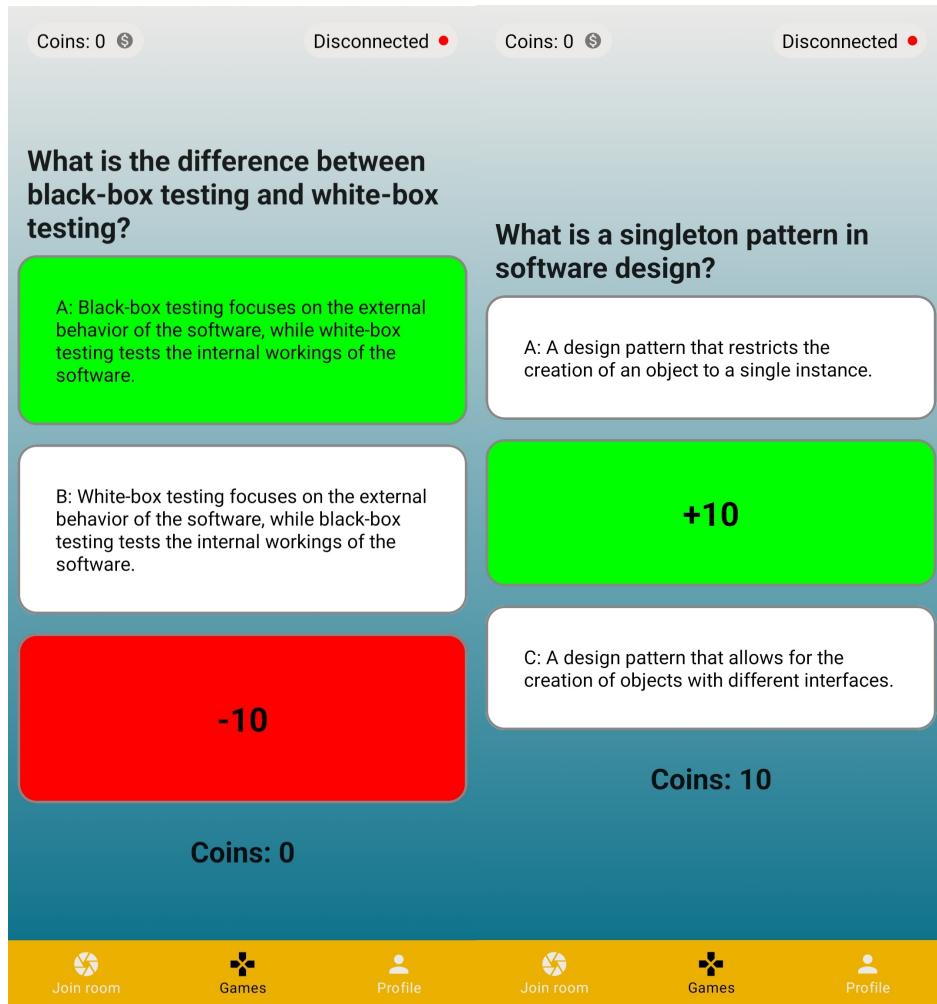
U sklopu razvoja komponente kontrolera postojala je potreba za odabir prikladnog kontrolera koji će omogućiti navigaciju prilikom igranja igara. Ispostavilo se da ne postoji jedan univerzalni kontroler koji bi mogao pokriti sve vrste igara koje bi potencijalno mogle biti implementirane. Iz tog je razloga implementiran samo jedan osnovni usmjereni D-pad kontroler (engl. *Directional pad*) koji korisniku omogućuje pomicanje u smjeru gore, dolje, lijevo ili desno. Takav kontroler uobičajeni je u igrama koje zahtijevaju jednodimenzionalnu navigaciju poput igre zmije (engl. *snake*). Još neke od vrsta kontrolera koje se ovisno o implementiranoj igri mogu uzeti u obzir te naknadno implementirani su pomični (engl. *slide*) kontroler te kontroler palicom (engl. *stick*). Primjer pomičnog kontrolera možemo vidjeti na trenutno postojećem FER Pong sus-

tavu u kojem se kontroler može pomicati gore-dolje te se poslužitelju dinamički šalje trenutna lokacija kontrolera. Kontroler palicom bio bi interesantan za implementaciju kod primjerice arkadnih i simulacijskih igrica gdje običan D-pad kontroler koji bilježi samo jedan smjer nije dovoljan, već je potrebno slati više detalja o kretnji kontrolera. Ipak, za potrebe ovog sustava, trenutno je implementiran samo D-pad kontroler, dok se ostali kontroleri mogu po potrebi implementirati u budućim verzijama aplikacije.

Implementacija

Sama funkcionalnost aplikacije već je nekoliko puta opisana, no ovdje ćemo se malo detaljnije dotaknuti implementacije same Android aplikacije. Kako bi se aplikacija koristila, obavezno je da korisnik bude prijavljen u sustav. Kada korisnik pokrene aplikaciju, prikazuje mu se početni ekran. Ako korisnik nije prijavljen, prikazuje mu se ekran za prijavu ili registraciju. Prijava i registracija se odvija putem Firebase servisa što će malo detaljnije biti objašnjeno u sljedećem poglavljju. Ako je korisnik prijavljen, automatski se preusmjerava na glavni ekran aplikacije. Glavni ekran sadrži navigacijsku traku na dnu ekrana, što omogućuje jednostavno prebacivanje između tri glavna ekrana aplikacije: ulazak u sobu, igra i profil.

Ekran "Igre" služi kao interaktivni hub za korisnika. Na ovom ekranu korisnik može igrati željene igre koristeći virtualni kontroler za upravljanje igrom, navigirati se kroz samu aplikaciju te odgovarati na pitanja kako bi sakupljao novčiće. Sakupljanje novčića odvija se odgovaranjem na pitanja koja se svakog puta nasumično generiraju korištenjem generativnog modela ChatGPT. Model generira pitanja s točnim odgovorom, a korisnik na temelju točnosti odgovora sakuplja novčiće. Samo generiranje pitanja već je objašnjeno u prethodnom poglavljju u sklopu poslužiteljske aplikacije. Trenutno je određeno da se korisniku prikazuje točno 3 pitanja na koja zatim odgovara, a ovisno o točnosti odgovora može dobiti ili izgubiti 10 bodova po pojedinom pitanju. Sljedeća slika 4.14 prikazuje odgovor na jedno točno i netočno pitanje. Možemo vidjeti kako je u slučaju točnog odgovora dodana animacija za dobivenih 10 bodova, a u slučaju netočnog za oduzimanje 10 bodova te označavanje zelenom bojom točnog odgovora.



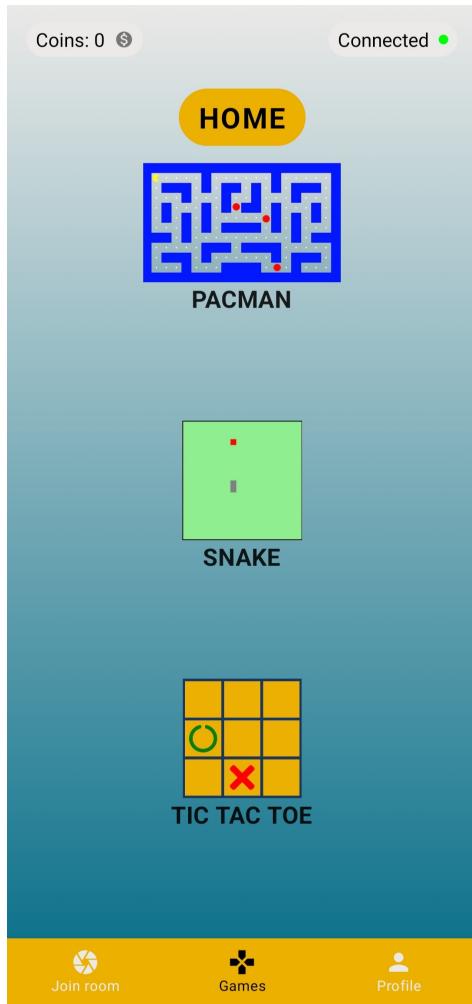
(a) Netočan odgovor na pitanje

(b) Točan odgovor na pitanje

Slika 4.14: Odgovaranje na pitanja i sakupljanje novčića

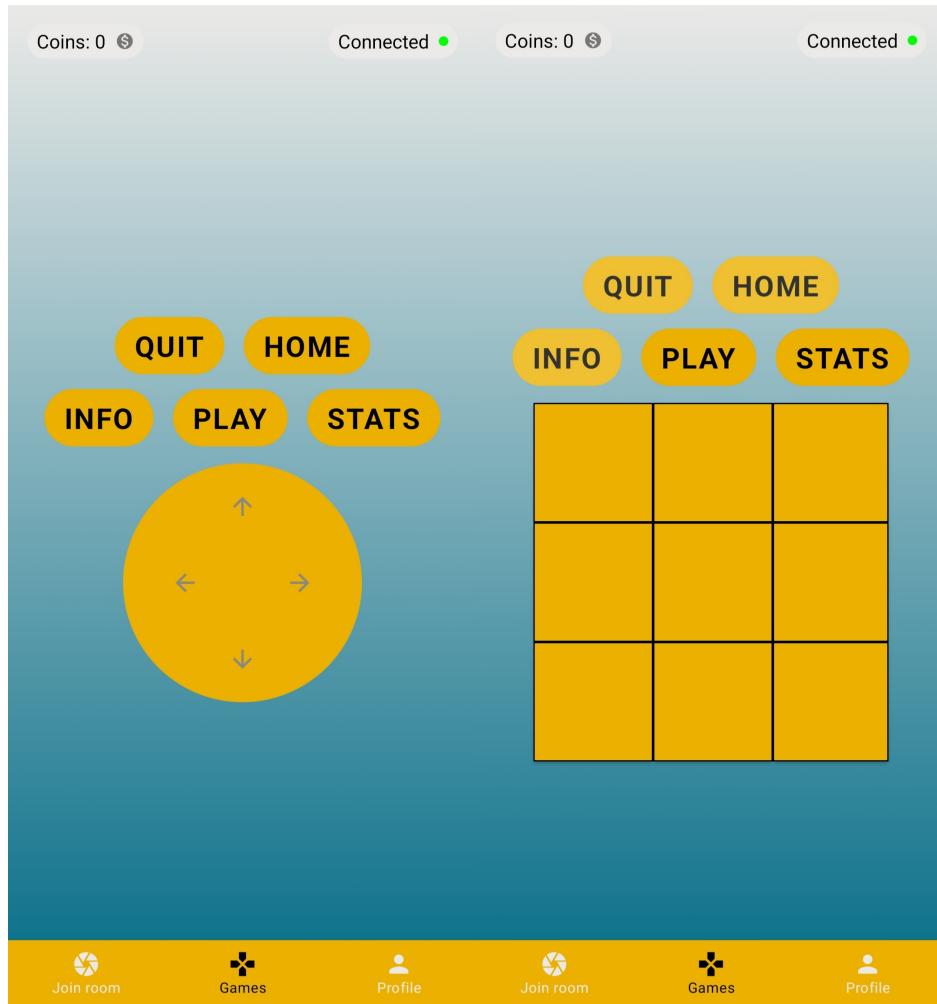
Ako korisnik nije prijavljen, na ekranu "Igre" jedina mogućnost mu je sakupljati novčiće. Trenutačno sakupljeni novčići se prilikom korištenja aplikacije mogu vidjeti u gornjem lijevom kutu aplikacije te se ažuriraju kroz korisničko korištenje istih.

Ako je korisnik prijavljen u sobu, u ekranu "Igre" vidi izbor trenutno dostupnih igara. Uz svaku igru prikazuje mu se slika te ime igre, dok detaljniji opis pojedine igre može vidjeti na web aplikaciji što će biti opisano u poglavljju kasnije. Slika 4.15 prikazuje izgled tog ekrana.



Slika 4.15: Ekran "Igre" s prikazom dostupnih igara

Otvaranjem pojedine igre korisniku se nudi opcija da pogleda više informacija o igri, pregleda listu najboljih 10 rezultata za danu igru ili započne igrati igru. Također ima opciju odustati te se vratiti na početni ekran. Akcija koju korisnike odabere prikazuje se na samome videozidu. Sljedeća slika 4.16 prikazuje različite prikaze kontrolera za upravljanje igram. Na slici pod a) možemo vidjeti D-pad kontroler koji je opisan u poglavlju ranije te služi za igranje "Snake" i "Pacman" igra u kojima nam je bitna kretanja u određenom smjeru. Pod slikom b) nalazi se svojevrstan kontroler za igranje igre križić-kružić putem kojeg korisnik može staviti simbol na mjesto na mreži i tako igrati u višekorisničkom načinu protiv drugog igrača.

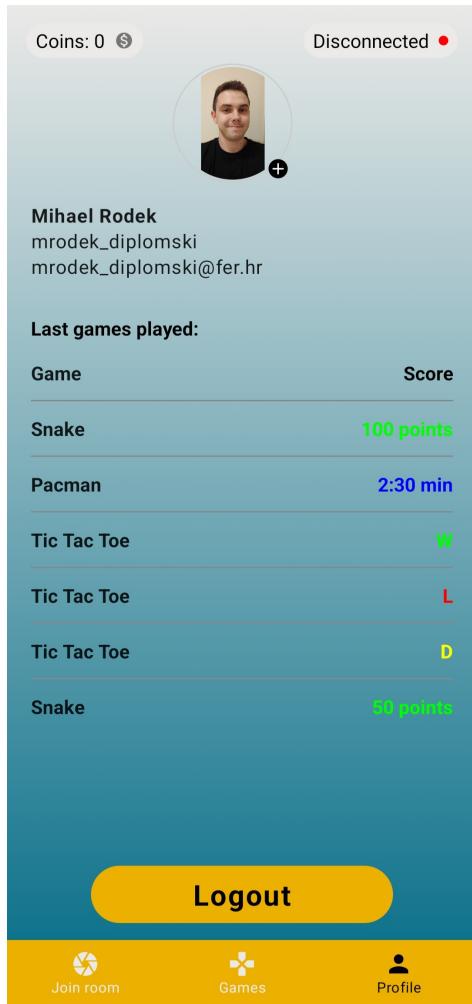


(a) D-pad mobilni kontroler

(b) Križić-kružić mobilni kontroler

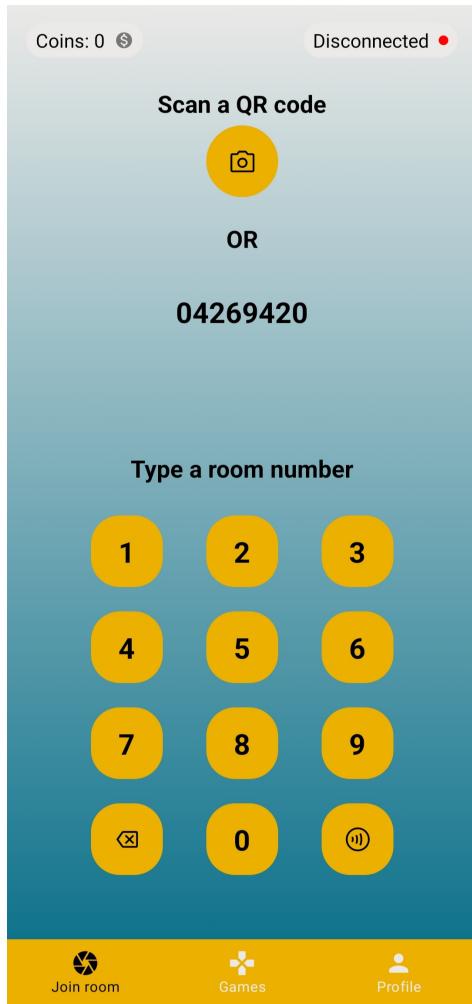
Slika 4.16: Kontroliranje različitih vrsta igara pomoću kontrolera

"Profil" je ekran koji nudi personalizirani pogled na korisničke informacije i postignuća. Ovdje korisnik može vidjeti svoje podatke s kojima se prijavio u sustav te promijeniti profilnu sliku. Uz to vidljiv je i prikaz zadnjih nekoliko igara koje je korisnik odigrao. Ovisno o odigranoj igri prikazan je određen rezultat koji je ostvaren. Na slici 4.17 vidljiv je prikaz ekrana "Profil" s navedenim informacijama. Ovdje se također nalazi i gumb za odjavu korisnika iz sustava.



Slika 4.17: Ekran "Profil" s prikazom korisničkih detalja i povijesti igara

Ekran za "Ulazak u sobu" omogućuje korisniku da se pridruži određenoj sobi, bilo skeniranjem QR koda putem kamere ili unošenjem ispravnog broja sobe. To pruža fleksibilnost korisnicima da se pridruže igri na način koji im najviše odgovara. U gornjem desnom kutu aplikacije stalno je prikazan plutajući prozor koji obavještava korisnika je li trenutno spojen u koju sobu ili ne. Ako je spojen pojavljuje se trepćući zeleni gumb na kojem piše "Connected" (spojen sa sobom), a ako nije pojavljuje se crveni gumb i piše "Disconnected" (odspojen od sobe). Ekran za ulazak u sobu možemo vidjeti na slici 4.18.



Slika 4.18: Ekran za ulazak u sobu

Čitač QR koda

QR kod je tip matričnog barkoda koji je zbog svoje jednostavnosti i brzine čitljivosti postao vrlo popularan, a sastoji se od crnih modula raspoređenih u kvadratni uzorak na bijeloj pozadini. Prethodnih je godina prilikom razvijanja aplikacija često postojala potreba za implementaciju čitača QR koda. U tu su se svrhu često koristile neke vanjske biblioteke koje su to rješavale zaobilaznim načinom pristupa kamери te je postojala potreba za razvojem intuitivnog rješenja. I ovo je područje koje je Google relativno nedavno odlučio unaprjediti te podržati stabilnu službenu implementaciju čitača QR koda. Google code scanner dolazi u sklopu ML Kit-a, alatnog seta razvijenog od strane Googlea koji omogućuje razvijateljima da implementiraju razna područja strojnog učenja poput obrade videa, teksta i slika u svoju aplikaciju. ML Kit sve svoje alate izvodi izravno na uređaju što znači da nije potrebna internetska veza. Umjesto pristupa kamери, u aplikaciju se u AndroidManifest.xml datoteku dodaje sljedeća ovisnost kako je prikazano u sljedećem djelu koda.

```
<application ...>
    ...
    <meta-data
        android:name="com.google.mlkit.vision.DEPENDENCIES"
        android:value="barcode_ui"/>
    ...
</application>
```

Ovo nam omogućuje da se ovisnosti potrebne za očitavanje QR koda direktno preuzmu kod preuzimanja same aplikacije. To znači da nije potrebno eksplicitno pristupati kamери, već se sve delegira i odvija preko Google play service-a koji obavlja posao skeniranja koda te rezultat prosljeđuje samoj aplikaciji. Ovakav princip je izrazito intuitivan za korištenje te razvijatelju omogućuje da što brže implementira samo očitavanje koda, dok u isto vrijeme zadržava sve potrebne vizualne i logičke funkcionalnosti. Jednom kad je čitač QR koda implementiran, njegov izgled nakon što se pritisne odgovarajući gumb u aplikaciji možemo vidjeti na slici 4.19.



Slika 4.19: Primjer korištenja čitača QR koda u Android aplikaciji

Firebase

Firebase je popularna platforma razvijena od strane Googlea, koja pruža brojne usluge i servise kod razvoja aplikacija, među kojima je i autentifikacija korisnika. Ovaj servis je posebno koristan jer omogućuje sigurnu i efikasnu autentifikaciju korisnika kroz različite metode, uključujući email i lozinku, telefon, te različite OAuth providere kao što su Google, Facebook, Twitter i drugi. Firebase je dizajniran da zadovolji stroge standarde sigurnosti i zaštite podataka, smanjujući rizik od sigurnosnih ranjivosti. Više o samom Firebase-u i njegovoj povijesti može se saznati na [11]. Osim toga, korištenje Firebasea za autentifikaciju znatno ubrzava razvoj aplikacije jer nije potrebno implementirati vlastiti autentifikacijski sustav. To smanjuje vrijeme razvoja, ali i potencijalne pogreške koje mogu nastati prilikom implementacije sigurnosnih sustava.

U kontekstu ovog projekta, Firebase je primarno korišten za autentifikaciju korisnika u mobilnoj aplikaciji. Autentifikacija korisnika se odvija na početnom ekranu

aplikacije gdje se korisnik prijavljuje i registrira u sustav, ovisno o tome ima li već postojeći račun u sustavu. Sljedeći kod sa slike 4.20 prikazuje način na koji se korisnik stvara u Firebase bazi koristeći email i lozinku koju unese. Kod registracije vrši se određena validacija unesenih polja. Iako bi u stvarnom sustavu ta validacija morala biti puno kompleksnija te uključivati razne provjere, ovdje se zbog jednostavnosti odvija samo osnovna validacija, primjerice da lozinka mora imati barem 6 znakova.

```
private fun createUserInFirebase(email: String, password: String) {

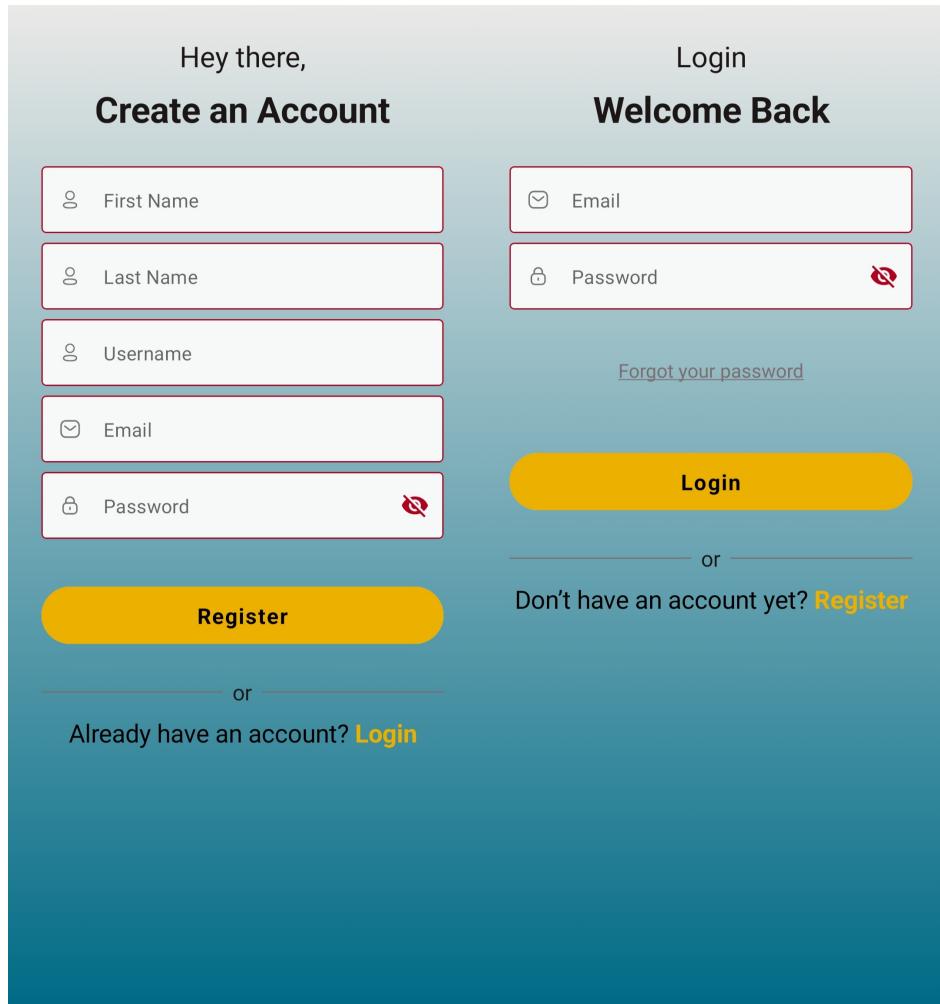
    signUpInProgress.value = true

    FirebaseAuth
        .getInstance()
        .createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener {
            Log.d(TAG, msg: "Inside_OnCompleteListener")
            Log.d(TAG, msg: "Registration successful = ${it.isSuccessful}")

            signUpInProgress.value = false
            if (it.isSuccessful) {
                navigateEvent.value = Screen.Main
            }
        }
        .addOnFailureListener {
            Log.d(TAG, msg: "Inside_OnFailureListener")
            Log.d(TAG, msg: "Exception= ${it.message} - ${it.localizedMessage}")
        }
}
```

Slika 4.20: Primjer koda koji stvara korisnika koristeći Firebase

Ekrani za prijavu i registraciju u sustav pomoću Firebsea prikazuju se korisniku samo u slučaju kada korisnik nije prijavljen u sustav. To je moguće prvi puta prilikom pokretanja aplikacije, jednom kada se korisnik ručno odjavi, ili svaka 3 sata koliko je postavljeno najduže moguće vrijeme prijave nakon čega slijedi automatska odjava. Te ekrane možemo vidjeti na slici 4.21.



(a) Registracija u sustav

(b) Prijava u sustav

Slika 4.21: Ekrani za autentifikaciju korisnika u sustavu

4.2.4. Web aplikacija

Kako je sama ideja aplikacije implementirati sustav za aplikacije na videozidu, tako web aplikacija unutar sustava upravo služi kao videozid na kojem se prikazuju interaktivne igre i korisnički unos koji je unesen putem mobilne aplikacije. Cilj je bio stvoriti aplikaciju koja je jednostavna i minimalistička u dizajnu, ali istovremeno dovoljno privlačna korisnicima da ih potakne na interakciju.

Korisničko sučelje

Upravo zbog toga je izbor pao na Chakra UI[12]. Chakra UI je moderna biblioteka za izradu sučelja, koja je napravljena za React i Next.js aplikacije. Ova biblioteka je karakterizirana svojom jednostavnosću korištenja, fleksibilnošću i pristupačnošću. Pruža bogat set komponenti koje su već prethodno stilizirane i spremne za upotrebu,

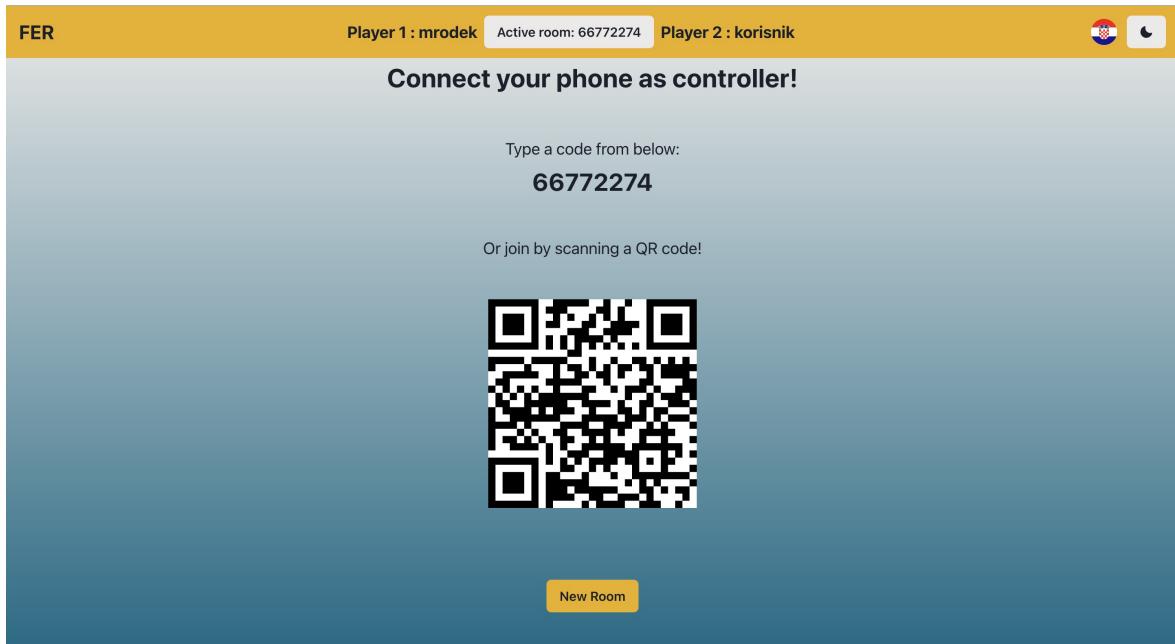
što značajno olakšava proces razvoja. Chakra UI se temelji na sustavu stilova koji omogućava razvojnim timovima da održavaju konzistentnost dizajna kroz cijelu aplikaciju, bez obzira na veličinu ili složenost samog projekta. Sve komponente u Chakra UI-u su potpuno prilagodljive i mogu se jednostavno prilagoditi specifičnim potrebama projekta. Upotrebom Chakra UI-a za ovaj projekt omogućeno je brzo i efikasno razvijanje korisničkog sučelja.

Kod prikaza boja u ovoj aplikaciji, naglasak je stavljen na usklađivanje boja s identitetom FER-a, kako bi se osigurala prepoznatljivost i integritet brenda unutar same aplikacije. Paleta boja koja se koristi pokušava pratiti neutralnu i dinamičnu paletu boja iz FER Brandbooka. Tako će dinamične boje poput Power Yellow i Deep Blue biti upotrijebljene za istaknute elemente, interaktivne ili akcijske gumbе kako bi se stvorio dojam dinamičnosti i privlačnosti, a neutralne boje, kao što su Silver grey i Deep space black, bit će korištene za pozadine, naglaske ili elemente dizajna koji pružaju ozbiljnost i umjerenost. Kombinacija ovih elemenata osigurat će da korisnici aplikacije prepoznaju FER-ov identitet i dožive njegov potpis kroz upotrebu odgovarajućih boja u samoj aplikaciji.

Implementacija

Svrha videozida je isključivo služiti kao ekran s kojim se upravlja preko kontrolera na mobilnom uređaju. Iz tog razloga sama web aplikacija nema previše vizualnih komponenata već je sve implementirano iznimno minimalistički. Sama ideja projekta bila je osmisliti sustav koji je višenamjenski, modularan i nadogradiv prema potrebama, za razliku od trenutnog FER Pong sustava koji se zasniva na samo jednoj aplikaciji. Upravo se oko toga kretala i implementacija same web aplikacije.

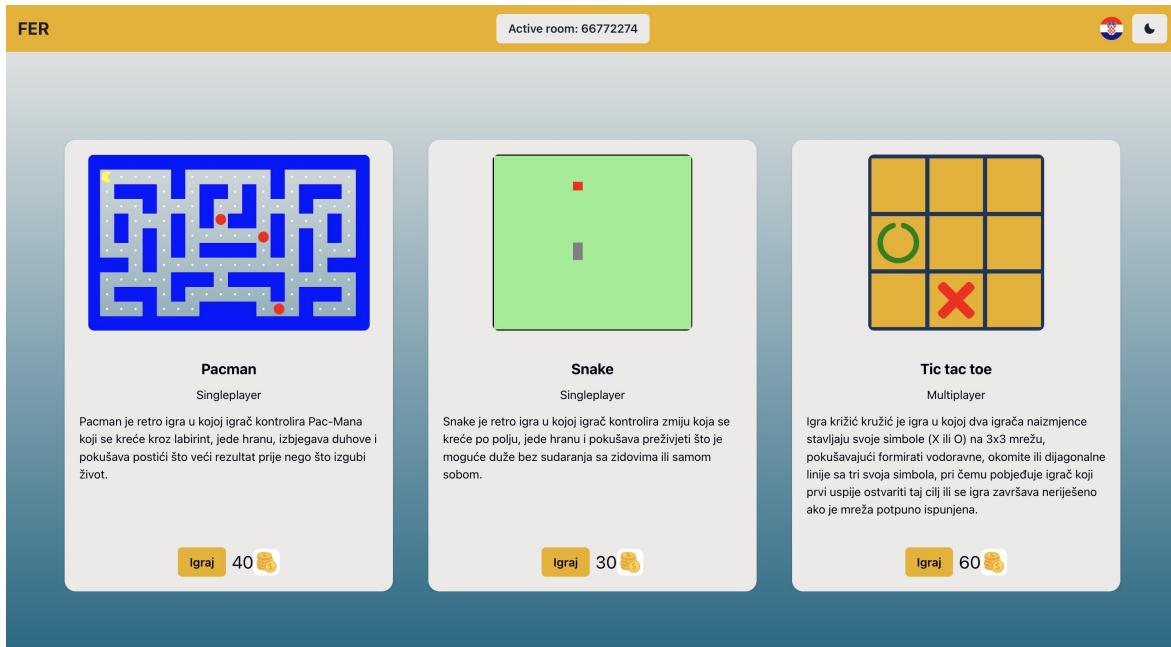
Početni ekran sustava je ekran koji prikazuje broj sobe i QR kod pomoću kojeg se korisnik može spojiti u trenutnu aktivnu sobu. Spajanje se odvija putem mobilne aplikacije kako je već bilo spomenuto. Primjer ekrana možemo vidjeti na slici 4.22. Ovdje možemo vidjeti još jednu bitnu komponentu sustava - navigacijsku traku. Ona po sredini prikazuje trenutni broj aktivne sobe, a s lijeve i desne strane do dva trenutno aktivna korisnika koji su prijavljeni u trenutnu sobu. S lijeve strane nalaze se alati za promjenu jezika aplikacije te promjenu iz svijetlog u tamniji način rada.



Slika 4.22: Ekran koji prikazuje mogućnost spajanja u sobu

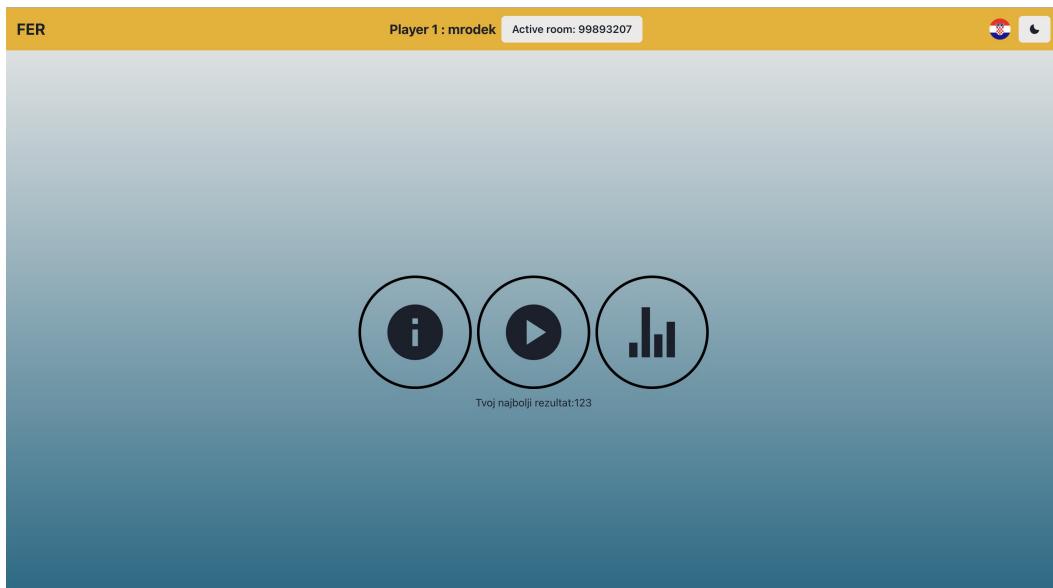
Jednom kada se barem jedan korisnik prijavi u sustav, navigacija se odvija isključivo putem mobilnog uređaja koji služi kao kontroler. Glavni ekran aplikacije je prikaz svih igara kojeg možemo vidjeti na 4.23 i u bilo kojem trenutku mu se može pristupiti putem mobilne aplikacije pritiskom na "Home".

Odabir igara za implementaciju u ovom sustavu pao je na klasične "retro" igre - Pac-Man, Snake (Zmija) i Tic-tac-toe (Križić-kružić) gdje prve dvije predstavljaju igru za jednog korisnika (engl. *singleplayer*), dok je zadnja višekorisnička igra (engl. *multiplayer*) za do 2 korisnika. Ove su igre odabrane zbog svoje jednostavnosti i prepoznatnosti, čime se olakšava korisnicima da brzo razumiju pravila i započnu igrati. Osim toga, te igre su intuitivne i pružaju zabavu bez obzira na to jesu li igrači iskusni ili novi u svijetu video igara.



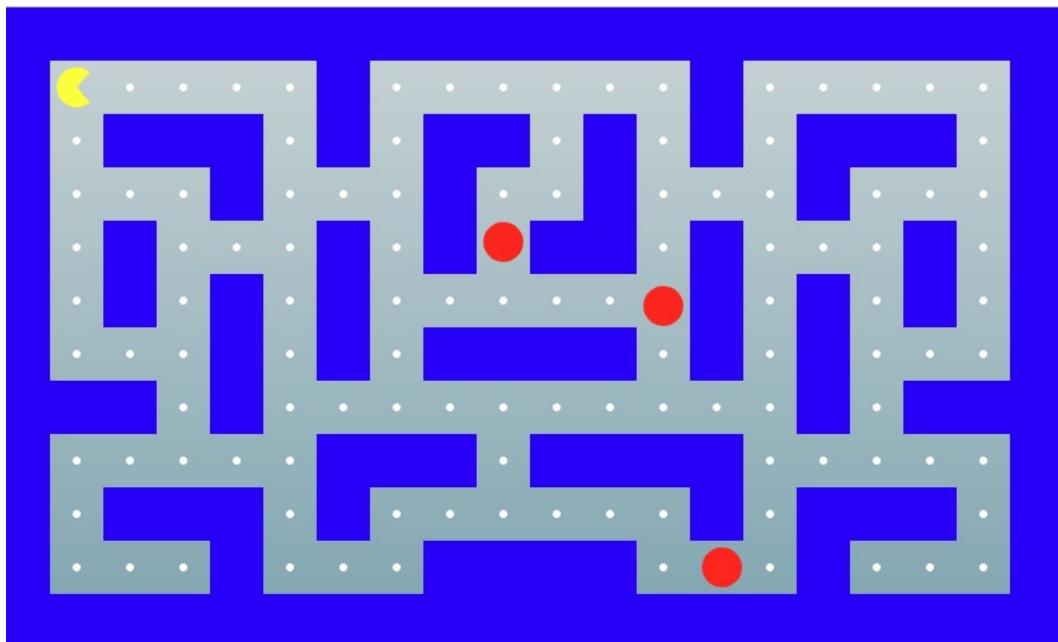
Slika 4.23: Glavni ekran aplikacije koji prikazuje prikaz svih igra

Cilj je bio napraviti što više generičkih komponenata koje se mogu na podjednaki način ponovno koristiti. Upravo na slici 4.24 možemo vidjeti primjer jedne takve komponente koju sadrži svaka igra. Ovu komponentu nasljeđuju sve igre te je iznimno jednostavno propagirati samo određene detalje koje vrijede za odabranu igru. Navedena komponenta pojavljuje se prilikom pritiska na gumb "Igraj" na glavnem ekranu, a korisniku omogućuje dodatnu interakciju sa sustavom. To su tri gumba koja redom predstavljaju modal za informacije o pojedinoj igri, mogućnost pokretanje igre te modal za statistiku trenutne igre.



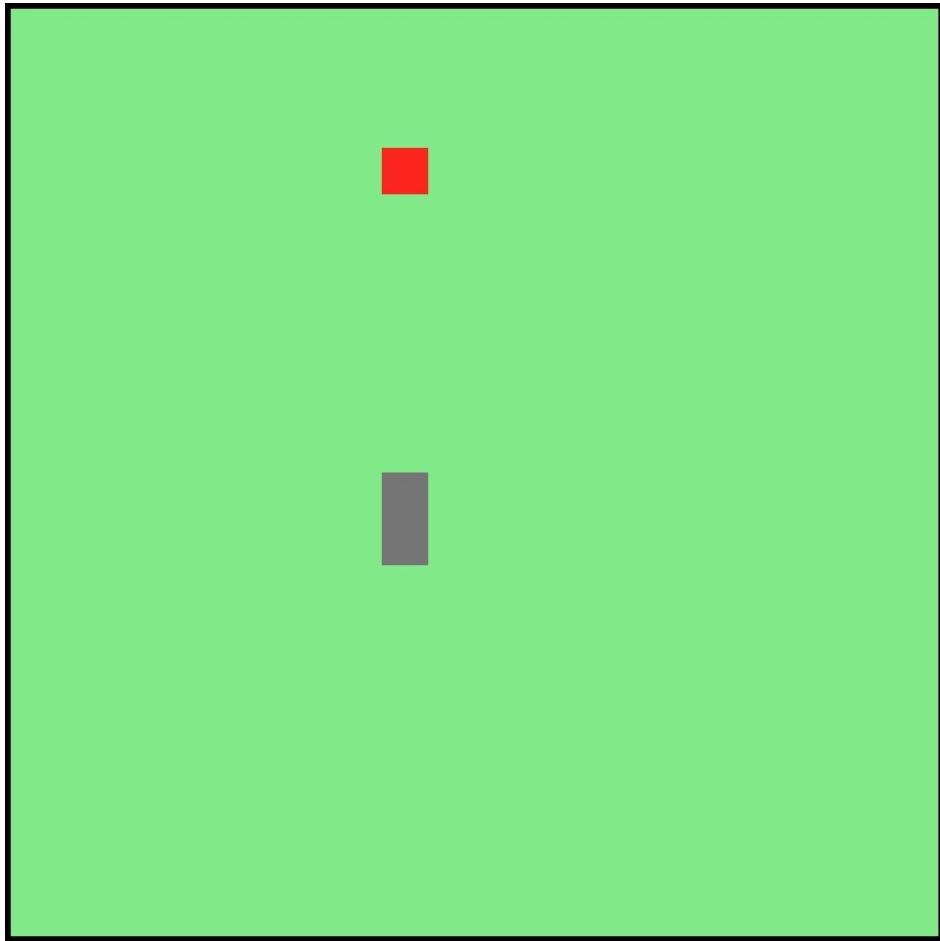
Slika 4.24: Komponente koje svaka igra sadrži

Pac-Man je jedan od najstarijih arkadnih hitova koji uključuje navigaciju kroz labirint dok jede hrani i izbjegava duhove. Cilj igre je pojesti svu hrani koja se nalazi u labirintu u isto vrijeme izbjegavajući duhove. Ako Pac-Man dođe u dodir s bilo kojim duhom, gubi život i igra se završava. Ova igra prikazana na slici 4.25 zahtijeva brzo razmišljanje i precizne odluke kod kretanja, a idealna je za kontroliranje pomoću D-pad kontrolera. U trenutnoj implementaciji lik kojeg korisnik kontrolira označen je crvenom bojom, dok je hrana bijela, a duhovi koje je potrebno izbjegavati crveni.



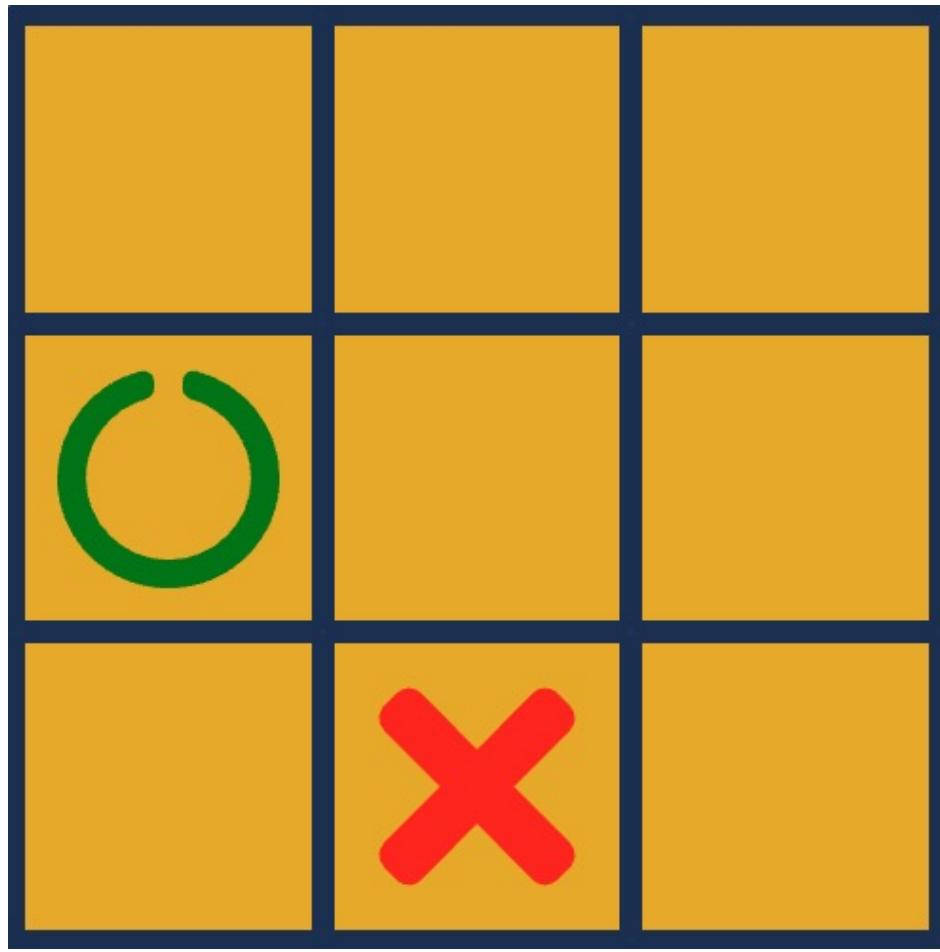
Slika 4.25: Igra Pac-Man

Snake, poznatija i kao Zmija, je klasična video igra koja se pojavila još u kasnim 70-ima. U ovoj igri, igrač upravlja zmijom koja se neprestano kreće po ekranu. Cilj igre je pojesti što više hrane. Svaki put kada zmija pojede hrani, ona raste za određeni broj segmenata i igrač dobiva bodove. Izazov u igri proizlazi iz činjenice da zmija postaje sve duža svaki put kada pojede hrani - što zmija postaje duža, to je teže izbjjeći sudaranje s vlastitim tijelom. Sudar zmije s vlastitim tijelom ili s granicama ekrana rezultira krajem igre stoga igrač mora pažljivo upravljati zmijom, održavajući balans između pokušaja da pojede što više hrane i izbjegavanja sudara. Kako igra napreduje, igra postaje sve teža što je zmija duža, a igrači se moraju osloniti na svoje refleks i sposobnosti planiranja kretanje zmije kako bi ostali u igri. U konkretnoj implementaciji zmija je prikazana sivom bojom, dok je hrana koju može pojesti prikazana crvenom bojom. Isto kao i Pac-Man, ova igra je idealna za implementaciju s jednostavnim D-pad kontrolerom kako je već ranije opisano.



Slika 4.26: Igra Snake

Tic-tac-toe ili križić-kružić je dobro poznata igra za dva igrača koja se često igra na običnom papiru ili na pametnom uređaju. Igru igraju dva igrački koji se izmjenjuju u postavljanju svojih simbola (obično križića **X** i kružića **O**) na 3x3 mrežu. Cilj igre je uspostaviti liniju od tri svoja simbola vodoravno, okomito ili dijagonalno prije nego što to učini protivnik. Jednostavnost ove igre čini je idealnom za implementaciju na videozidu. Svaki igrač koristi svoj mobilni uređaj kao kontroler, izmjenjujući se pri dodavanju svojih simbola na mrežu koja je prikazana na videozidu. To omogućava igračima da igraju igru na velikom ekranu, dodajući novu dimenziju ovom tradicionalnom obliku zabave. Iako je poznato da ako se igra optimalno, Tic-tac-toe je igra koja uvijek završava neriješeno, no usprkos toj činjenici, igra ostaje popularna zbog svoje jednostavnosti i potrebe za strateškim razmišljanjem. Primjer konkretne implementacije vidljiv je na slici 4.27.



Slika 4.27: Igra Tic-tac-toe

Nakon završetka svake igre, igrači imaju opciju ponovnog pokretanja igre. To omogućuje brzu i jednostavnu interakciju, bez potrebe za ponovnim navigiranjem kroz cijeli sustav, čime se osigurava kontinuirano korisničko iskustvo. Sustav također pruža mogućnost praćenja korisničkih rezultata. Za svaku odigranu igru, rezultat se automatski bilježi i povezuje s odgovarajućim korisnikom. Ova značajka omogućuje igračima da prate svoj napredak i uspoređuju svoje rezultate s drugim, dodajući u aplikaciju element igrifikacije. Ova statistika je dostupna kroz sučelje i može se koristiti za poticanje zdrave konkurenциje među igračima. Važno je napomenuti kako je čitav sustav dizajniran s ciljem modularnosti i skalabilnosti. Kao takav, otvoren je za buduće nadogradnje i poboljšanja, bilo da je riječ o dodavanju novih igara, poboljšanju sučelja ili dodavanju novih funkcionalnosti. Ovaj fleksibilan pristup osigurava da sustav može rasti i razvijati se kako bi zadovoljio sve veće potrebe korisnika i ostao relevantan u dinamičnom svijetu video igara, a više o tome bit će opisano u sljedećim poglavljima.

5. Edukacijski i igrifikacijski aspekti sustava

Uz samu implementaciju interaktivnog sustava, vrlo bitna stavka kod izrade sustava bilo je pobrinuti se na dostupne edukacijske i igrifikacijske (engl. *gamification*) elemente. Ti su elementi dizajnirani da poboljšaju angažman i sudjelovanje korisnika, transformirajući pasivne interakcije u aktivna i poticajna iskustva.

5.1. Elementi igrifikacije

Određeni elementi igrifikacije već su proučeni u sklopu mentoriranog rada "Seminar" prilikom kojeg se unaprjeđivao trenutno postojeći sustav FER Pong. Neke od metoda igrifikacije koje su u sklopu tog rada proučene su igra upravljana novčićem (engl. *coin operated game*) i sistem ljestvice (engl. *leaderboard*). S obzirom na samu ideju aplikacije, ove metode primjenjive su i u ovoj aplikaciji. Konkretno, zamišljeno je da je valuta s kojom se upravlja unutar aplikacije novčić (engl. *coin*). Novčić se može sakupiti točnim odgovaranjem na kviz pitanja. S prikupljenim novčićima korisnik može odigrati određenu igru prilikom čega mu se iz trenutne sume oduzima broj novčića koji posjeduje. Kada korisnik odigra igru, ovisno o tipu igre, bilježi se njegov rezultat. Ovisno o igri taj rezultat može biti vremenski ili bodovno određen. Za svaku igru postoji ljestvica najboljih igrača koja prikazuje najboljih 10 rezultata za trenutnu igru čiji primjer možemo vidjeti na slici 5.1. S tim se dodaje dodatan element igrifikacije koji stvara kompetitivnost među korisnicima. Uključivanje natjecateljskih elemenata poput ljestvica, postignuća i nagrada može značajno povećati angažman i motivaciju korisnika. Takva igrifikacija, koja koristi mehanizme slične onima u igram, može motivirati korisnike da se više uključe i postanu aktivniji u samom sustavu. Time se stvara dinamično i interaktivno okruženje koje potiče korisnike da nastave učiti i uspinjati se na ljestvici, čineći obrazovanje uzbudljivijim i privlačnijim.

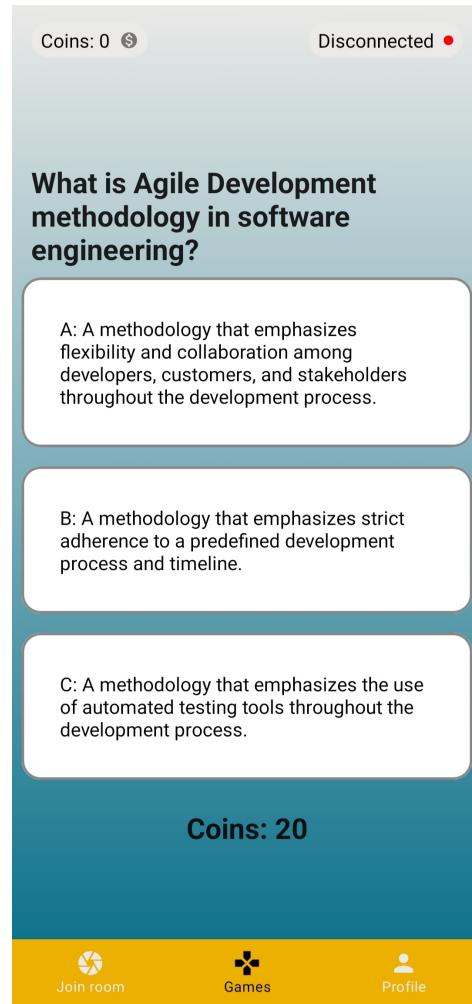
| MJESTO | IME | BODOVI | VRIJEME |
|--------|---------------------|--------|---------|
| 1 | test_videowall_user | 150 | 60 SEC |
| 2 | lorem_ipsum | 120 | 60 SEC |
| 3 | fer_videowall | 110 | 55 SEC |
| 4 | lorem_ipsum | 110 | 58 SEC |
| 5 | fer_videowall | 110 | 60 SEC |
| 6 | fer_user_1 | 100 | 60 SEC |

Slika 5.1: Ljestvica korisnika za određenu igru

5.2. Edukacijski elementi

Od edukacijskih elemenata ovog sustava, važno je istaknuti interaktivni modul za odgovaranje na pitanja. Ovaj modul omogućuje korisnicima da odaberu temu iz željenog područja interesa i na taj način personaliziraju svoje iskustvo učenja. Izabrana tema koristi se kao ulazni podatak za dohvaćanje relevantnih pitanja koristeći ChatGPT API, čime se osigurava da svaki korisnik dobiva jedinstvena nasumična pitanja prilagođena njegovim specifičnim interesima. Ključni dio ovog procesa je motivacija korisnika za točnim odgovaranjem na postavljena pitanja. Točni odgovori nagrađuju se "novčićima" unutar aplikacije, koji su integralni dio sustava igrifikacije. Na taj način, sustav kombinira učenje s elementima igre, čime se povećava angažman korisnika i motivacija za učenje.

U poglavlju o Android aplikaciji već je dan uvid u primjer kako izgledaju pitanja, a još jedan primjer pitanja možemo vidjeti na slici 5.2.



Slika 5.2: Primjer pitanja na koje se traži odgovor

6. Diskusija i analiza

6.1. Limitacije i izazovi

Prilikom razvijanja sustava, kao što to obično biva, došlo je do nekoliko manjih ograničenja i izazova koji su djelom utjecali na razvoj i konačan rezultat. Prvo i možda najvažnije ograničenje bilo je ono vremensko. Iako je vremenski okvir bio dovoljan za izradu osnovnog sustava, ograničavao je mogućnosti za razvoj sofisticiranijeg i detaljnijeg sustava u nekim segmentima aplikacije. Iako je postojala želja za izradom više igara, zbog vremenskog okvira to nije bilo moguće. Izazov je bio pronaći ravnotežu između kvalitete i kvantitete, a to je jedan od kompromisa koji se morao napraviti tijekom razvijanja projekta. Ipak, to idalje ostavlja mogućnosti za izradu novih aplikacija koje će samo nadograditi sustav. Drugi izazov do kojeg je došlo prilikom razvoja odnosio se na korištenje Jetpack Composea. Iako je ova nova biblioteka za izradu sučelja na Androidu vrlo inovativna i obećavajuća, također donosi i izazove. Jedan od ključnih problema je nedostatak obrazovnih resursa i dokumentacije. Budući da je biblioteka relativno nova, trenutno nedostaje detaljnih vodiča i primjera za neke složenije aspekte korištenja Jetpack Compose-a. To je u nekim slučajevima otežalo razumijevanje kako određene značajke funkcioniraju i kako ih najbolje primijeniti. Bez obzira na to, Jetpack Compose je u samome početku projekta odabran kao alat za razvijanje Android aplikacije, i kao takvog ga je trebalo prihvati zbog njegovog potencijala za izradu učinkovitijih i intuitivnijih korisničkih sučelja.

Ovo je poslužilo kao važna pouka: iako je privlačno uvijek koristiti najnovije i najinovativnije alate, oni često donose vlastite izazove i mogu komplikirati proces razvoja. Ponekad, sigurnije i efikasnije može biti korištenje dobro uspostavljenih tehnologija s bogatijim resursima za učenje i boljom podrškom zajednice. No, unatoč izazovima, korištenje Jetpack Compose-a omogućilo je stjecanje novih iskustava i nošenje s teškoćama novih tehnologija. Ovo iskustvo će zasigurno biti korisno u budućim projektima i pomoći u donošenju informiranih odluka o tome koje tehnologije je u kojem trenutku najbolje koristiti.

6.2. Poboljšanja i nadogradnje sustava

Kao što je već spomenuto u prethodnim poglavljima, postoji nekoliko mogućnosti za poboljšanje i nadogradnju trenutačnog sustava. Jedno od ključnih područja za poboljšanje je povećanje broja dostupnih igara. S obzirom na to da je trenutni broj implementiranih igara ograničen, dodavanje novih igara značajno bi povećalo zanimljivost sustava za korisnike. Dodatno, stvaranje novih igara povlačilo bi i razvoj više vrsta različitih kontrolera. Unutar sustava bi se mogle implementirati i funkcionalnosti za dodavanje prijatelja i povezivanja preko društvenih mreža. Korisnici bi tako mogli pratiti napredak svojih prijatelja, igrati igre s njima, ili čak i uspostaviti vlastite ljestvice među grupama prijatelja. Ovo bi moglo doprinijeti dubljem korisničkom angažmanu i značajno poboljšati društveni aspekt aplikacije.

U sklopu unaprjedenja edukacijskih elemenata aplikacije, moguće je uvođenje funkcionalnosti za spremanje određenih pitanja za kasniju reviziju. Na taj način, korisnici bi mogli kreirati svoju osobnu bazu pitanja koja bi mogla poslužiti kao materijal za daljnje učenje ili ponavljanje. Takav bi se sustav mogao povezati s nekom od postojećih baza pitanja poput primjerice onom koja postoji u sustavu Edgar [13].

Integracija reklama u mobilne aplikacije postala je jedan od standardnih načina monetizacije, posebno u sektoru besplatnih aplikacija. Moguće poboljšanje sustava bi moglo biti implementacija ovog modela. Konkretno, korisnici bi mogli gledati reklame u zamjenu za novčiće u igri. Ovo bi trebalo biti pažljivo urađeno kako bi se održala ravnoteža između korisničkog iskustva i monetizacije. Korištenje takvog modela može biti dvostruko korisno: ne samo da generira prihode, već i potiče korisnike da češće koriste aplikaciju kako bi sakupili više novčića. Ipak, potrebno je osigurati da su reklame relevantne i nemetljive kako ne bi ometale korisničko iskustvo.

Kada je riječ o estetici aplikacije, ključno je da korisničko sučelje bude atraktivno i intuitivno. To je posebno važno za aplikacije koje se koriste za zabavu, poput ove. Aktualni dizajn aplikacije mogao bi biti poboljšan u nekoliko aspekata. Određene komponente mogle bi se bolje implementirati sa strane korisničkog sučelja (UI) i iskustva (UX). Također je poželjno integrirati i animacije koje bi korisničko iskustvo učinile dinamičnijim i zanimljivijim. Animacije mogu pridonijeti cjelokupnom vizualnom dojmu aplikacije i učiniti je privlačnjom za korisnike.

Sve navedene nadogradnje bi dodatno obogatile korisničko iskustvo, stvorile veći osjećaj zajednice među korisnicima i potaknule daljnje korištenje aplikacije. No, treba uzeti u obzir da bi implementacija ovih značajki zahtijevala dodatno vrijeme i resurse, kao i pažljivo promišljanje o načinu na koji se te značajke integriraju u trenutačno korisničko sučelje.

7. Zaključak

Iako su videozidovi prvotno korišteni uglavnom za informativne i promotivne svrhe, ovaj rad pokazuje kako se uz odgovarajuću programsku podršku mogu transformirati u korisnički interaktivna sučelja i sustave. Cilj ovog rada bila je upravo ta transformacija - pretvoriti videozidove s tradicionalne, informativne upotrebe i ponuditi dizajn i implementaciju sustava koji će ih učiniti korisnički interaktivnim. Rad se sastoji od nekoliko komponenata : baze podataka, poslužiteljskog servisa, mobilne aplikacije i web aplikacije. Svaka od ovih komponenti igra važnu ulogu u radu sustava i njihova međusobna interakcija omogućava besprijekorno korisničko iskustvo. Primjerice, baza podataka koristi se za pohranu korisničkih podataka i informacija o igrama, poslužiteljski servis omogućava komunikaciju između različitih komponenti sustava, dok mobilna aplikacija služi kao svojevrstan kontroler za upravljanje web aplikacijom koja predstavlja videozid. Ovaj rad obuhvaća detaljnu implementaciju pojedine komponente te detaljne korake i razloge prilikom odabira tehnologija i alata kod razvijanja istih.

Kroz ovaj rad, stvorena je snažna osnova za daljnji razvoj i nadogradnju interaktivnog sustava. Za unaprjeđenje sustava postoji još mnogo implementacijskih detalja koji bi se mogli nadodati, a opisani su u prethodnim poglavljima, no za potrebe ovog rada zbog ograničenja dostupnih resursa uspješno su implementirani samo oni osnovni. Kako se tehnologija i korisničke potrebe nastave razvijati, a ovisno o prihvaćanju sustava unutar uže zajednice, postoji veliki potencijal za daljnje unaprjeđenje i proširenje ovog sustava, kako bi se poboljšala njegova funkcionalnost i korisničko iskustvo.

LITERATURA

- [1] N.-D. AG, *AirConsole Multiplayer Platform*, pristupljeno: 06. 05. 2023. Poveznica: <https://www.airconsole.com/>
- [2] J. Games, *Jackbox Games Multiplayer Platform*, pristupljeno: 06. 06. 2023. Poveznica: <https://www.jackboxgames.com/>
- [3] HappyFunTimes, *HappyFunTimes Multiplayer Platform*, pristupljeno: 07. 05. 2023. Poveznica: <https://docs.happyfuntimes.net/>
- [4] Ramseyjiang, *A Comprehensive Guide for Using Makefile In Golang Projects*, pristupljeno: 11. 06. 2023. Poveznica: <https://levelup.gitconnected.com/a-comprehensive-guide-for-using-makefile-in-golang-projects-c89edebcbe6e>
- [5] i18next community, *React i18n*, pristupljeno: 27. 05. 2023. Poveznica: <https://react.i18next.com/>
- [6] N. Snyder, *go-i18n*, pristupljeno: 27. 05. 2023. Poveznica: <https://github.com/nicksnyder/go-i18n/>
- [7] G. P. Languages, *Documentation*, pristupljeno: 29. 05. 2023. Poveznica: <https://go.dev/>
- [8] C. Team, *What is Rest?*, pristupljeno: 02. 06. 2023. Poveznica: <https://www.codecademy.com/article/what-is-rest>
- [9] A. Asai, *What is web socket and how it is different from the HTTP?*, pristupljeno: 29. 05. 2023. Poveznica: <https://www.scaler.com/topics/computer-network/websocket/>
- [10] A. D. L. Grana, *Jetpack Compose vs XML: Android UI Development Compared*, pristupljeno: 14. 06. 2023. Poveznica: <https://blog.moove-it.com/jetpack-compose-vs-xml-android-ui/>

- [11] D. Stevenson, *What is Firebase? The complete story, abridged*, pristupljeno: 14. 06. 2023. Poveznica: <https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0>
- [12] ChakraUI, *Documentation*, pristupljeno: 11. 06. 2023. Poveznica: <https://chakra-ui.com/>
- [13] FER, *Edgar - Sustav za održavanja online testova na FER-u*, pristupljeno: 15. 06. 2023. Poveznica: <https://edgar.fer.hr>

POPIS SLIKA

| | |
|---|----|
| 2.1. Igra FER Pong | 2 |
| 2.2. AirConsole Web sučelje | 4 |
| 2.3. AirConsole mobilna aplikacija | 4 |
| 2.4. Brojni članci koji objašnjavaju korištenje AirConsole API-a | 5 |
| 2.5. Paket "Jackbox Party Pack 9" (izvor jackboxgames.com) | 6 |
| 2.6. Razvijanje igre za HappyFunTimes (izvor happyfuntimes.net) | 7 |
| | |
| 3.1. Prijedlog arhitekture sustava | 11 |
| | |
| 4.1. GitLab ploča zadataka | 15 |
| 4.2. Docker compose | 16 |
| 4.3. Makefile datoteka | 18 |
| 4.4. Struktura lokalizacije | 19 |
| 4.5. Datoteka shared.json | 20 |
| 4.6. Komunikacija s ChatGPT modelom | 24 |
| 4.7. REST API na poslužiteljskoj strani | 25 |
| 4.8. Primjer simplex, half-duplex i full duplex komunikacije | 26 |
| 4.9. Upravljanje korisnicima putem Hub-a | 27 |
| 4.10. Stvaranje nove sobe unutar Handler-a | 28 |
| 4.11. Make skripta za migracije | 30 |
| 4.12. Go kod za migracije | 31 |
| 4.13. Struktura migracijskih skripti | 32 |
| 4.14. Odgovaranje na pitanja i sakupljanje novčića | 36 |
| 4.15. Ekran "Igre" s prikazom dostupnih igara | 37 |
| 4.16. Kontroliranje različitih vrsta igara pomoću kontrolera | 38 |
| 4.17. Ekran "Profil" s prikazom korisničkih detalja i povijesti igara | 39 |
| 4.18. Ekran za ulazak u sobu | 40 |
| 4.19. Primjer korištenja čitača QR koda u Android aplikaciji | 42 |
| 4.20. Primjer koda koji stvara korisnika koristeći Firebase | 43 |
| 4.21. Ekrani za autentifikaciju korisnika u sustavu | 44 |

| | |
|---|----|
| 4.22. Ekran koji prikazuje mogućnost spajanja u sobu | 46 |
| 4.23. Glavni ekran aplikacije koji prikazuje prikaz svih igra | 47 |
| 4.24. Komponente koje svaka igra sadrži | 47 |
| 4.25. Igra Pac-Man | 48 |
| 4.26. Igra Snake | 49 |
| 4.27. Igra Tic-tac-toe | 50 |
| 5.1. Ljestvica korisnika za određenu igru | 52 |
| 5.2. Primjer pitanja na koje se traži odgovor | 53 |

Sustav za podršku interaktivnim aplikacijama i igram na videozidu

Sažetak

U ovom diplomskom radu predstavljen je koncept transformacije videozida s tradicionalne, informativne upotrebe na korisnički interaktivne sustave. U radu je predložena arhitektura sustava te su odabrani i implementirani alati i tehnologije potrebni za ostvarenje pojedine komponente interaktivnog sučelja. Glavna značajka sustava je mobilna aplikacija koja služi kao kontroler za upravljanje web aplikacijom koja prikazuje korisničke interakcije na videozidu. Cijeli sustav osmišljen je kako bi bio modularan omogućavajući daljnju nadogradnu i prilagodbu sučelja prema specifičnim potrebama korisnika.

Ključne riječi: videozid, igre, aplikacije, igrifikacija, mobilna aplikacija, web aplikacija, websocket

System for support of interactive applications and video games on a video wall

Abstract

This master's thesis introduces the concept of transforming videowalls from their traditional, informational use into user-interactive systems. The thesis proposes a system architecture, alongside the selection and implementation of necessary tools and technologies to realize each component of the interactive interface. A standout feature of this system is the mobile application that serves as a controller for a web application, which displays user interactions on the videowall. The entire system is designed to be modular, facilitating further enhancements and customization of the interface to cater to specific user requirements.

Keywords: videowall, games, gamification, mobile application, web application, websocket