Dizajn i razvoj D&Dash aplikacije u React Native okruženju

Brčić, Antonio

Sveučilište u Rijeci – Odjel za informatiku

Informacijsko-komunikacijski sustavi

Antonio Brčić

Dizajn i razvoj D&Dash aplikacije u React Native okruženju

Diplomski rad

Mentor: izv. prof. dr. sc. Marina Ivašić-Kos

Rijeka, 15.9.2021.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 13.2.2021.

Zadatak za diplomski rad

Pristupnik: **Antonio Brčić**

Naziv diplomskog rada**: Dizajn i razvoj D&Dash aplikacije u React Native okruženju**

Naziv diplomskog rada na eng. jeziku: **Design and development of D&Dash application in React Native framework**

Sadržaj zadatka:

Proučiti tehnologije i alate za razvoj web i mobilnih aplikacija. Odabrati programski jezik u kojem će se razvijati aplikacija i pripremiti i instalirati tehnološki stog te ukratko opisati ključne karakteristike alata koji će se koristiti u praktičnom dijelu rada.

Proučiti igru D&D5e (Dungeons and dragons fifth edition) i osmisliti aplikaciju koja će pomoći igračima da prilikom igranja brže dođu do odgovarajućih informacija o likovima i o pravilima igre.

Izraditi aplikaciju te opisati ključne elemente aplikacije i prikazati odgovarajuće dijelove koda.

Mentor:

Izv. prof. dr. sc. Marina Ivašić-Kos

Voditeljica za diplomske radove:

Izv. prof. dr. sc. Ana Meštrović

Zadatak preuzet: 25.02.2021

(potpis pristupnika)

Antonio Brčić

Sadržaj

Sažetak

Rad se sastoji od teorijskog i praktičnog dijela. U teorijskom dijelu objašnjeni su korišteni alati React Native, Android studio, Expo Go i Visual Studio Code, te D&D Tabletop Roleplaying game. U praktičnom dijelu opisana je implementacija i izrada aplikacije te njene funkcionalnosti.

Ključne riječi: React Native, Cross-platform development, Android studio, Expo Go, D&D Tabletop RPG

Uvod

U ovom diplomskom radu opisan je postupak postavljanja radnog okruženja potrebnog za izradu jednostavne aplikacije koristeći Javascript programski jezik, React Native framework, Android studio emulator za virtualni android device, Expo Go alat za testiranje i distribuciju aplikacije na mobilne uređaje te postavljana radnog okruženja u Visual Studio Code-u.

U zadnjih par godina, na tržište su stupili novi trendovi za web i mobilne aplikacije. PWA (Progresivne web aplikacije) pripomažu izuzetno brzom učitavanju web stranica te podpomažu rad offline aplikacija poput, primjerice Google Mapsa. Starbucks, Uber, Pinterest i Twitter ih također koriste.

Još jedan veoma zanimljiv trend su AIC (Artificial intelligence chatbots) koji se koriste za support. Uz pomoć strojnog učenja, natural language processinga te informacijama danih od korisnika daju nekakvu povratnu informaciju korisnicima koji koriste support (poput FAQ tj. Frequently asked questions).

Osim toga dosta su popularne i metode pretrage putem glasovnog inputa, ili Motion UI, user interface koji je postao dosta popularan zbog svoje unikatnosti gdje korisnicima daje mogućnost micanja, okretanja, odskakivanja elemenata i slično.

Također, u ovom diplomskom sastavljena je komparacija React Nativa sa sličnim frameworkovima te alatima za izradu mobilnih aplikacija [1].

Javascript

Javascript (JS) je lightweight interpretirani ili just-in-time kompajlirani programski jezik. Iako je najpoznatiji kao skriptni jezik za izradu web stranica, koristi se i za mnoga okruženja koja nisu preglednici, poput Node.js, Apache CouchDB i Adobe Acrobat. Javascript programski jezik je prototype-based jezik, stil objektno orijentiranog programiranja u kojem klase nisu eksplicitno definirane, već su izvedene dodavanjem svojstava i metoda u instancu druge klase, ili rječe, dodavanjem u prazan objekt. Također je jezik sa više paradigmi koji je dinamičan te podržava objektno orijentirane i imperativne funkcionalne stilove programiranja (Slika 1) [2].

Vrijedi i napomenuti da Javascript nije „interpretirana Java“. Javascript se bazira na dinamični skriptni jezik koji podržava konstrukciju objekata temeljenih na prototipovima. Osnovna sintaksa je namjerno slična Javi i C++ programskim jezicima kako bi se smanjio broj novih koncepata potrebnih za učenje javascript programskog jezika. Jezične konstrukcije kao što su npr. if, for i while petlja, switch te try...catch blokovi funkcioniraju gotovo identično kao i u Javi i C++ (ili približno).

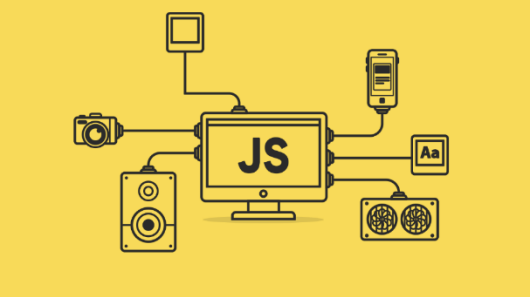
Javascript može funkcionirati i kao proceduralni i kao objektno orijentiran jezik. Objekti se stvaraju programski u Javascriptu, dodavanjem metoda i svojstava inače praznim objektima za vrijeme izvođenja. Za razliku uobičajenih sintaktičkih definicija u klasi u C++ i Java jezicima. Nakon što je objekt izgrađen, može se koristiti kao blueprint (ili prototype) za stvaranje sličnih objekata.

Dinamičke mogućnosti Javascripta uključuju izgradnju objekata tijekom izvođenja programskog koda, popisivanje varijabli parametara, varijable funkcija, kreiranje dinamičke skripte, introspekciju objekata te oporavak izvornog koda. Javascript programi mogu dekompajlirati tijela funkcija natrag u njihov izvorni tekst.

Prvu verziju JavaScripta izumio je Brendan Eich, 1995. kao način dodavanja programa na web stranice u Netscape Navigator pregledniku, a od 1996. usvajaju ga i podržavaju svi Netscape i Microsoft web preglednici.

Izvorno se trebao zvati LiveScript, ali da bi se potakla uporaba novog skriptnog jezika, nazvan je slično jeziku Java, od kojeg se u tadašnje vrijeme dosta očekivalo. Nakon usvajanja jezika izvan Netscapea, osmišljena je standardizacija koja opisuje način na koji jezik JavaScript treba raditi tako da različiti softveri podržavaju taj isti programski jezik. Ovo se naziva ECMAScript standardom, prema Ecma International organizaciji koja je zaslužna za standardizaciju. Do sada je objavljeno pet inačica standarda ECMA-262 [3].

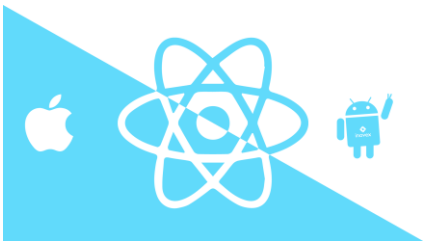
Ono što nam JavaScript kao skriptni jezik omogućava jest pisanje jednostavnih programa i njihovo ugrađivanje izravno u HTML dokument. Omogućuje dodavanje dinamičkog teksta HTML dokumentu, reagira na događaje (event based programming) tj. može se podesiti tako da je izvršavanje skripte povezano s nekim događajem. JavaScript može čitati i mijenjati sadržaj HTML elementa, a može i prepoznavati web preglednik krajnjeg korisnika tako omogućava da taj krajnji korisnik vidi dokument kreiran upravo za web preglednik koji on koristi. Ideja iza dizajna JavaScripta jest da programiranje učini lakšim za početnike. Uglavnom to čini pronalaženje problema u svojim programima teže jer ih sustav neće pokazati, međutim, ta fleksibilnost također ima svoje prednosti. Ostavlja prostor za puno tehnika koje su nemoguće na ostalim strožim jezicima.



Slika 1 Javascript

React native

Već nekoliko godina, React Native (također poznat i kao RN) koji se temelji na JavaScript programskom jeziku je popularan framework u svijetu mobilnog razvoja jer omogućuje razvijanje mobilnih aplikacija za iOS i Android istovremeno. Framework također omogućuje kreiranje aplikacija za različite platforme koristeći isti codebase (Slika 2). React Native su uspješno usvojile stotine tvrtki širom svijeta, uključujući Uber, Microsoft i Facebook, a koristi se i učitavom nizu industrija[4].



Slika 2 React Native- JavaScript framework za mobilne aplikacije koji omogućuje izradu izvornih mobilnih aplikacija za iOS i Android.

Povijest React Native-a

React Native je po prvi put kao projekt otvorenog koda objavio Facebook, 2015. godine. U samo nekoliko godina postao je jedno od najboljih rješenja za razvoj aplikacija u mobile developmentu. React Native razvoj koristi se za pokretanje nekih od vodećih svjetskih mobilnih aplikacija, uključujući Instagram, Facebook i Skype.

Razlozi zašto je React Native postao ekstremno popularan i uspješan su vrlo jednostavni, tvrtke mogu stvoriti kod jednom i koristiti ga za iOS i Android development istovremeno, što dovodi do značajne uštede vremena i resursa potrebnih za razvoj nekog projekta koji se zasniva na bazi mobilne aplikacije. React Native je također izgrađen na temelju React-a, JavaScript biblioteke koja je već bila iznimno popularna od kada je framework objavljen. Osnažio je razvojne programe na sučeljima koji su prije mogli raditi samo sa web-tehnologijama, te omogućio stvaranje robusnih mobilnih aplikacija spremnih za proizvodnu za mobilne platforme. Zanimljivo je i napomenuti da je React Native kao i kod mnogo revolucionarnih izuma, nastao kao odgovor na tehnološku grešku.

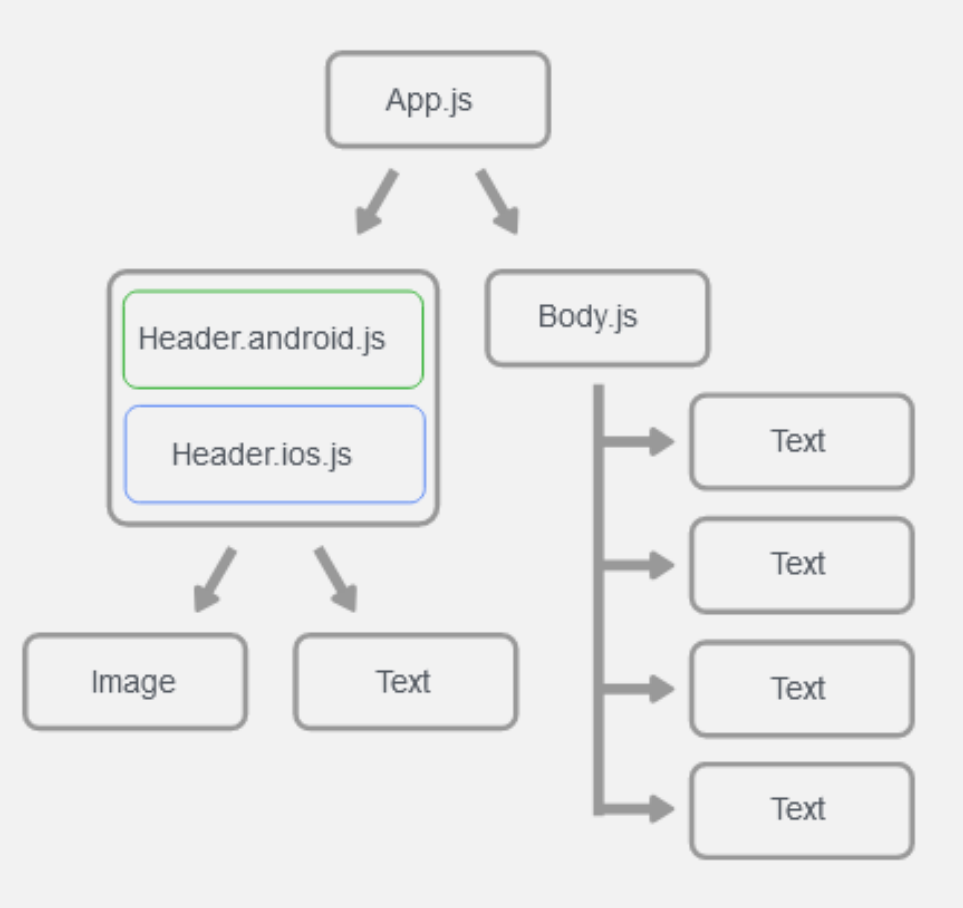
Kada je Facebook po prvi puta odlučio učiniti svoju uslugu dostupnom na mobilnim uređajima, umjesto izgradbe izvorne aplikacije poput mnogih vrhunskih tehnoloških konkurencija u to vrijeme, odlučili su pokrenuti mobilnu web stranicu baziranu i temeljenu na HTML5. Međutim, rješenje nije izdržalo test vremena, tj ostavilo je mnogo prostora za korisničko sučelje i poboljšanja performansi. 2012. godine Mark Zuckerberg je izjavio: „Najveća pogreška koju smo napravili kao tvrtka, bazirali smo se previše na HTML code za razliku od izvornog koda.“

Ubrzo nakon toga, 2013. godine, razvojni programer Facebooka Jordan Walke predstavio je metodu za generiranje elemenata korisničkog sučelja za iOS aplikacije korištenjem JavaScripta. Na toj ideji je organiziran Hackaton kako bi se otkrilo koliko koje su mogućnosti razvoja mobilnih uređaja korištenjem JavaScripta .

Tako je zaživio React Native. U početku razvijen samo za iOS, Facebook ga je ubrzo nadogradio podrškom za Android i prije nego što je framework postao dostupan 2015. godine. Tri godine kasnije, React Native postao je drugi najveći projekt na GitHubu, mjerenom po brojem suradnika. U 2019. godini bio je snažan i zauzeo šesto mjesto sa više od 9100 suradnika [4].

Slika 3 prikazuje koncept rada React Native-a. Aplikacija se kodira u App.js ili povlači komponente iz drugih skripti. Zatim se kompajliranjem dijeli na Body.js i

Header. React Native samostalno određuje koristi li Header.android.js ili Header.ios.js ovisno o kojem je uređaju riječ, te zatim renderira tekst ili slike iz App.js u uređaj.



Slika 3 React Native koncept

React vs React Native

Razlike između Reacta i React Nativa su vrlo male. Može se reći da je React Native nije „novija“ verzija React-a, iako ga React Native ipak koristi. React (također poznat kao ReactJS) je JavaScript biblioteka koja se koristi za izradu prednjeg dijela web stranice. Slično kao i React Native, razvio ga je inženjerski tim Facebooka. U međuvremenu, React Native, kojeg pokreće React, omogućuje programerima da koriste skup komponenti korisničkog sučelja za brzo sastavljanje i pokretanje iOS i Android aplikacija. I React i React Native koriste mješavinu Javascript-a i posebni markup language, JSX. Međutim, sintaksa koja se koristi za renderiranje komponenti u JSX razlikuje se između Reacta i React Nativa. React koristi neke HTML i CSS elemente (<h1>, <p> i slično), dokle React Native dopušta upotrebnu izvornih elemenata mobilnog korisničkog sučelja (<View>, <Text>, <Image>, <ScrollView> itd.). Iako su oba frameworka međusobno povezana, koriste se u različite svrhe. Poznavanje React-a nije dovoljno za razvoj mobilnih aplikacija za iOS i Android [5].

Cross-Platform Development

Cross-Platform Development praksa je izgradnje softvera kompatibilnog s više vrsta hardverske platforme. Aplikacija za više platformi može raditi na više sustava, primjerice na Microsoft Windows, Linux i MacOS. Jedan od primjera aplikacije za više platformi je web preglednik ili Adobe Flash koji radi isto, bez obzira na računalo ili mobilni uređaj koji ga pokreće.

Cross-Platformom se smatra idealnim rješenjem u domeni razvijanja softvera. Baza koda se jednom izgradi i pokreće na bilo kojoj platformi, za razliku od softvera izgrađenog primarno za jednu od izvornih platformi. Programeri mogu upotrijebiti alate i jezike s kojima su već upoznati, poput Javascripta i C# za izgradnju platformi koje su im nepoznate. Vlasnici nekih softvera su također zainteresirani jer se vrijeme i trošak za razvoj aplikacija znatno smanjuje kada je u pitanju Cross-Platform razvoj.

Postoje mnogi razlozi zašto se vlasnici softvera opredjeljuju za Cross-Platform Development, u ovom slučaju React Native. Jedan od njih je šira publika. Developeri se ne fokusiraju na jednu ciljanu publiku, primerice korisnike Android vs korisnike iOS-a jer sotfver radi na obje platforme, što daje veći pristup široj korisničkoj bazi.

Postoje neke navigacijske i dizajnerske razlike između iOS-a i Androida koji se u Cross-Platform Developmentu rješavaju prema zadanim postavkama zahvaljujući zajedničkoj bazi kodova, čime se dovodi do konzistentnosti platforme i olakšanog brendinga aplikacije.[8]

Jedna od najvećih prednosti Cross-Platform Developmenta je što se može izgraditi samo jedan codebase za Android i iOS istovremeno, te na taj način ostvara „Reusable code“ (kod za višestruku upotrebu). Za razvoj izvornih aplikacija potrebno je zasebno pisanje koda i često su potrebna dva različita programera, jedan za iOS i jedan za Android.

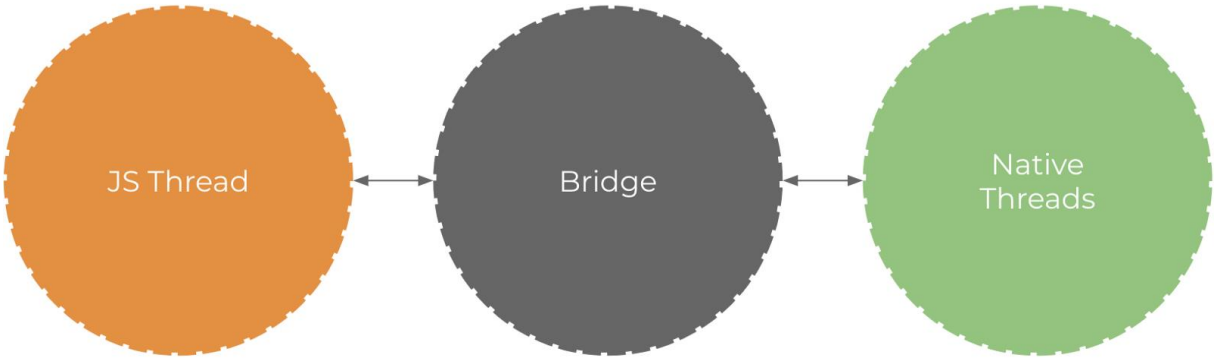
Budući da je za upravljanjem iOS-om i Androidom potrebna samo jedna baza kodova koristeći React Native, sve se nalazi na jednom mjestu, što dozvoljava znatno brži razvoj proizvoda. Cross-Platform aplikacije su izrađene kao pojedinačni projekti iako podržavaju različite uređaje a velika količina koda može se ponovno koristiti između platformi.

Smanjenje troška je također značajna uloga. Izgradnja aplikacije pomoću Cross-Platform razvoja može biti 30% jeftinija od izrade izvornih aplikacija zahvaljujući mogućnosti ponovne uporabe koda i smanjenom vremenu razvijanja, što dovodi do re-alokacije resursa i programera na jedan projekt umjesto na zasebne iOS i Android projekte koji zahtijevaju znatno veću alokaciju resursa i programera.

Funkcije React Nativa

Kada pričamo o Cross-Platform developmentu, vrijedno je spomenuti i na koji način React Native radi, te spomenuti kako se razlikuje. React native je mješavina Javascripta i JXL-a, posebnog koda za označavanje (markup) u XML-u. Framework ima mogućnost komunikacije s oba područja, JavaScript based threadovima, i existent native app threadovima.[9][10]

React Native koristi takozvani „Bridge“ ili „Most“, što upućuje da iako su JavaScript i Native napisani sasvim različitim jezicima, značajka premošćivanja omogućuje dvosmjernu komunikaciju, što znači da ako već imate postojeću aplikaciju u iOS-u ili Androidu, React Native dozvoljava prebacivanje komponenti u React Native development (Slika 4).



Slika 4 React Native Bridge

Android Studio

Android Studio je službeni Integrated Development Environment (IDE) koji se koristi za razvoj primarno Android aplikacija, a temelji se na IntelliJ IDEA. Osim odličnog uređivača koda i alata za razvojne programere, Android Studio nudi još značajki koje povećavaju produktivnost pri izradi Android aplikacija.

Nudi fleksibilan build-up sistem baziran na Gradle-u, brzi i bogat značajkama emulator za Android i iOS, jedinstveno okruženje u kojem se može razvijati aplikacije za gotovo sve Android uređaje, real-time programiranje gdje se promjene izvršavaju direktno na uređaju bez potrebe ponovnog pokretanja koda, GitHub integraciju za pomoć pri gradnji bazičnih funkcija aplikacije te extenzivne alate za testiranje i frameworkove [11].

Uz to podržava C++ i NDK, visoko je optimiziran te ima ugrađenu podršku za Google Cloud što pomaže pri olakšavanju integracije Google Cloud Messagging-a (GCM) i Google App Engine-a (GAE) [11].

GCM je mobilna usluga obavijesti koju je Google razvio isključivo kao pomoć progamerima. Omogućuje programerima da putem third-party aplikacija pošalju podatke, obavijesti ili informacije s poslužitelja. 2019. godine zamjenjuje ga novi software pod imenom Google Firebase Cloud Messaging (FCM) [12].

GAE je platforma za računalstvo u oblaku kao usluga za razvoj i hosting web aplikacija u podatkovnim centrima kojima upravlja Google. One su zaštićene i rade na više poslužitelja. Nudi automatsko skaliranje za web aplikacije te dodijeljuje resurse web aplikaciji. Podržava aplikacije poput Go, PHP, Java, Python, Node.js, .NET i Ruby. Usluga je besplatna do određene potrošnje resursa.[13]

U ovom radu Android studio nije fokus, već se koristi za virtualni emulator android uređaja za potrebe testiranja i pokretanja aplikacije.

Expo Go

Expo je open source framework i platforma za univerzalne React aplikacije. Sastoji se od skupa alata i usluga izgrađenih oko React Native i izvornih platformi koje pomažu pri razvoju, izgradnji, implementaciji te brzoj iteraciji na iOS, Android i web aplikacijama iz iste JavaScript/TpyeScript baze koda.

Expo Go je u ovom slučaju korišten kao premosnica između Visual Studio Code radnog okruženja koje koristi React Native i Android Studia koji se koristi za Android emulator za Android Pixel API 29 za testiranje aplikacije, također moguće je preko Expo Go-a preuzeti aplikaciju Expo Go sa play store-a ili apple store-a i bootati aplikaciju na realnom uređaju koristeći Expo Go server na lokalnom serveru (local host).

Slika 5 Expo go

Visual Studio Code

Za izradu projekta korišten je Visual Studio Code, open source code editor koji sadrži alate za razvojne pogramere IntelliSense code completion i debugging. Sadrži brzi source code editor koji je savršen za svakodnevnu upotrebu, podršku za stotine jezika, isticanjem sintakse, podudaranjem zagrada, automatskim uvlačenjem i box selectionom, snippetima itd.

Ima intuitivne shortcutove na tipkovnici koje doprinosi zajednica, te nam omogućuju lagano upravljanje kodom. IntelliSense je ugrađena podrška za dovršavanje koda, koje koristi bogato razumijevanje semantičkog koda, navigaciju i preradu koda.

Uz odlični alat za debugging dozvoljava korisnicima navigaciju kroz svake iteracije izvršavanja koda te pregledavanjei varijabli i poziva funkcija i naredba u konzoli. Uz sve to, također sadrži i podršku za Git.

Slika 6 VS Code

Izrada D&D Aplikacije

Cilj ovog rada bila je izrada jednostavne aplikacije koja će olakšati igranje igre D&D5e (Dungeons and dragons fifth edition) TTRPG (Tabletop role-playing games) na način da se sve bitne informacije za igrače pohranjujemo u aplikaciji.

Za razvoj aplikacije korišteni su odgovarajući alati koji se danas preporučuju za razvoj mobilnih aplikacije kako bi se postigao brz cross-plaform razvoj mobilne aplikacije s mogućnošću jednostavnog proširivanja i nadogradnje.

Što je D&D TTRPG?

Slika 7 D&D Logo

Dungeons & Dragons (skraćeno D&D ili DnD) je fantasy stolna igra uloga (RPG – Role playing game), izvorno dizajnirana od strane Gary Gygax-a i Dave-a Arneson-a.[14] Prva edicija objavljena je 1974. godine u časopisu Tactical Studies Rules Inc. (TSR). Objavljuje ga Wizards of the Coast (sada podružnica Hasbro-a) a od 1997. godine igra je izvedena iz minijaturnih ratnih igara, s varijacijom igre Chainmail iz 1971. godine koja je poslužila kao početni sustav pravila. Publikacija D&D-a općenito je prepoznata kao početak modernih igara uloga i industrije igara.

D&D odstupa od tradicionalnog wargaminga poput Warhammer 40k [15], dopuštajući svakom igraču da stvori vlastitog lika za igru umjesto vojne formacije. Likovi se zatim upuštaju u zamišljene avanture unutar nekog fantasy okruženja. Dungeon Master (DM) služi kao „sudac“ i pripovijedač priče igre, opisivajući okruženje u kojem se događaju avanture i igrajući ulogu NPC-a (non-player characters). Likovi najčešće čine stanku između zadataka i borbi kako bi komunicirali sa stanovnicima okruženja i međusobno te zajedno rješavaju dileme, upuštaju se u fantasy bitke te istražuju, prikupljaju blago i znanje, iskustvene bodove (experience points, XP) kako bi povećali razinu svojega lika te postali moćniji u nizu zasebnih sesija igara.

WotC (Wizards of the Coast) je 2014. godine izdao petu ediciju igre sa potpuno novim i pojednostavljenim setom pravila, podijeljenim u različite knjige. Jedna od tih knjiga, Player's Handbook sadrži ogromnu količinu informacija i mogućnosti izrade likova.

Igrači svoje likove kreiraju zajedno sa Dungeon Masterom koristeći knjigu koja

sadrži osnovna pravila za igrače tj. Player's Handbook. Igrač mora odabrati svoju rasu primjerice orc, elf, dwarf, human itd. od kojih svaka ima svoje određene bonuse. Također mora odabrati svoju klasu, tj. u čemu je lik specijaliziran, primjerice barbarian, fighter, monk ako je lik osmišljen kao nekakav borac ili ratnik, wizard, sorcerer, warlock ako je lik osmišljen kao nekakav čarobnjak i slično. Također odabiru za likove svoju „priču“ koju osmišljaju zajedno sa Dungeon Masterom, a tu pripomažu tablice sa različitim psihološkim opisima lika. Zatim bacaju kockice kako bi raspodjelili atribute likovima. Postoje šest atributa koji svaki ima svoju ulogu u igri a one se djele na tri fizička atributa i tri psihička atributa. Primjerice, tri fizička atributa su strength (snaga) se koristi za udaranje sa oružjima ili podizanje, dexterity (spretnost) za šuljanje ili pucanje iz oružja, constitution (tjelesna konstitucija) za životne bodove. Intelligence (inteligencija), wisdom (mudrost), charisma (karizma) su psihički atributi, klase koje koriste magiju ih koriste kao katalizator magije. Inteligencija se također koristi za razumijevanje nekakvih zagonetki, mudrost za prostornu svijest, a karizma za socijalne interakcije.

Kada su igrači raspodjelili sve svoje atribute i napravili svoje likove, igra započinje. Dungeon Master predstavlja priču i opisuje okruženje, mirise, sliku i zvukove na lokaciji gdje se likovi nalaze, te pita igrače što njihovi likovi rade. Likovi tj. igrači zatim pričaju međusobno i donose odluke što će raditi kao grupa. Hoće li otići u tamnicu, socijalizirati i slično, te Dungeon Master ponovno opisuje i postavlja okruženje. Prosječna sesija igranja traje oko četiri sata, a prosječna kampanja (dugoročno igranje) nema određeni broj sesija.

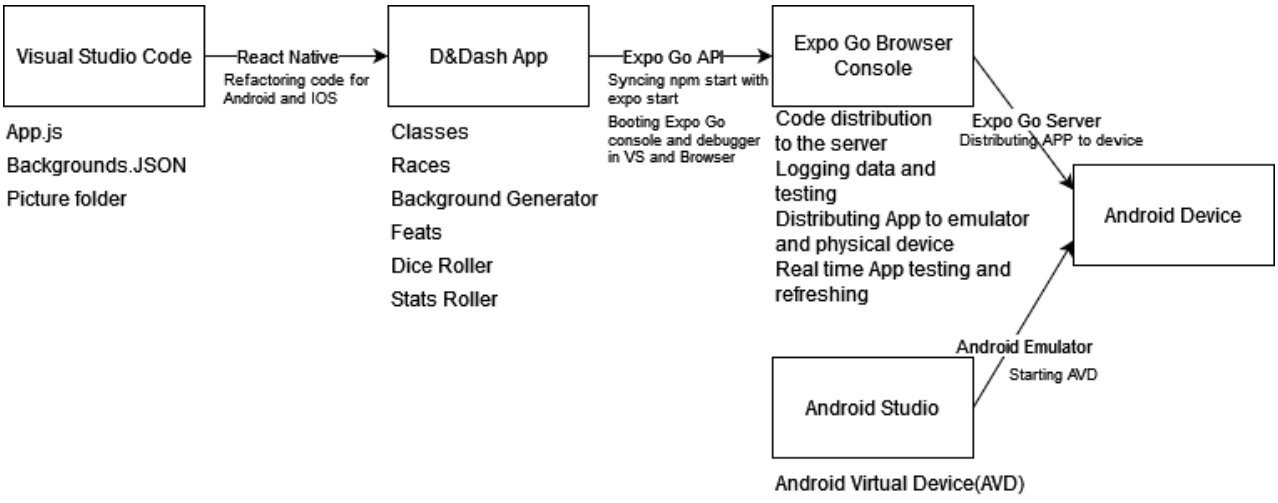
Taj proces izrade lika traje, te je tako nastala ideja za ovu aplikaciju, svrstati, podijeliti i skratiti te pojasniti proces izrade likova novijim ali i iskusnijim igračima te ju digitalizirati na mobilni uređaj za brzu i jednostavnu upotrebu.

Postavljanje radnog okruženja i arhitektura

Pri izradi aplikacije najbitnije je ustanoviti za što se koji alat koristi te dizajnirati arhitekturu aplikacije i postaviti radno okruženje. Za izradu aplikacijskog koda, korišten je Visual studio code, u kojemu se nalazi programski kod aplikacije App.js, Backgrounds.JSON koji se koristi za Background Generator te folder sa Slikama koje se koriste u aplikaciji u Classes i Races ekranu. Putem Visual Studio Code-a, React Native refactorira programski kod za Android i iOS uređaje te pokreće D&Dash aplikaciju koja ima programski kod za ekrane Classes, Races, Background Generator, Feats, Dice Rolelr i Stats Roller.

Expo Go API sinkronizira npm start sa expo start komandnim linijama, te pokreće Expo Go konzolu i debugger za Visual Studio Code i preglednik po izboru (defaultni preglednik sustava). Expo Go preglednik konzola se koristi za distribuiranje koda na Expo Go servere, upisivanje podataka i testiranje istih, distribuciju aplikacije na emulator ili fizički uređaj te testiranje i osvježavanje aplikacije u stvarnom vremenu što pripomaže pri testiranju i debugiranju.

Android Studio nudi opciju AVD, tj. Android Virtual Device emulatora za testiranje te pomoću njegovog API-ja pokreće AVD na kojem se distribuira aplikacija (specifično kodirana za Android zbog React Native API-ja). Također, moguće je aplikaciju testirati i pokrenuti na fizičkom uređaju korištenjem QR koda ili drugih metoda koje Expo Go nudi za testiranje na fizičkom uređaju (slika 8).



Slika 8 - Arhitektura Aplikacije D&Dash

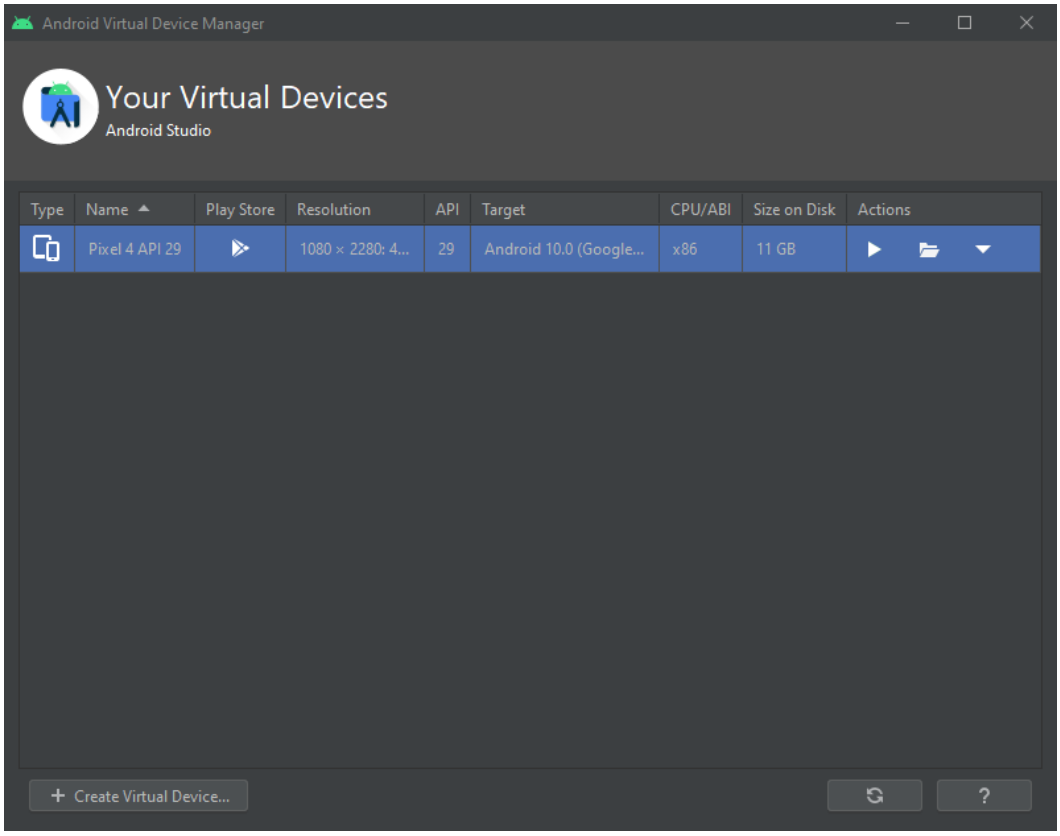
Za izradu ove aplikacije potrebno je bilo postaviti radno okruženje. U Visual studio codeu instalirane su ekstenzije za React Native, code snippeti za React Native te Prettier – alat za uređivanje i formatiranje koda. Za virtualni emulator korišten je Android studio, a za premosnicu Expo Go. Za emulator korišten je Pixel 4 API 29. Na slici 9 možemo vidjeti sve korištene biblioteke.

```
import * as React from 'react';
import { Button, View, Text, Image, ScrollView, SafeAreaView, StyleSheet, StatusBar, TouchableOpacity, Pressable, ImageBackground } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
```

Slika 9 Libraries

Za potrebe projekta korištene su biblioteke NavigationContainer, createNativeStackNavigator, React, te Button, View, Text, Image, Scrollview, SafeAreaView, StyleSheet, StatusBar, TouchableOpacity, Pressable, Image Background.

Za emulator android uređaja korišten je android studio, te pixel 4 API 29 (slika 10).



Slika 10 Android Studio Emulator

Za pokretanje aplikacije korišten je paket Expo Go. Na slici 11 prikazane su naredbe za inicijalizaciju Expo Go paketa.

```
# Create a project named my-app. Select the "blank" template when prompted
$ expo init my-app
# Navigate to the project directory
$ cd my-app
```

Slika 11 Expo go instalacija

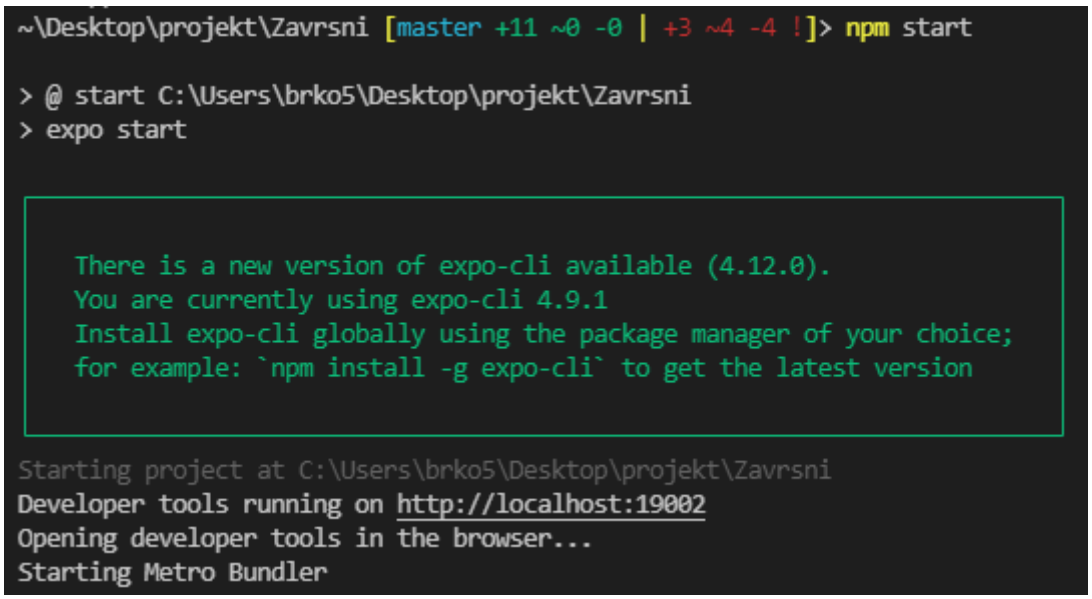
Aplikaciju pokrećemo sa npm start funkcijom, koja pokreće Expo go Development server koristeći naredbu expo start sa danim parametrima. To nam dozvoljava da možemo pokretati aplikaciju putem npm start, a expo go će ju interpretirati kao da je pokrenuta putem naredbe expo start.

package.json > ...

```
{
  "main": "node_modules/expo/AppEntry.js",
  ▷ Debug
  "scripts": {
    "start": "expo start",
    "android": "expo start --android",
    "ios": "expo start --ios",
    "web": "expo start --web",
    "eject": "expo eject"
  },
  "dependencies": {
    "@react-navigation/native": "^6.0.1",
    "@react-navigation/native-stack": "^6.0.2",
    "expo": "~42.0.1",
    "expo-constants": "~11.0.1",
    "expo-status-bar": "~1.0.4",
    "react": "16.13.1",
    "react-dom": "16.13.1",
    "react-native": "https://github.com/expo/react-native/archive/sdk-42.0.0.tar.gz",
    "react-native-safe-area-context": "^3.2.0",
    "react-native-screens": "^3.4.0",
    "react-native-web": "~0.13.12",
    "react-navigation": "^4.4.4",
    "react-navigation-stack": "^2.10.4"
  },
  "devDependencies": {
    "@babel/core": "^7.9.0"
  },
  "private": true
}
```

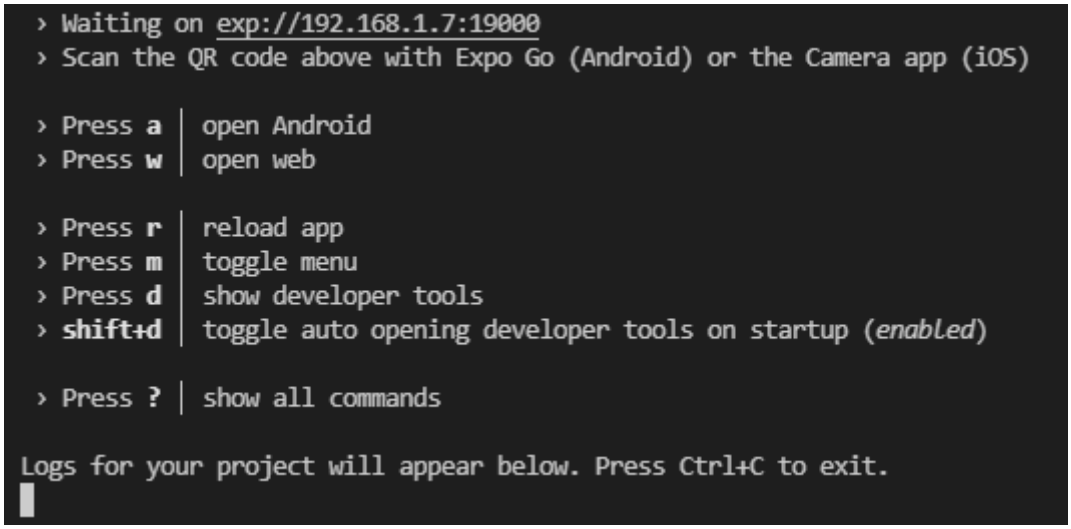
Slika 12 Expo go server

Na slici 13 možemo vidjeti pokretanje putem npm start, gdje terminal pokreće expo start funkciju. Također nam otvara local host server na kojemu se prate sve izmjene aplikacije u realnom vremenu te je odličan tool za debugganje.



Slika 13 Pokretanje

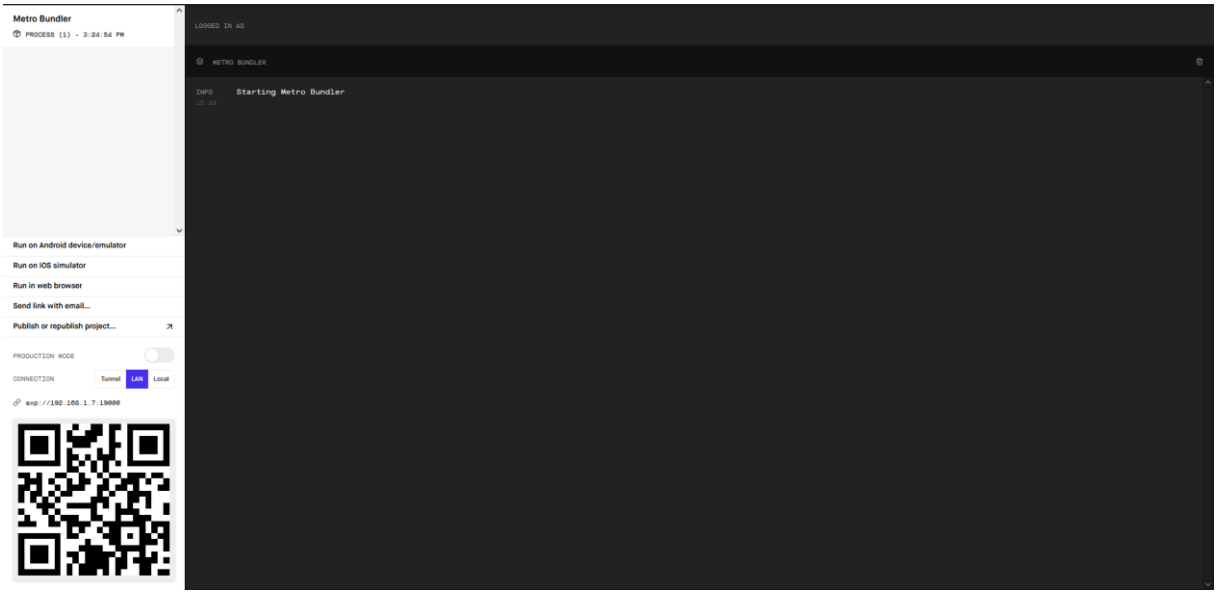
Na slici 14 možemo vidjeti meni koji Expo go nudi. Ako pretisnemo neke od tipki u konzoli, automatski će se pokrenuti Expo go skripta ovisno o inputu. Primjerice, „a“ otvara aplikaciju na android uređaju (ili emulatoru) itd.



Slika 14 Expo Go shortcuts

Zatim, nakon pokretanja radnog okruženja, Expo Go napravi premosnicu između local host servera i aplikacije te izbaci radno sučelje u odabranom browseru i da povratnu informaciju u Visual Studio Code terminalu nudeći shortcute za brzinski startup aplikacije.

Na slici 15 možemo vidjeti kako izgleda sučelje Expo Go local hosta u terminalu koji je učitan u browseru. Nudi nam slične opcije kao u terminalu u visual studiu, poput „Run on Android device/Emulator“ ali također i „Publish or republish„ project...“ kako bi aplikacija bila dostupna android uređajima koji nisu povezani na istu internet mrežu.



Slika 15 Expo Go localhost terminal

Aplikaciju možemo pokrenuti stisnući tipku „a“ u Terminalu Visual Studio Code-a, što će, ako imamo pokrenuti emulator, bootati aplikaciju na virutalnom emulatoru od Android Studia. Ukoliko želimo pokrenuti aplikaciju na vlastitom realnom uređaju, preko Expo Go mobilne aplikacije možemo skenirati QR code te će nam se radno okruženje i promjene prikazati na mobilnom uređaju. Expo Go također nudi Publishanje ili Republishanje projekata preko svojih Web Server servisa, što dozvoljava testiranje i aplikacije sa drugih uređaja koji nisu spojeni na localhost mrežu.

Na slici 16 prikazan je frontend aplikacije, tj. početni zaslon. Pozadinska slika je D&D logo, ime aplikacije (ili navigacije u drugim slučajevima) nalazi se u headeru, te su prikazani gumbi za Classes, Races, Backgrounds, Feats, Dice Roller i Stat Roller.

Slika 16 Pokrenuta aplikacija na emulatoru

Dijagram klasa

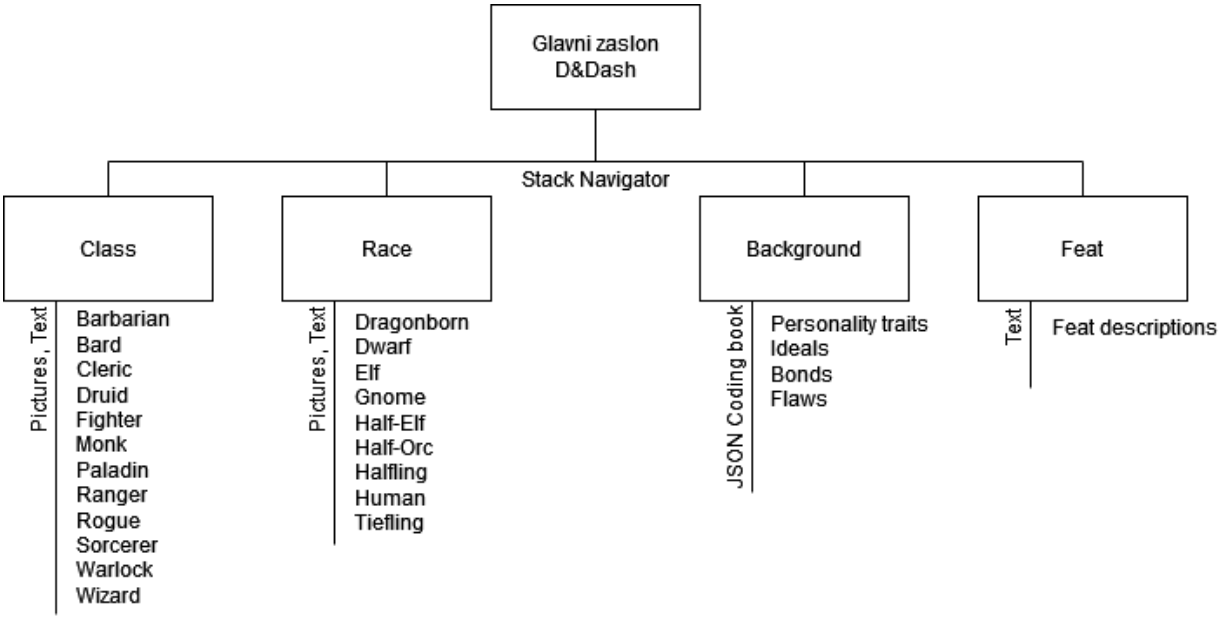
Dijagram klasa prikazuje konceptualno razrađenu aplikaciju. Kao glavnu ideju koristimo Stack Navigator koji mora na stack stavljati određene ekrane i pratiti gdje se korisnik nalazi kada preskače iz ekrana u ekran, i paziti da ako korisnik primjerice uđe u neku klasu ili rasu, da se vrati na meni koji je za to predviđen.

Class je realiziran pomoću formatiranja teksta i korištenja slika, te je veoma sličan interpretaciji u knjizi (Player's handbook).

Za rase je bilo moguće koristiti JSON šifrarnik kao i za klase, ali nijedna stvar se ne pojavljuje više puta, stoga je redundantno koristiti JSON i učitavati ga svaki put kada hoćemo nešto prikazati. Tim načinom štedimo na performansama.

Featovi su isključivo tekstualni te jedina stvar što se ponavlja je ponekad nekakvi prerekviziti koji igrači trebaju zadovoljiti.

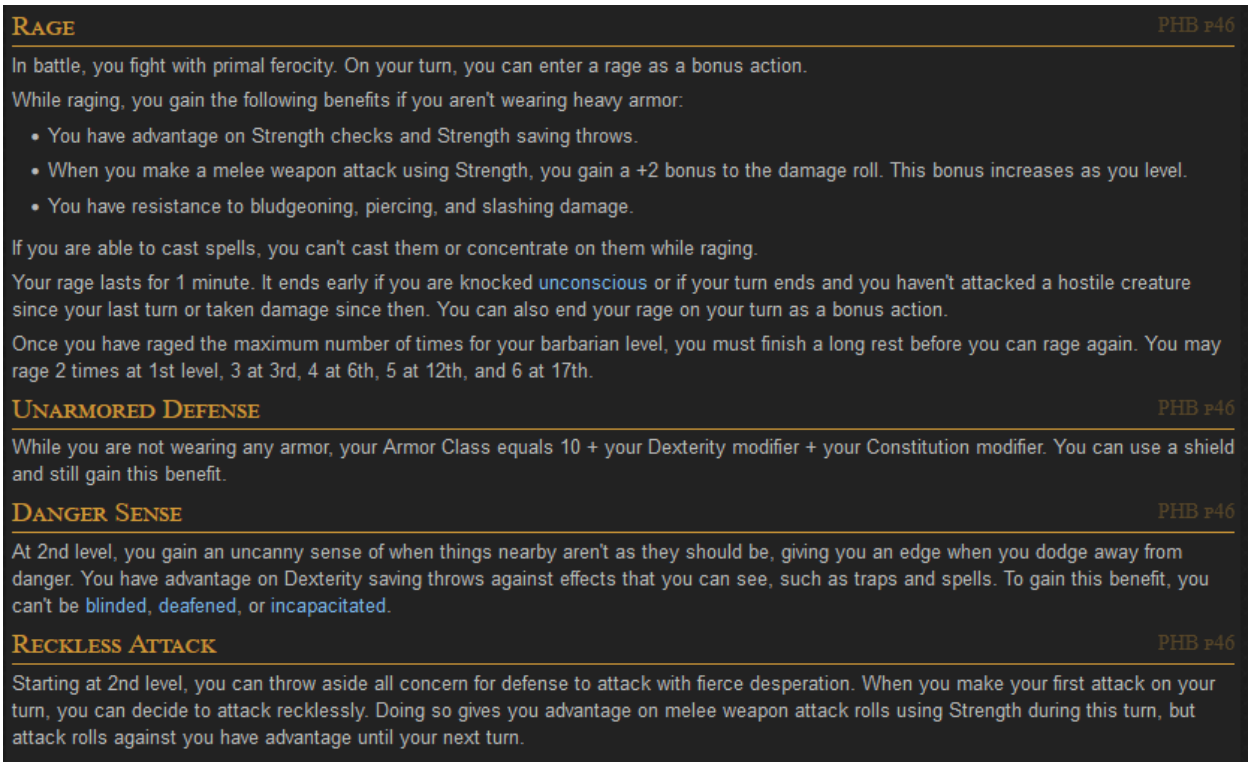
Backgrounds funkcija realizirana je pomoću JSON šifrarnika i UseCase kako bi se uštedilo na performansi (JSON file se reloada samo pritiskom gumba, što dovodi do dinamičkog izmjenjivanja teksta zbog UseCase, inače bi tekst bio cachiran i nebi se mjenjao) te kako bi se prikazali različiti psihološki aspekti lika, što bi igračima pomoglo pri odabiru psihološkog aspekta lika, što pomaže u roleplay-u, tj. igranju uloga (slika 17).



Slika 17 - Dijagram Klasa

Izrada aplikacije D&Dash

U D&D 5e TTRPG-u, pri izradi lika, igrač mora odabrati svoju klasu, rasu, background (povijest lika) ili različite feat-ove (feature), stoga su kategorije podijeljene u četiri različita gumba te zasebno objašnjene na svakoj strani. Sama ideja aplikacije bila je sažeti nešto što izgleda ovako kako je prikazano na slici 18:



Rage PHB p46

In battle, you fight with primal ferocity. On your turn, you can enter a rage as a bonus action.

While raging, you gain the following benefits if you aren't wearing heavy armor:

- You have advantage on Strength checks and Strength saving throws.
- When you make a melee weapon attack using Strength, you gain a +2 bonus to the damage roll. This bonus increases as you level.
- You have resistance to bludgeoning, piercing, and slashing damage.

If you are able to cast spells, you can't cast them or concentrate on them while raging.

Your rage lasts for 1 minute. It ends early if you are knocked unconscious or if your turn ends and you haven't attacked a hostile creature since your last turn or taken damage since then. You can also end your rage on your turn as a bonus action.

Once you have raged the maximum number of times for your barbarian level, you must finish a long rest before you can rage again. You may rage 2 times at 1st level, 3 at 3rd, 4 at 6th, 5 at 12th, and 6 at 17th.

Unarmored Defense PHB p46

While you are not wearing any armor, your Armor Class equals 10 + your Dexterity modifier + your Constitution modifier. You can use a shield and still gain this benefit.

Danger Sense PHB p46

At 2nd level, you gain an uncanny sense of when things nearby aren't as they should be, giving you an edge when you dodge away from danger. You have advantage on Dexterity saving throws against effects that you can see, such as traps and spells. To gain this benefit, you can't be blinded, deafened, or incapacitated.

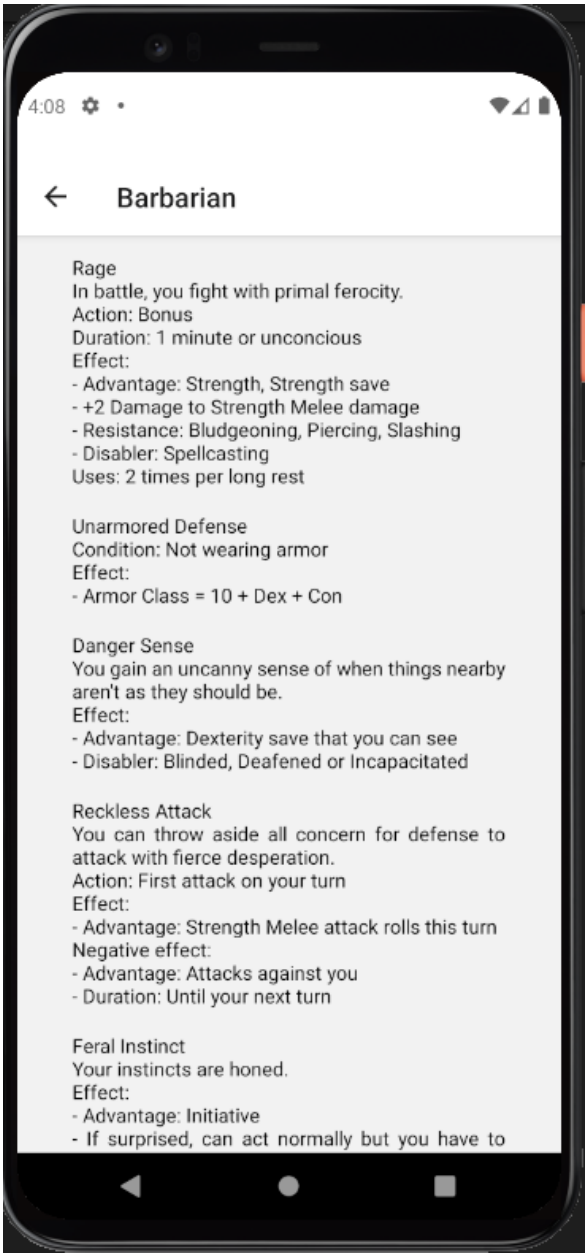
Reckless Attack PHB p46

Starting at 2nd level, you can throw aside all concern for defense to attack with fierce desperation. When you make your first attack on your turn, you can decide to attack recklessly. Doing so gives you advantage on melee weapon attack rolls using Strength during this turn, but attack rolls against you have advantage until your next turn.

Slika 18 Original tekst

U slici 18 prikazan je tekst iz „Player's Handbook“ knjige za igrače D&D-a. Tekst je podosta dug i kompliciran, te je cilj aplikacije bio napraviti tekst koji je lako čitljiv i brzo se pronalazi.

Prerađeni tekst da bude jednostavniji i čitljiviji za nove i iskusne igrače, izgleda ovako kao na slici 19:



Slika 19 Simplified text

Na slici 19 prikazan je Barbarian class tab. Navigacija se dinamički ažurira i prikazuje u headeru, a tekst je sažet i strukturiran na način da je podijeljen u odlomke. Također, cijeli view aplikacije je malo udaljen od rubova kako bi bio lakše čitljiv, te wrappan u scrollview kako bi korisnik aplikacije mogao scrollati i vidjeti opis svih moći jedne klase.

Na slici 20 djelomično je prikazan je kod funkcije Barbarian(). Vidimo da funkcija vraća SafeAreaView u kojem su ugniježđeni scrollview, slika za klasu te formatirani tekst koji se prikazuje.

```
function Barbarian() {
  return (
    <SafeAreaView style={{ flex: 1, alignItems: 'center', justifyContent: 'center', width: '100%' }}>
      <ScrollView
        style={{paddingVertical:10, paddingLeft:40, paddingRight:40}}
        contentContainerStyle={{alignItems:'stretch', justifyContent:'center'}}>
          <Image
            source={require('./app/assets/Classes/Barbarian_Class.png')}
            style={{flex: 1, justifyContent:'center', width:'100%', resizeMode: 'contain'}}
          />
          <Text style={{textAlign:'justify'}}>
            Rage{'\n'}
            In battle, you fight with primal ferocity.{'\n'}
            Action: Bonus{'\n'}
            Duration: 1 minute or unconcious{'\n'}
            Effect:{'\n'}
            - Advantage: Strength, Strength save{'\n'}
            - +2 Damage to Strength Melee damage{'\n'}
            - Resistance: Bludgeoning, Piercing, Slashing{'\n'}
            - Disabler: Spellcasting{'\n'}
            Uses: 2 times per long rest{'\n'}
          </Text>
```

Slika 20 Kod funkcije Barbarian

Na slici 21 možemo vidjeti kod početnog zaslona. Funkciji HomeScreen prosljeđujemo parametar navigacije kako bi aplikacija znala na kojem se ekranu nalazi. Učitana je pozadinska slika sa svojim stilom, te su napravljeni gumbi koji kad su pretisnuti pokreću funkciju navigate() sa danim parametrima za prijelaz na zaslone koje se prosljeđuju kao parametar.

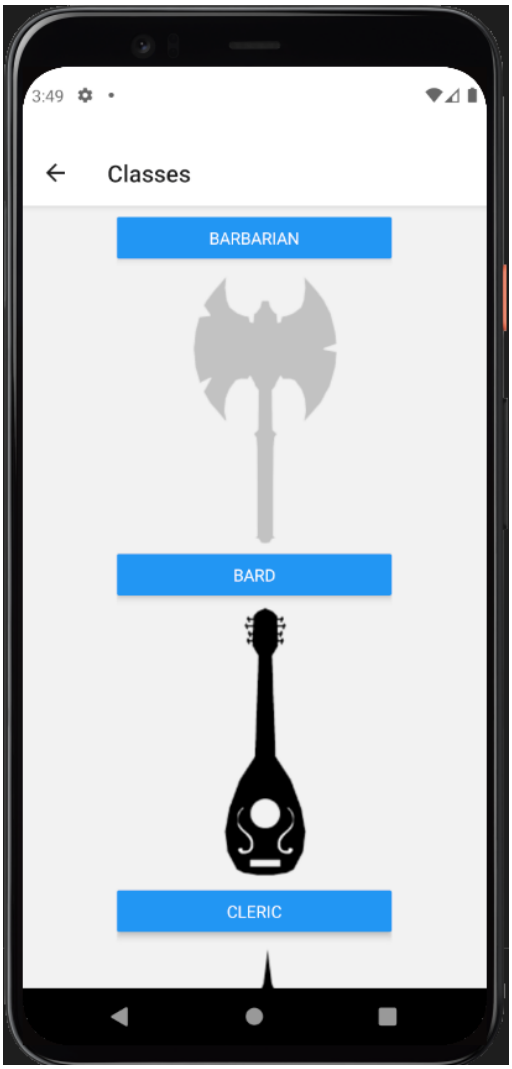
```
function HomeScreen({ navigation }) {
  return(
    <ImageBackground source={require('./app/assets/background.jpg')} style={{flex: 1, resizeMode: 'cover', justifyContent: 'center', width: '100%'}}>
    <SafeAreaView style={{ flex: 1, alignItems: 'center', justifyContent: 'space-evenly', width: '100%', padding:2 }}>

      <Button
          title="Classes"
          onPress={() => navigation.navigate('Classes')}
      />
      <Button
          title="Races"
          onPress={() => navigation.navigate('Races')}
      />
      <Button
          title="Backgrounds"
          onPress={() => navigation.navigate('Backgrounds')}
      />
      <Button
          title="Feats"
          onPress={() => navigation.navigate('Feats')}
      />
      <Button
          title="Dice Roller"
          onPress={() => navigation.navigate('Dice Roller')}
      />
      <Button
          title="Roll Stats for me!"
          onPress={() => navigation.navigate('Stats')}
      />
    </SafeAreaView>
    </ImageBackground>
  );
}
```

Slika 21 Početni zaslon

Za prijelaze između ekrana, napravljen je stack navigator koji se mora proslijediti funkcijama da bi aplikacija znala s kojega ekrana mora na koji ekran ići. Za primjer uzeti ćemo ekran 'Classes'. Klikom na gumb classes prikaže se novi ekran koji sadrži „meni“. Taj meni sadrži sve klase (primjerice Barbarian, Fighter, Druid, Cleric itd.) i njihove vizualne značajke za lakše prepoznavanje koje su sortirane u ScrollView te omogućavaju scrollanje za odabir informacija o klasi. Zatim, klikom na gumb klase Barbarian (gumbi se highlightaju da bi korisnik lakše vidio koji je gumb stisnuo) otvaramo klasu barbarina te dobivamo sve potrebne informacije sortirane i složene u jednostavan tekst format koji je lako čitljiv.

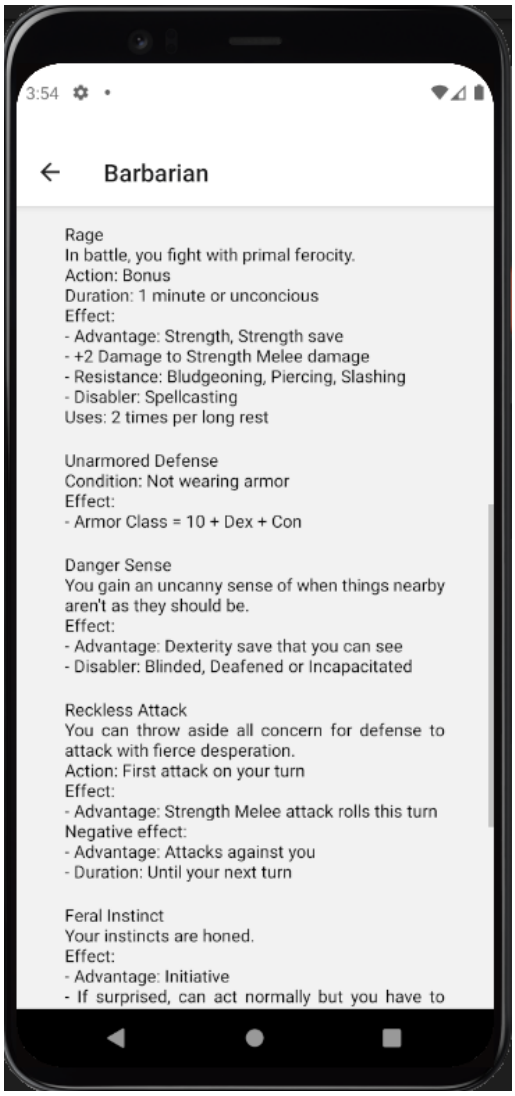
Na slici 22 možemo vidjeti kako izgleda meni u funkciji Classes. Classes je prikazan u headeru kako bi znali u kojem se meniju nalazimo (što je povučeno iz funkcije stack navigatora). Pritiskom na gumb „Barbarian“ ili na sliku sjekire (barbarian icon) korisnik dobije vizualni feedback da je gumb pritisnut, te se na pritisak tog gumba poziva funkcija Barbarian().



Slika 22 Meni, Highlight gumbova

Na slici 23 i 24 možemo vidjeti kako izgleda funkcija Barbarian(). U header je prikazana navigacija, slika uzeta iz Player's handbook knjige, te opisane raznorazne moći koje klasa daje igraču ako odabere Barbarian klasnu specijalizaciju, sažeto i formatirano u ScrollView-u za lakšu preglednost.

Slika 23 Barbarian Klasa 1. dio



Slika 24 Barbarian Klasa 2. dio

U slici 25 prikazan je kod za navigaciju. Koristeći funkciju createNativeStackNavigator(), kreirali smo <NavigationContainer> te na stack dodali inititialRouteName (početni zaslon/ekran) koji u headeru prikazuje „D&Dash“. Također su dodani ekrani za sve klase, rase, background-ove i feat-ove.

```
const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="D&Dash">
          <Stack.Screen name="D&Dash" component={HomeScreen} />
          <Stack.Screen name="Classes" component={Classes} />
          <Stack.Screen name="Races" component={Races} />
          <Stack.Screen name="Backgrounds" component={Backgrounds} />
          <Stack.Screen name="Feats" component={Feats} />
          <Stack.Screen name="Barbarian" component={Barbarian}/>
          <Stack.Screen name="Bard" component={Bard}/>
          <Stack.Screen name="Cleric" component={Cleric}/>
          <Stack.Screen name="Druid" component={Druid}/>
          <Stack.Screen name="Fighter" component={Fighter}/>
          <Stack.Screen name="Monk" component={Monk}/>
          <Stack.Screen name="Paladin" component={Paladin}/>
          <Stack.Screen name="Ranger" component={Ranger}/>
          <Stack.Screen name="Rogue" component={Rogue}/>
          <Stack.Screen name="Sorcerer" component={Sorcerer}/>
          <Stack.Screen name="Warlock" component={Warlock}/>
          <Stack.Screen name="Wizard" component={Wizard}/>
          <Stack.Screen name="Dragonborn" component={Dragonborn}/>
          <Stack.Screen name="Dwarf" component={Dwarf}/>
          <Stack.Screen name="Elf" component={Elf}/>
          <Stack.Screen name="Gnome" component={Gnome}/>
          <Stack.Screen name="Halfelf" component={Halfelf}/>
          <Stack.Screen name="Halforc" component={Halforc}/>
          <Stack.Screen name="Halfling" component={Halfling}/>
          <Stack.Screen name="Human" component={Human}/>
          <Stack.Screen name="Tiefling" component={Tiefling}/>
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

Slika 25 Kod – Navigation stack

Osim toga, jako je bitno specificirati u funkcijama navigatora koja funkcija gdje vodi kako bi stack navigator dodavao i skidao sa stacka točne ekrane. U slici 26 je prikazana funkcija Classes. Proslijeđen joj je parametar navigation (kako bi navigator znao gdje se nalazi), te su gumbovi prikazani koristeći <Button> a ikone koristeći <TouchableOpacity>. Razlika je u tome što je <Button> funkcija fiksna i ne prima style parametar već uzima default stilizaciju ovisno o Android ili IOS platformi.

```
function Classes({navigation}){
  return (
    <SafeAreaView style={{ flex: 1, alignItems: 'center', justifyContent: 'center', width: '100%' }}>
      <ScrollView style={{paddingVertical:10, paddingLeft: 80, paddingRight:80, width:'100%'}}
        contentContainerStyle={{alignItems:'stretch', justifyContent:'center'}}
        >
          <Button
            title="Barbarian"
            onPress={() => navigation.navigate('Barbarian')}
          />
          <TouchableOpacity onPress={() => navigation.navigate('Barbarian')}>
          <Image
            source={require('./app/assets/Barbarian.png')}
          />
          </TouchableOpacity>

          <Button
            title="Bard"
            onPress={() => navigation.navigate('Bard')}
          />
          <TouchableOpacity onPress={() => navigation.navigate('Bard')}>
          <Image
            source={require('./app/assets/Bard.png')}
          />
          </TouchableOpacity>

          <Button
            title="Cleric"
            onPress={() => navigation.navigate('Cleric')}
          />
          <TouchableOpacity onPress={() => navigation.navigate('Cleric')}>
          <Image
            source={require('./app/assets/Cleric.png')}
          />
          </TouchableOpacity>
```

Slika 26 Kod - Navigacije Klase

Usporedba JSON šifrarnika sa <text> parametrom

Do sada su sve funkcije bile izrađene koristeći <text> parametar za klase i rase, zato što su sve moći neke klasne specijalizacije specifične za tu klasu te se ne ponavljaju. Backgrounds se mogu nasumično generirati putem bacanja kockica i mogu se ponavljati, stoga u toj funkciji koristimo JSON šifrarnik umjesto <text> parametara. Bitno je znati da pri izradi aplikacije treba unaprijed pretpostaviti što će se ponavljati a što ostaje statično, tj. pojavljuje se samo jednom. U ovom slučaju klasne specijalizacije i rase i feat-ovi su statični, a background dinamički. Pri izradi pozadinske priče za svog lika (engl. background story, ili backstory) igrač definira psihološku stranu lika, što ponekad može biti teško. Za to postoje tablice u knjizi te bacanjem kockica se odabiru nasumično. To je simulirano u aplikaciji koristeći RNG (random number generator) i JSON šifrarnik koji ovisno o danom broju, pomoću UseCase prosljeđuje iz JSON šifrarnika određeni tekst. Na slici 27 možemo vidjeti isječak JSON šifrarnika, gdje koristi „id“ kao šifrarnik, te parametar text u koji su proslijeđeni stringovi koji opisuju osobine i ličnosti iz knjige.

app > assets > {} backgrounds.json > ...

```json
{
    "Personality_traits":[
        {
            "id": 1,
            "text": "I idolize a particular hero of my faith, and constantly refer to that person's deeds and example."
        },
        {
            "id": 2,
            "text": "I can find common ground between the fiercest enemies, empathizing with them and always working toward peace."
        },
        {
            "id": 3,
            "text": "I see omens in every event and action. The gods try to speak to us, we just need to listen."
        },
        {
            "id": 4,
            "text": "Nothing can shake my optimistic attitude."
        },
        {
            "id": 5,
            "text": "I quote (or misquote) sacred texts and proverbs in almost every situation."
        }
    ]
}
```

Slika 27 JSON Šifrarnik

Na slici 28 prikazana je backgrounds funkcija koja koristi UseCase hookove u React nativeu. Hookovi se koriste za rješavanje širokog raspona nepovezanih problema u Reactu. Korisni su jer se mogu koristiti bez ponovnog pisanja postojećega koda, 100% je backwards compatible. Hooks također pružaju izravniji API za React koncepte (props, state, context itd). U ovom slučaju korišten je {text} hook. Pritiskom na gumb Generate, funkcija setText dobija parametar iz funkcije GenerateBackground te taj parametar sprema u cache. Kasnije možemo taj tekst ponovno koristiti u bilo kojem slučaju i staviti tag {text}. Također je korisno u ovom slučaju jer se dinamički izmjenjuje što nam dozvoljava korištenje JSON filea kao šifrarnika (JSON file se najčešće pohranjuje u cache te izmjene nisu moguće dokle se file ponovno ne pozove, dokle ga useState sam na novo poziva po potrebi, što je idealno za ovaj slučaj).

```
function Backgrounds() {
  const [text,setText] = useState(null)

  return (
    <SafeAreaView style={{ flex: 1, alignItems: 'center', justifyContent: 'center', width: '100%'}}>
          <Button
            title="Generate"
            onPress={() => setText(GenerateBackground())}
          />
          <Text style={{paddingRight: 20, paddingLeft: 20}}>
            {text}
          </Text>
          </SafeAreaView>
  );
}
```

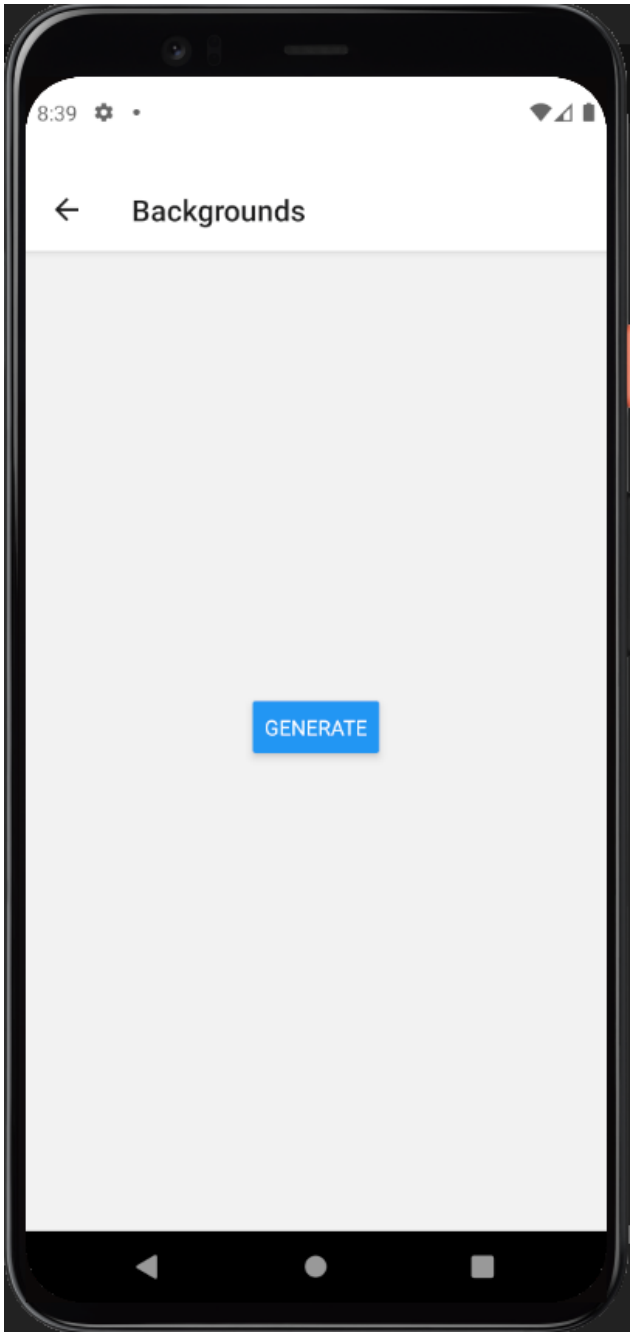
Slika 28 Backgrounds() funkcija

Na slici 29 prikazana je funkcija GenerateBackground(). Koristeći Math.Random generira se broj između 1 i 5 te ovisno o njemu funkcija vraća tekst iz JSON šifrarnika. U ovom slučaju bilo je isključivo potrebno koristiti UseCase jer se inače tekst nebi dinamički osvježavao već bi cache-irao rezultat i konstantno ga prikazivao u zaslonu.

```
function GenerateBackground(){
  var rng = Math.floor(Math.random() * 4);
  //console.log(rng);
  //console.log("background: " , backgrounds?.Personality_traits[rng].text);
  return backgrounds?.Personality_traits[rng].text;
}
```

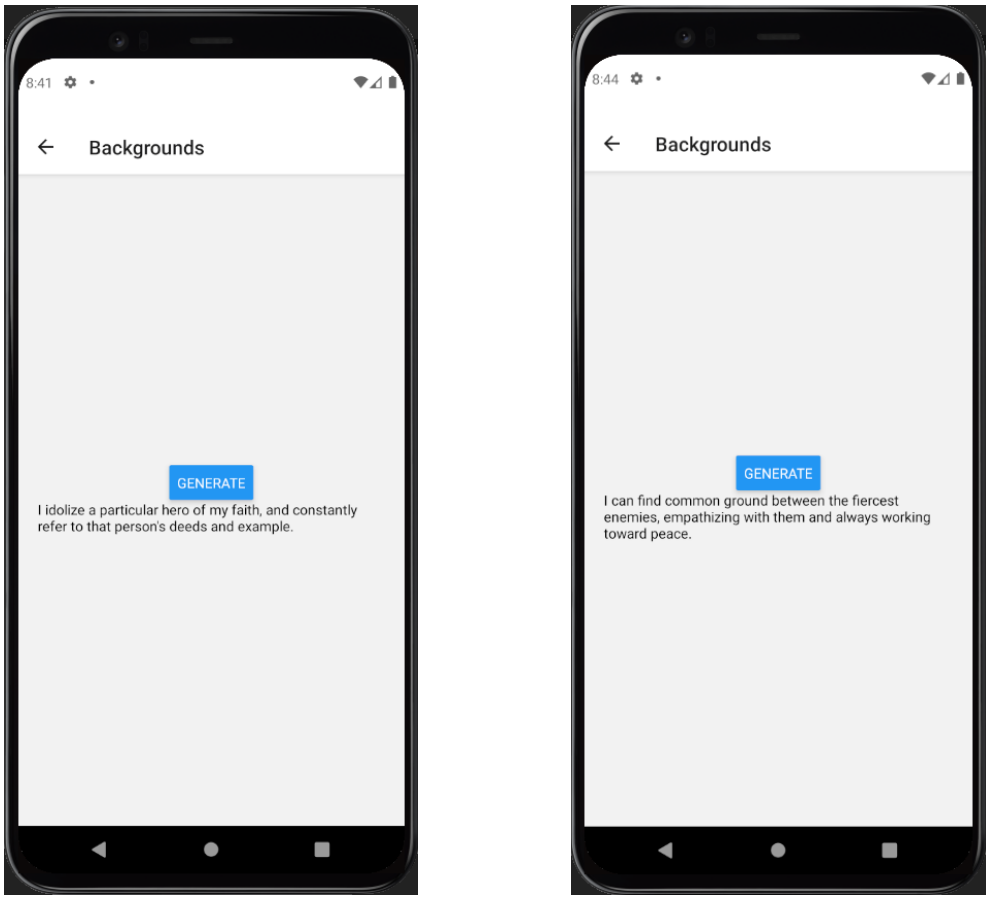
Slika 29 GenerateBackgrounds() funkcija

Na slici 30 možemo vidjeti kako Backgrounds ekran izgleda. Naizgled je prazan, ali pritiskom na gumb „GENERATE“, generirati će se nasumična osobnost koju igrač može iskoristiti za svojeg lika, ili ponovno generirati dokle mu neka opcija ne zvuči zanimljivije.



Slika 30 Izgled Backgrounds ekrana

Na slici 31 možemo vidjeti da se tekst generira ispod gumba, pritiskom na gumb „GENERATE“ zbog UseCase ponovno se dinamički generira broj te JSON file osvježava te odabire nasumični parametar, što je odlično ako želimo konstantno izmjenjivati JSON file dokle radimo na aplikaciji, ali također i da uštedimo na memoriji i vremenu procesiranja.



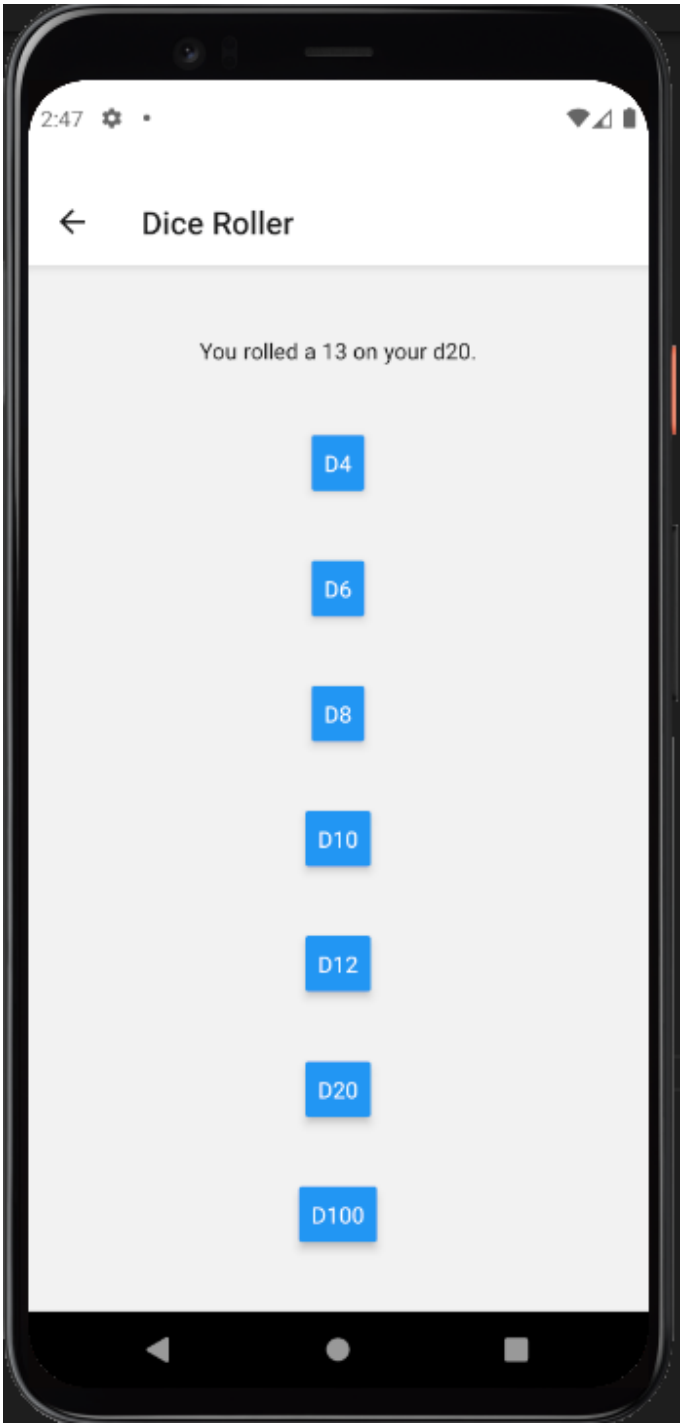
Slika 31 Generacija Backgrounda

Dice Roller i Stat Roller

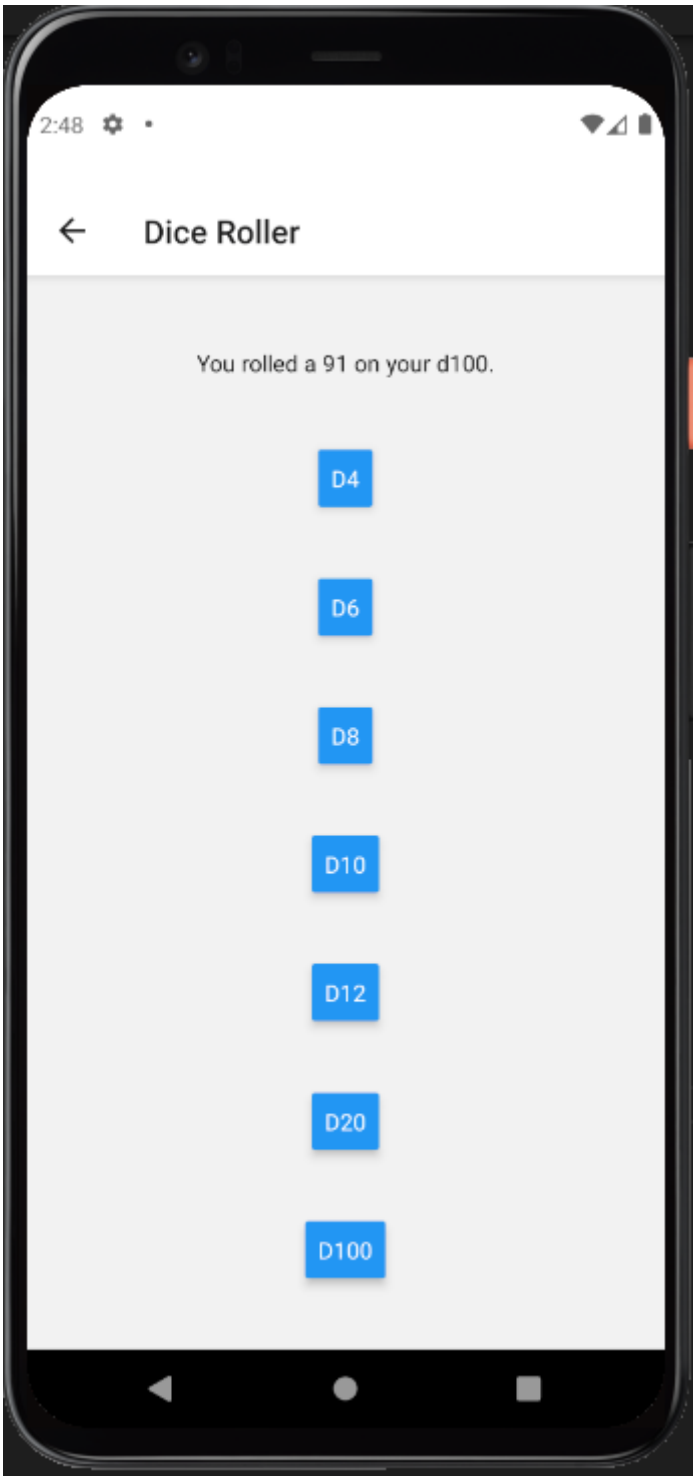
D&D kao tabletop igra uključuje i različite borbe ili nekakve prepreke i zadatke koje DM predstavlja svojim igračima. Za igru se koristi set od sedam „kockica“ sa različitim brojem strana a one se označavaju kao d4, d6, d8, d12, d10, d20 i d100. Oznaka d dolazi od englekog naziva za kockice, tj. die ili u množini dice, a broj označava broj stranica koju „kockica“ ima, tj. moguće brojeve. Primjerice d6, poznatija kao kocka od šest strana (geometrijski oblik kocke) ima šest strana sa brojevima od 1 do 6. Kockice su napravljene po Platonovim poliedrima tj. mnogokutima koji su ravninski geometrijski likovi kojima su svi kutovi i bridovi sukladni. Ponekad igrači zaborave ili nemaju kockice, stoga nastaje ideja izrade Dice Rollera, tj. dio programskog koda koji generira rezultate bacanja kocke.

Slika 32 - Set kockica za D&D

Na slikama 33 i 34 prikazan je izgled bacača kockica gdje se igraču nudi brzinski bacač kockica. Funkcije su izrađenje pomoću Math.random() metode sa specifičnim parametrima generacije, a pomoću test case vraćaja se dinamički parametar Text te prikazuje na ekranu. Bitno je bilo koristiti test case slučaj zbog problema cache-iranja podataka. Kada nebi koristili test case aplikacija bi jednom bacila kocku (iterirala kroz kod funkcije) te zapamtila vrijednost i bez obzira na pritisak nekog drugog gumba nebi pregazio prijašnju vrijednost već bi ona ostala ista. Također se ispisuje koji je broj dobio te na kojoj kockici.



Slika 33 - Ekran Dice Roller 1. dio



Slika 34 - Dice Roller 2. dio

Na slici 35 prikazan je programski kod funkcije Dice_Roller. Ovisno na koji gumb igrač pritisne, pomoću test case se funkciji roll prosljeđuje parametar koju kockicu mora bacati te vraća parametar text koji se kasnije u tekst segmentu renderira na ekranu pomoću {text} test case-a.

```
function Dice_Roller() {
  const [text,setText] = useState(null)
  return (
    <SafeAreaView style={{ flex: 1, alignItems: 'center', justifyContent: 'space-evenly'}}>
      <Text style={{paddingRight: 20, paddingLeft: 20}}>
          {text}
        </Text>
      <Button
          title="D4"
          onPress={() => setText(roll("d4"))}
      />
      <Button
          title="D6"
          onPress={() => setText(roll("d6"))}
      />
      <Button
          title="D8"
          onPress={() => setText(roll("d8"))}
      />
      <Button
          title="D10"
          onPress={() => setText(roll("d10"))}
      />
      <Button
          title="D12"
          onPress={() => setText(roll("d12"))}
      />
      <Button
          title="D20"
          onPress={() => setText(roll("d20"))}
      />
      <Button
          title="D100"
          onPress={() => setText(roll("d100"))}
      />
    </SafeAreaView>
  );
}
```

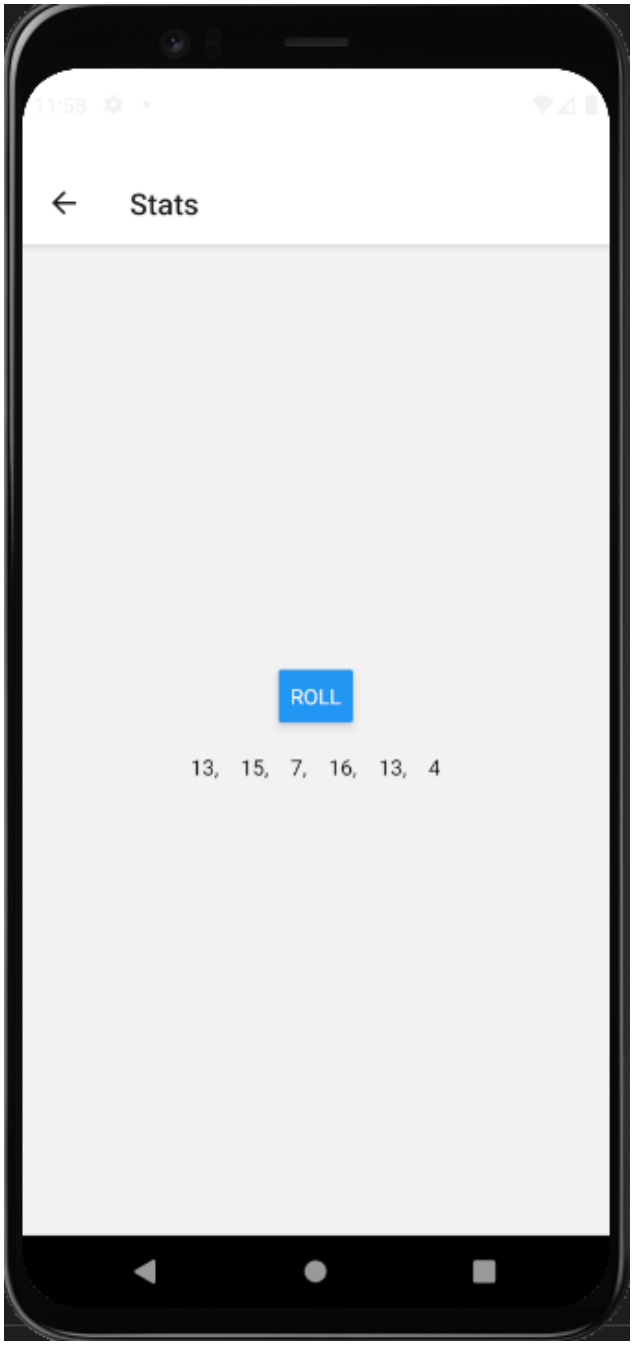
Slika 35 - Dice_Roller() funkcija

Na slici 36 prikazana je roll funkcija. Pomoću switcha detektira koji parametar je prosljeđen funkciji te ovisno o tome generira nasumični broj. Bitno je napomenuti da kockica od 100 strana može baciti brojeve od 0 do 99, stoga je njen programski za generaciju broja počinje od 0, za razliku od ostalih.

```
function roll(dice){
  switch(dice){
    case "d4":
      var RolledNumber = Math.floor(Math.random() * 4) + 1;
      break;
    case "d6":
      var RolledNumber = Math.floor(Math.random() * 6) + 1;
      break;
    case "d8":
      var RolledNumber = Math.floor(Math.random() * 8) + 1;
      break;
    case "d10":
      var RolledNumber = Math.floor(Math.random() * 10) + 1;
      break;
    case "d12":
      var RolledNumber = Math.floor(Math.random() * 12) + 1;
      break;
    case "d20":
      var RolledNumber = Math.floor(Math.random() * 20) + 1;
      break;
    case "d100":
      var RolledNumber = Math.floor(Math.random() * 100);
      break;
  }
  return RolledNumber
}
```

Slika 36 – roll funkcija

U D&Du se kocke također koriste pri izradi lika, tj. za određivanje statistike lika korištenjem određenih metoda. U alatu je implementiran idući način bacanja kockica; bacaju se četiri d6, te se zadržavaju tri najveća broja, što znači dobiti ćemo brojeve od minimum 3 (ako igrač baci četiri jedinice a zadrži tri jedinice) do 18 (ako igrač baci 3 šestice i bilo koji drugi broj, jer se zadržavaju tri najveće s toga četvrti broj nije bitan) te se postupak ponavlja šest puta te sa time kreira niz od šest brojeva. Zatim ako igrač nema barem dvije ili više petnaestice, na novo se kreira niz. Igrači zatim raspodjeljuju te brojeve u svoje statistike. Cilj Stats Rollera je ubrzati proces ručnog bacanja kockica automatizmom i matematikom programskog koda (slika 37).



Slika 37 - Stats ekran

Na slici 38 prikazan je programski kod funkcije Stats, ponovno koristeći test case funkciji StatArray šaljemo text koji kasnije u funkciji iz integer Arraya pretvaramo u string array te prikazujemo na ekranu pomoću {text} test case-a.

```
function Stats(){
  const [text,setText] = useState(null)
  return(
    <SafeAreaView style={{ flex: 1, alignItems: 'center', justifyContent: 'center'}}>
      <Button
              title="Roll"
              onPress={() => setText(StatArray())}
      />
      <Text style={{padding: 20}}>
              {text}
            </Text>
    </SafeAreaView>
  );
}
```

Slika 38 - Stats funkcija

Na slici 39 prikazan je programski kod koji koristi matematičku logiku iz igre kako bi stvorio niz statistika koje igrač može raspodijeliti. Definirane su varijable StatsRolled kao polje te RollCondition i checkForFifteen koje provjeravaju jesu li uvijeti zadovoljeni. Uvijet je da barem dva od šest brojeva u nizu moraju biti jednaki petnaest ili više (po pravilima igre). Također vrijedno je napomenuti da postoji razlika u generiranim brojevima. Ako se oni generiraju pomoću prve metode (zakomentirana linija) može nastati velika devijacija od standardnog polja (polje od šest mjesta ispunjeno brojem deset), stoga je korištena baš matematička metoda koja se koristi u igri. Simulira se bacanje četiri kockica sa šest strana, tako da dobiveni brojevi mogu iznositi od jedan do šest. Od njih se uzimaju tri najveća, najmanji miče, te ostatak zbraja i sprema u niz. U funkciji je to skraćeno zbog optimizacije. Generiraju se četiri broja koji se zbrajaju, od njih se traži najmanji te oduzima od ukupnog zbroja da se dobije broj između 3 i 18. Ako niz ne sadrži barem dva broja 15 ili više u sebi, while petlja se ponavlja dokle uvijet nije zadovoljen. Brojevi se spremaju u niz StatsRolled[] te se na njih nadodaje string koji ih automatski pretvara u text koji vraća nazad u prijašnju funkciju i pomoću test case-a prikazuje na ekranu.

```
function StatArray(){
  let StatsRolled = [];
  var RollCondition = 1;
  var checkForFifteens = 0;
  while(RollCondition){
    checkForFifteens = 0;
    StatsRolled = [];
    for(let i=0; i<6; i++){
      //var rng = Math.floor(Math.random() * 16) + 3;
      var roll1 = Math.floor(Math.random() * 6) + 1;
      var roll2 = Math.floor(Math.random() * 6) + 1;
      var roll3 = Math.floor(Math.random() * 6) + 1;
      var roll4 = Math.floor(Math.random() * 6) + 1;
      var rng = roll1 + roll2 + roll3 + roll4 - Math.min(roll1,roll2,roll3,roll4)
      if(i == 5){
        StatsRolled[i] = rng
      }
      else{
        StatsRolled[i] = rng + ",    ";
      }
      if(rng>=15){checkForFifteens +=1;}
      if(checkForFifteens>=2){RollCondition = 0;}
    }
  };
  return StatsRolled;
}
```

Slika 39 - Stat Array funkcija

Zaključak

React Native je odličan za razvijanje aplikacija za više okruženja, zbog mogućnosti korištenja modula i reusable code-a, znatno ušteđuje na vremenu i trošku izrade aplikacije. Vrlo je jednostavan za učiti te ima odličnu dokumentaciju za izradu aplikacija koja podkrijepljuje i jednostavno objašnjava module i funkcije te kako one funkcioniraju, a pripomaže i činjenica popularnosti, što znači da ima jako puno video i vizualnih objašnjena na platformama kao što su Youtube, Reddit i StackOverflow. Expo Go je savršen alat za testiranje aplikacija u pravom vremenu, a Android Studio odličan alat za emuliranje uređaja. Sve u svemu React Native je vrlo jednostavan i lako naučiv zbog ogromne podrše koju ima kroz godine razvoja, a vjerojatno će se rasprostraniti još više zbog uštede vremena i novaca koju pruža tvrtkama i indie developerima.

Literatura

[1] App Trends https://blog.hubspot.com/website/web-development-trends

[2] JavaScript https://www.w3schools.com/js/

https://www.javascript.com/

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

[3] ECMA-262 https://www.ecma-international.org/publications-and-standards/standards/ecma-262/

[4] React Native https://reactnative.dev/

https://www.netguru.com/glossary/react-native

[5] React vs HTML5 https://softwareplanetgroup.co.uk/native-vs-html5-vs-react-natve-apps-who-wins-updated/

[6] React Native Doc https://reactnative.dev/docs/getting-started

[7] React Native Reddit https://www.reddit.com/r/reactnative/

[8] Android vs IOS https://decode.agency/article/native-vs-cross-platform-mobile-apps/

[9] JS Threads https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers

[10] Existing app threads https://developer.android.com/guide/components/processes-and-threads

[11] Android Studio https://developer.android.com/studio

https://searchmobilecomputing.techtarget.com/definition/Android-Studio

[12] GCM https://en.wikipedia.org/wiki/Google_Cloud_Messaging

[13] GAE https://en.wikipedia.org/wiki/Google_App_Engine

[14] D&D https://hr.wikipedia.org/wiki/Dungeons_%26_Dragons

https://dnd.wizards.com/

[15] Wargames https://hr2.wiki/wiki/Wargame

[16] Expo Go https://docs.expo.dev/get-started/create-a-new-app/

[17] React Hooks https://reactjs.org/docs/hooks-intro.html