Razvoj aplikacija za više platformi primjenom modernih Web tehnologija

Anđel, Mihael

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Mihael Anđel

RAZVOJ APLIKACIJA ZA VIŠE PLATFORMI PRIMJENOM MODERNIH WEB TEHNOLOGIJA

DIPLOMSKI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mihael Anđel

Matični broj: 44864/16–R

Studij: Informacijsko i programsko inženjerstvo

RAZVOJ APLIKACIJA ZA VIŠE PLATFORMI PRIMJENOM MODERNIH WEB TEHNOLOGIJA

DIPLOMSKI RAD

Mentor :

Prof. dr. sc. Dragutin Kermek

Varaždin, srpanj 2022.

Mihael Anđel

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Web aplikacije sve su više prisutne u svakodnevnom životu. Postoji mnogo platformi za razvoj stolnih i mobilnih aplikacija. Nativne aplikacije imaju svoje prednosti što se tiče performansi, no probleme stvara velik broj tehnologija za razvoj aplikacija na više platformi. Web aplikacije ne ovise o platformi, već o Web pregledniku koji je u pravilu standardiziran. Postoji nekoliko temeljnih Web tehnologija pomoću kojih se gradi svaka Web aplikacija. Usprkos svojim manama, Web aplikacije mogu se koristiti za razvoj mobilnih i stolnih aplikacija, što može uvelike olakšati i ubrzati razvoj aplikacije. Neke od najpopularnijih tehnologija za razvoj izvornih aplikacija Web tehnologijama su *React Native* i *Electron*. *React Native* tehnologija koristi snage *React* JavaScript biblioteke za razvoj mobilnih Android i iOS aplikacija. *Electron* JavaScript okvir pruža veliku količinu sučelja za razvoj izvornih stolnih aplikacija primjenom temeljnih Web tehnologija. Obje tehnologije, kao i standardne izvorne i Web aplikacije, imaju svoje specifične prednosti i mane. Za praktični dio rada razvijene su dvije aplikacije. Prva aplikacija je izvorna stolna aplikacija za Windows, Linux i MacOS, dok je druga izvorna mobilna aplikacija za Android i iOS. Stolna aplikacija razvijena je primjenom *Electron* okvira, a mobilna primjenom *React Native* tehnologije. Prikaz rezultata razvoja aplikacija pomoću snimaka zaslona i isječaka programskog koda.

Ključne riječi: Web; Web aplikacija; JavaScript; Web tehnologija; NodeJS; TypeScript; mobilna aplikacija; platforma; React Native; Electron

Sadržaj

1. Uvod

Web je u današnjici neizbježan dio svačijeg svakodnevnog života. Od novosti, društvenih mreža i web trgovina, pa sve do mobilnog bankarstva, Web brokera i platformi za uživi prijenos multimedijskog sadržaja - nemoguće je opovrgnuti činjenicu da je Web sve oko nas. Činjenica je da je Web u današnjem svijetu nezamjenjiv i sveobuhvatan. Ne samo to, već sama popularnost Weba raste iz dana u dan - sve više aspekata našeg života pokušava se digitalizirati i uz to prenijeti na Web.

Srodno s time raste i popularnost razvoja Web aplikacija, a na tržištu se nalazi sve više i više Web razvojnih programera. Velika količina tih programera koristi neke od najnovijih i najmodernijih tehnologija za razvoj Weba, bilo to na strani poslužitelja ili na strani klijenta. Naravno, sam Web, odnosno klijentski dio, izgrađen je gotovo isključivo pomoću tri temeljne tehnologije, a to su HTML, CSS i JavaScript. Što se tiče poslužiteljskog dijela, tu postoji puno veća sloboda odabira što se tiče programskih jezika i tehnologija za razvoj.

S druge strane, izvorne stolne i mobilne aplikacije tradicionalno su razvijane pomoću standardnih tehnologija koje su obično vezane uz platformu na kojoj se koriste za razvoj. Primjerice, aplikacije za Android operacijski sustav u pravilu su razvijene u Java programskom jeziku, no Kotlin programski jezik postaje sve popularnija opcija za razvoj Android aplikacija. S druge strane aplikacije za iOS operacijski sustav su obično razvijene u Swift (starije u Objective-C) programskom jeziku.

Velika prednost izvornih aplikacija je oduvijek bila lakoća instalacije, jednostavnost pristupa i "udobnije" korištenje. Nema potrebe za otvaranjem Web preglednika i utipkavanjem adrese do Web lokacije kako bi se pristupilo nekom sustavu. Također, direktan pristup podsustavima operacijskog sustava omogućuje ljepše iskustvo za krajnjeg korisnika aplikacije.

Uzme li se u obzir sve navedeno, postavlja se pitanje: zašto ne razviti izvornu aplikaciju tehnologijama kojima već baratamo, odnosno Web tehnologijama?

U ovom radu biti će istražene i prikazane tehnologije za izradu stolnih i mobilnih aplikacija pomoću temeljnih Web tehnologija. Biti će napravljen pregled sveukupnog razvoja izvornih, pa tako i Web, aplikacija - prednosti, nedostaci i općenite karakteristike razvoja obje vrste aplikacija.

2. Izvorna aplikacija

Izvorna aplikacija mogla bi se opisati kao bilo koja aplikacija čija namjena je da se izvršava na određenoj platformi, odnosno operacijskom sustavu. Izvorna aplikacija također može biti razvijena za rad na više različitih platformi, obično korištenjem posebnih razvojnih okvira koji to omogućuju. Osim toga, izvorna aplikacija mogla bi se opisati kao aplikacija koja nije razvijena za Web, odnosno ne izvršava se unutar okrilja Web preglednika.

U ovom poglavlju napraviti će se pregled najčešće korištenih tehnologija za razvoj izvornih stolnih i mobilnih aplikacija. Pregled tehnologija napraviti će se na primjeru nekoliko najpopularnijih platformi, odnosno operacijskih sustava, današnjice. Također će se istražiti koraci razvoja izvornih aplikacija, od odabira tehnologija, pa do same implementacije i izgradnje konačnog proizvoda. Na kraju poglavlja iznijeti će se glavne prepreke, odnosno problemi, kod razvoja izvornih stolnih i mobilnih aplikacija za bilo koju platformu.

2.1. Glavne tehnologije za razvoj

Tehnologije za razvoj izvornih stolnih i mobilnih aplikacija mogu se podijeliti u dvije vrste. Jedna vrsta je tehnologija za programiranje same poslovne logike aplikacije, dok je druga tehnologija odgovorna za prikaz korisničkog sučelja aplikacije. Naravno, te tehnologije moraju međusobno biti povezane.

Općeniti način povezivanja ovih tehnologija je korištenje sučelja za programiranje aplikacija (*API*). Najčešći oblik sučelja za programiranje aplikacija je takav da se komponente korisničkog sučelja direktno koriste u programskom kodu, odnosno postoje funkcije ili klase koje odgovaraju pojedinoj komponenti sučelja. Još jedan česti oblik povezivanja je taj da se definicija korisničkog sučelja nalazi u posebnom formatu u zasebnoj datoteci, no i dalje mora postojati neko sučelje za programiranje komponenti korisničkog sučelja.

2.1.1. Windows

Što se tiče programiranja aplikacija za Windows operacijski sustav, izbor tehnologija je poprilično širok. Microsoft relativno često uvodi nove tehnologije za razvoj aplikacija za vlastitu platformu, no starije tehnologije su i dalje upotrebljive. Zapravo, starije tehnologije se i danas koriste, ovisno o potrebama razvojnih programera.

Najnovija tehnologija za razvoj izvornih stolnih aplikacija za Windows operacijski sustav je *WinUI3*. Također, Microsoft preporučuje upravo ovu tehnologiju za razvoj novih Windows aplikacija [1]. Od Microsoft-ovih tehnologija za razvoj izvornih aplikacija za Windows platformu postoje sljedeće [1]:

- Win32
- Windows Forms

- WPF
- UWP

Win32 jedna je od starijih tehnologija, a koristi se za razvoj Win32 stolnih Windows aplikacija. Često se koristi drugi naziv za takve aplikacije: klasične stolne aplikacije [1]. Win32 tehnologija agnostična je što se tiče programskih jezika, odnosno nije strogo vezana uz pojedini programski jezik. Postoje Win32 biblioteke za više različitih jezika. Za najbolje performanse preporučuje se koristiti programski jezik C++ i službenu Win32 biblioteku, odnosno sučelje za programiranje aplikacija [1]. Kratki primjer C++ programskog koda kojim se otvara novi prazan prozor u Windows operacijskom sustavu [1]:

```
HWND hwnd = CreateWindowEx(
    0,
    L"Sample Window Class",
    L"Learn to Program Windows",
    WS_OVERLAPPEDWINDOW,
    // velicina i pozicija prozora
    CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,

    NULL,
    NULL,
    hInstance,
    NULL
);

ShowWindow(hwnd);
```

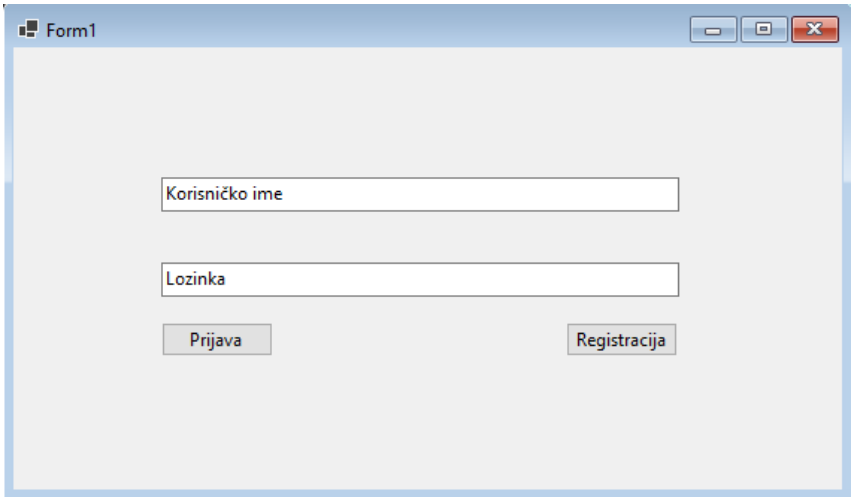
Kao što se može vidjeti, Win32 biblioteka poprilično je niske razine, pa je potrebna visoka razina iskustva kod korištenja ove biblioteke, pogotovo u sklopu s C++ programskim jezikom.

Windows Forms jedna je od najpoznatijih tehnologija za razvoj Windows stolnih aplikacija, a prepoznatljiva je po svojem standardnom izgledu korisničkog sučelja. Koristi se u sklopu .NET razvojnog okvira kojeg je također razvio Microsoft. Kao takva, dostupna je bilo kojem programskom jeziku koji spada u obitelj .NET programskih jezika, no najpopularniji je definitivno C#. Tehnologija je specifična po tome što se razvoj korisničkog sučelja svodi na alat za dizajniranje koji je integriran u Visual Studio alat. Nema potrebe za pisanjem nekog jezika za označavanje kako bi se realiziralo korisničko sučelje. Iz tog razloga je tehnologija stekla relativno veliku popularnost. To za sobom vuče neka ograničenja. Primjerice nastaje više posla kada se Windows Forms aplikacija želi intenzivnije urediti što se tiče izgleda. [1]

Kratki primjer male Windows Form aplikacije pisane u C# programskom jeziku [autorski rad]:

```
public partial class LoginForm : Form
  {
      public LoginForm()
      {
          InitializeComponent();
      }

      private void loginButton_Click(object sender, EventArgs e)
      {
          string username = usernameInput.Text;
          string password = passwordInput.Text;
```

```
            LoginService service = new LoginService();

            User user = loginservice.loginUser(username, password);
            if (user != null)
            {
                HomepageForm form = new HomepageForm();
                form.Show();
            }
        }

        private void registerButton_Click(object sender, EventArgs e)
        {
            string username = usernameInput.Text;
            string password = passwordInput.Text;

            LoginService service = new LoginService();

            User user = loginservice.registerUser(username, password);
            if (user != null)
            {
                HomepageForm form = new HomepageForm();
                form.Show();
            }
        }
    }
```



Slika 1: Prikaz, odnosno grafičko sučelje, Windows Forms aplikacije [autorski rad]

WPF i UWP relativno su slične tehnologije. UWP je novija tehnologija, no obje tehnologije imaju slične korijene. Primjerice, obje tehnologije koriste XAML jezik za označavanje kako bi korisničko sučelje mogli odvojiti u zasebnu datoteku. Na taj se način poslovna logika, odnosno sam programski kod, može nalaziti odvojen u svojoj datoteci. Osim toga, obje tehnologije vezane su uz .NET razvojni okvir, što znači da, kao i kod Windows Forms tehnologije, se u skladu s njima mogu koristiti svi jezici .NET obitelji. UWP tehnologija je posebna po tome što se njome mogu razviti aplikacije koje se mogu izvršiti na više Microsoft-ovih proizvoda kao što je Xbox igraća konzola. [1]

Kratki primjer WPF Windows aplikacije pisane u C# programskom jeziku [autorski rad]:

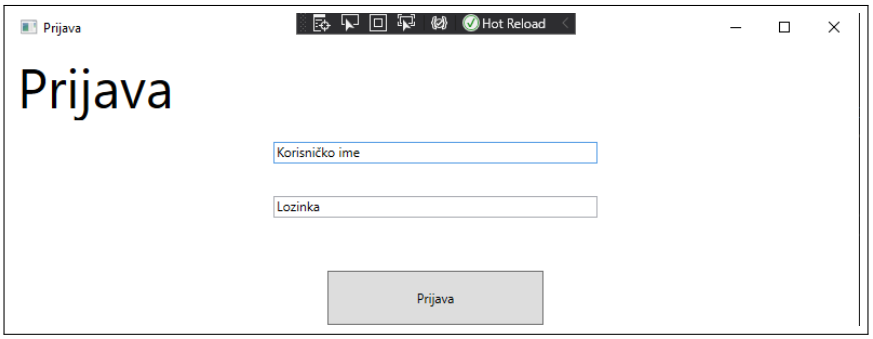
```
    private void loginButton_Click(object sender, EventArgs e)
    {
        string username = usernameInput.Text;
        string password = passwordInput.Text;

        LoginService service = new LoginService();

        User user = loginservice.loginUser(username, password);
```

```
        if (user != null)
        {
            HomePage newPage = new HomePage();
            newPage.Show();
        }
    }
```

Ovdje se nalazi samo kod za prijavu korisnika. Kao što se može vidjeti, programski kod je gotovo identičan onome u Windows Forms aplikaciji.

Korisničko sučelje aplikacije pisano u XAML jeziku za označavanje [autorski rad]:

```
<Grid>
      <Grid.RowDefinitions>
          <RowDefinition Height="75"/>
          <RowDefinition Height="50"/>
          <RowDefinition Height="50"/>
          <RowDefinition Height="50"/>
      </Grid.RowDefinitions>

      <Label FontSize="50">Prijava</Label>
      <TextBox Grid.Row="1" Height="20" Width="300" Margin="10 10 10 10" Text="
          Korisnicko ime" HorizontalAlignment="Center" Name="usernameInput"/>
      <TextBox Grid.Row="2" Height="20" Width="300" Margin="10 10 10 10" Text="
          Lozinka" HorizontalAlignment="Center" Name="passwordInput"/>
      <Button Width="200" Content="Prijava" Margin="0,34,0,-34" Grid.Row="3"/>
  </Grid>
```



Slika 2: Rezultat, odnosno konačno grafičko sučelje, WPF aplikacije [autorski rad]

Glavna namjena WPF i UWP tehnologija je razvoj aplikacija kojima su potrebne nešto kompleksniji grafički prikazi. Tako se, primjerice, ovim tehnologijama mogu razviti i video igre. Za razliku od Windows Forms tehnologije, jednostavnije je kreirati detaljnije i kompleksnije dizajne sučelja aplikacije. Budući da su obje tehnologije nešto novije od prije spomenutih, relativno jednostavno se mogu "prevesti", odnosno migrirati, u najnoviju tehnologiju - WinUI3. [1]

Što se tiče razvojnog okruženja, za razvoj Windows aplikacija obično se preporučuje Microsoft-ov Visual Studio. Visual Studio je veliko integrirano razvojno okruženje s podrškom za mnoštvo programskih jezika i tehnologija za razvoj aplikacija. [1]

2.1.2. Linux, macOS, Android i iOS

Linux je izrazito specifičan sustav. Razlog tome je velika količina različitih i specifičnih distribucija. Svaka distribucija mogla bi se nazvati svojim operacijskim sustavom zbog razlika u radnom okruženju, ugrađenih aplikacijama i alatima i razlika u općenitom izgledu grafičkog

sučelja. Usprkos tomu, jezgra svake Linux distribucije je identična, tako da bi u praksi sve aplikacije pisane za jednu distribucije trebale raditi u bilo kojoj drugoj. Jedna od glavnih prepreka tome su drugačiji načini pakiranja izgrađenih aplikacija. Iako je to prepreka, zapravo nije velika prepreka jer se može riješiti relativno jednostavno - izvorni kod je i dalje identičan za sve distribucije.

Usprkos tome što su sve distribucije manje-više identične, što znači da programski kod može ostati isti, razvojni okvir za razvoj grafičkog sučelja aplikacije može se razlikovati. Srećom, primarno se koriste dva razvojna okvira za razvoj grafičkih sučelja Linux aplikacija: GTK i Qt, odnosno KDE razvojni okvir. GTK razvojni okvir popularizirani je Ubuntu Linux distribucijom, odnosno njenim GNOME radnim okruženjem. GTK može se koristiti s gotovo bilo kojim modernim programskim jezikom, od C++-a do Python-a, pa čak i JavaScript-a. Qt razvojni okvir primarno se koristi na distribucijama koje koriste KDE Plasma radno okruženje, a glavni programski jezik je C++. [2] [3]

MacOS i iOS održava tvrtka Apple, pa ujedno razvija nove tehnologije za razvoj aplikacija na svojoj platformi. Središnji dio macOS platforme koji povezuje sve ostale tehnologije je macOS SDK, odnosno skup alata za razvoj aplikacija na macOS platformi. Uz macOS SDK moguće je koristiti hrpu ostalih tehnologija i alata koje nudi Apple. Glavna, ali starija, tehnologija za razvoj macOS stolnih aplikacija je Cocoa, a u zadnje vrijeme zamjenjuje ju SwiftUI tehnologija. Cocoa tehnologija koristi se uglavnom s Objective-C programskim jezikom, dok se novi SwiftUI koristi s istoimenim Swift programskim jezikom. Glavna prednost nove SwiftUI tehnologije je to što je svi noviji Apple proizvodi podržavaju. Posljedica toga je to što se jednom napisani programski kod može koristiti više puta za razvoj iste aplikacije na mnoštvo platformi Apple. Isto kao i kod macOS platforme, iOS platforma isključivo koristi SwiftUI razvojni okvir, uz Swift programski jezik. Osim toga, iOS platforma koristi svoj vlastiti SDK, koji je posebno prilagođen za mobilne uređaje poput iPhone-a. Neke od ostalih, češće korištenih tehnologija, su ARKit i RealityKit za proširenu stvarnost, Create ML i Core ML za strojno učenje i Metal za izrazito zahtjevne aplikacije s kompleksnim grafikama. [4]

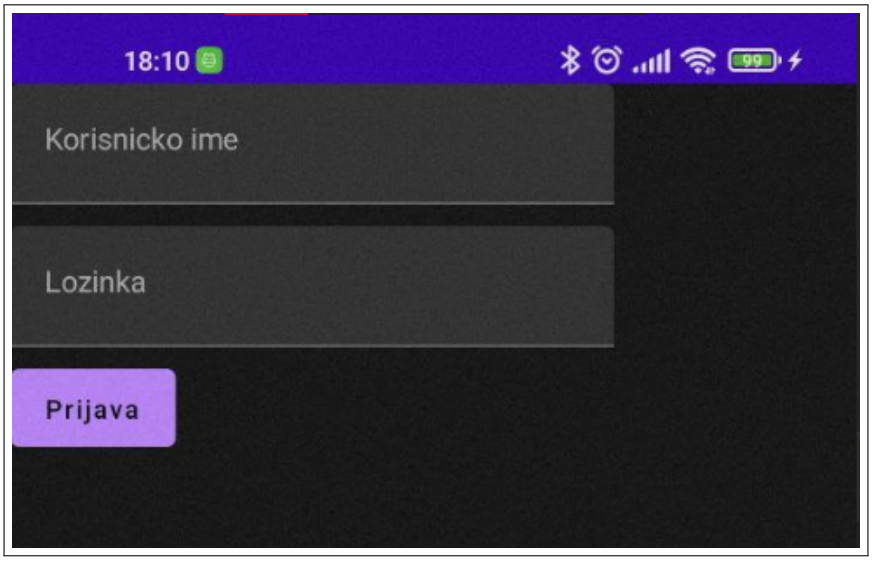
Glavni, i u praksi jedini, alat za razvoj svih aplikacija baziranih na Apple tehnologijama je Xcode. Xcode je ekvivalent Visual Studio integriranom razvojnom okruženju za macOS. Alat podržava sve Apple platforme, od stolnih računala, do mobitela, tableta, pametnih satova i televizora. [4]

Android je operacijski sustav za mobilne telefone razvijen od strane Google-a. Čitav operacijski sustav baziran je na Linux operacijskom sustavu - zapravo koristi i istu jezgru operacijskog sustava. Manje-više sve ostale komponente Android sustava razvijene su primjenom Java tehnologija. Središnji paket razvojnih tehnologija i alata je Android Jetpack. Android Jetpack sadrži sve potrebne tehnologije za razvoj Android aplikacije. U zadnje vrijeme sve popularniji postaje Kotlin programski jezik. Kotlin je napravljen od strane Google-a i namijenjen je kao zamjena za Javu kod razvoja Android aplikacija. Iako je i dalje direktno povezan s Java tehnologija jer se izvršava u JVM, Android razvojni programer više ne mora znati programirati u Java programskom jeziku. Za razvoj korisničkog sučelja Android aplikacija koristi se Jetpack Compose komponenta Android Jetpack paketa. [5]

Kratki primjer Jetpack Compose sučelja napisanog u Kotlin programskom jeziku [autorski rad]:

```
class MainActivity : ComponentActivity() {
    var username: String = ""
    var password: String = ""

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MandelTheme {
                Surface() {
                    Column(verticalArrangement = Arrangement.spacedBy(Dp(10f))) {
                        TextField(value = "", onValueChange = { username = it },
                            label = { Text("Korisnicko ime")})
                        TextField(value = "",
                            visualTransformation = PasswordVisualTransformation(),
                            onValueChange = { password = it }, label = { Text("
                                Lozinka")})
                        Button(onClick = { login() }) {
                            Text(text = "Prijava")
                        }
                    }
                }
            }
        }
    }

    fun login() {
        LoginService.loginUser(username, password)
    }
}
```

Primjer predstavlja jednu Kotlin klasu u kojoj se nalazi definicija sučelja za jedan pogled aplikacije. Zanimljivo je kako se sučelje definira. Ne koristi se nikakav jezik za označavanje kao što HTML ili XAML. Elementi sučelja zapravo su zasebne klase koje primaju određene parametre, neki od kojih su obavezni, a neki ne. Najčešće su neobavezni parametri oni koji mijenjaju zadano ponašanje ili izgled elementa. Ostatak klase je poprilično standardan. Moguće je definirati varijable klase i metode koje se pozivaju kad se okine neki događaj u aplikaciji.



Slika 3: Jetpack Compose grafičko sučelje [autorski rad]

2.2. Koraci razvoja

Razvoj bilo koje aplikacije dug je proces neovisno o tome radilo li se o izvornoj, Web ili nekoj trećoj vrsti aplikacije. Iako ne postoji neka stroga, formalna lista faza ili koraka razvoja izvorne aplikacije, takva lista može se relativno jednostavno sastaviti. Naravno, lista koja će se sastaviti ovdje nikako ne predstavlja apsolutno sve aspekte razvoja izvorne aplikacije. Glavni razlog sastavljanja ovakve liste zapravo je usporedba, odnosno analiza, razlika između razvoja izvorne i Web aplikacije. Lista koraka je:

- razvoj ideje aplikacije,
- analiza tržišta i krajnjeg korisnika,
- odabir platforme/i za razvoj,
- analiza i odabir najbolje tehnologije za razvoj,
- razvoj aplikacije,
- odabir načina pakiranja i distribucije aplikacije

Razvoj, odnosno dolazak do, same ideje aplikacije može biti relativno kratak, ali i poprilično dug proces. Ovaj korak direktno utječe na ostale korake jer sam tip aplikacije može usmjeriti razvojnog programera na odabir platforme za razvoj, pa čak i na odabir tehnologija i alata koje će se koristiti kod razvoja. Kao takav, ovaj korak nije ovisan o nijednom drugom koraku, no svi ostali koraci do neke mjere ovise o njemu.

Analiza tržišta logički slijedi nakon razvoja ideje jer, slično kao i prethodni korak, direktno utječe na sve sljedeće korake. Ovdje razvojni programer postavlja mnogo pitanje vezanih uz to kako će se i gdje će se aplikacija koristiti. Ima li smisla razviti aplikaciju za mobilne uređaje ili će se aplikacija primarno koristiti u uredima gdje se koriste stolna računala? U ovaj korak spada i analiza krajnjeg korisnika aplikacije. Nema smisla da se aplikacija razvije za korisnike koji ju rijetko ili gotovo nikad neće koristiti.

Odabir platforme za razvoj aplikacije jedan je od najbitnijih koraka što se tiče općenitog odabira tehnologija. Ovaj korak usko je vezan uz prijašnji korak. Ako je najprimjereniji uređaj za korištenje aplikacije mobilni telefon, logički odabir platforme za razvoj je Android ili iOS. Opet, potrebno je analizirati krajnjeg korisnika i vidjeti koja se od tih dviju platformi više koristi. Kod mobilnih platformi, država u kojoj žive ciljani korisnici puno utječe na korištenost neke platforme. Primjerice, Apple-ov iOS puno je popularniji u SAD-u, dok je Google-ov Android izrazito popularan u siromašnijim zemljama poput Indije. Naravno, postoji i opcija razvoja dvije verzije aplikacije, jednu za svaku ciljanu platformu. S druge strane, ako je pak najbolja platforma stolna aplikacija, izbor platformi je još veći. Svaka platforma je specifična i ima svoje nedostatke i prednosti. Primjerice, ako se cilja da se aplikacija instalira i koristi na što više računala, očiti izbor je Windows platforma zbog svoje zastupljenosti na tržištu. Linux platforma mogla bi biti pogodnija ako je ciljani korisnik iskusniji u korištenju računala: razvojni programer,

informatičar ili administrator sustava. MacOS poprilično je popularan kod grafičkih dizajnera i ostalih umjetnika poput glazbenih producenata, tako da valja i to uzeti u obzir.

Nakon što je odabrana platforma za razvoj aplikacije potrebno je odlučiti koje tehnologije će biti najpogodnije za razvoj aplikacije. Sve ovisi o prioritetima aplikacije. Prioritet može, primjerice, biti brzina odvijanja ili lakoća razvoja specifične funkcionalnosti aplikacije.Najočitiji izbor su tehnologije koje će najviše olakšati sam razvoj aplikacije. Prednosti takvih tehnologija su, naravno, jednostavnost korištenja i brzina razvoja aplikacije. S druge strane, ukoliko aplikacija zahtjeva visoku razinu performansi, odabir tehnologija se mijenja. Konkretno, za stolna računala C++ programski jezik i mnoštvo podržanih biblioteka i okvira dobar su izbor za aplikacije koje zahtijevaju visoku razinu performansi i brzinu izvršavanja aplikacije. Osim toga, C++ je podržan na manje-više svim modernim platformama. S druge strane, za općenite poslovne aplikacije, pogotovo tamo gdje je pogodan MVC uzorak dizajna, odličan izbor je .NET razvojni okvir i C# programski jezik [1]. Što se tiče mobilnih aplikacija odabir tehnologija je definitivno sužen. Usprkos tomu razvojni programeri koji održavaju mobilne tehnologije pružaju razne module pomoću kojih se s lakoćom mogu razviti specijalizirane aplikacije.

Razvoj same aplikacije puno ovisi o prijašnjim koracima. Točnije, puno ovisi o odabranim platformama i tehnologijama za razvoj. Sve, od načina programiranja do raznim konvencija kod razvoja, ovisi o tome koje tehnologije su odabrane. Uz to, trajanje ovog koraka izrazito je varijabilno. Naravno, trajanje ovisi o tome koliko se kompleksna aplikacija razvija, ali i o tome je li odabrana jedna platforma za razvoj ili njih više. Dio izvornog koda uvijek se može ponovo iskoristiti kod razvoje

Za kraj, bitan korak je odabir načina pakiranja aplikacije, kao i odabir načina distribucije aplikacije. Ovaj korak najviše ovisi o platformi za koju je aplikacija razvijena. Ukoliko se radi modernim mobilnim platformama poput Android i iOS operacijskih sustava, čitav proces pakiranja i distribucije aplikacije jako je pojednostavljen. Kod Android platforme, alat Android Studio nudi opcije pakiranja aplikacije, a sam paket se relativno jednostavno prenese na Google-ov Play Store. Vrlo slična priča je i s iOS aplikacijama. XCode alat nudi funkcionalnost pakiranja aplikacije koja se tada može prenijeti na Apple-ov App Store. App Store ima puno strože uvjete koje aplikacija mora zadovoljavati prije nego što bude odobrena za preuzimanje. Kod Linux platforme odabir je, kao i kod ostalih koraka, poprilično velik. Jedan od popularnijih načina pakiranja aplikacija je Snap - tehnologija koja omogućuje pokretanje aplikacije na bilo kojoj Linux distribuciji. Microsoft i Apple, slično kao i kod mobilnih platformi, nude svoje varijante pakiranja i distribucije aplikacije za svoje stolne platforme. [5][4]

2.3. Glavne prepreke kod razvoja

Svaka vrsta aplikacije, pa tako i izvorna aplikacija, ima određene prepreke kod razvoja. Što se tiče izvorne aplikacije, te prepreke se u pravilu svode na to da postoji mnoštvo izvornih platformi, bilo za mobilne ili stolne aplikacije. Često se dogodi to da se aplikacija želi razviti za više, ili čak sve popularne, platforme. U tom slučaju nastaje mnogo problema.

Najveći problem je to što se za svaku platformu mora razviti zasebna aplikacija. Ovo

samo po sebi znači da se prije navedeni koraci razvoja izvorne aplikacije moraju više puta ponavljati. Ovo za sobom povlači dodatne probleme. Primjerice, ako aplikaciju razvija grupa razvojnih programera, u grupi se moraju nalaziti programeri za svaku od platformi za koju se želi razviti aplikacija. Alternativa tomu je manji broj razvojnih programera koji su vrsni u raznim tehnologijama, što zahtijeva veći trud za svakog pojedinog programera. S druge strane, ako se radi o jednom razvojnom programeru koji sam razvija aplikaciju za više platformi, tada on sam mora učiti mnogo različitih tehnologija što postaje ogromna vremenska investicija. Osim samih tehnologija, svaka platforma obično ima specifičan razvojni alat koji je potrebno koristiti za razvoj, učenje korištenja kojeg ponovo zahtijeva veliku vremensku investiciju. Još jedan problem definitivno snažnog vremenskog i financijskog utjecaja je posjedovanje hardvera, odnosno uređaja, za razvoj na specifičnim platformama. Standardno stolno računalo dovoljno je za razvoj na nekoliko platformi, no svakako je poželjno imati mobilni telefon na kojem se direktno može testirati aplikacija. Također, standardno stolno računalo nije kompatibilno s Apple uređajima, odnosno potrebno je posjedovati barem jedno stolno Apple računalo kako bi se mogla razviti aplikacija na bilo kojoj Apple platformi. Između ostalog, postoje i razni standardni kojih se aplikacije moraju pridržavati ukoliko ih se želi prenijeti i promovirati na raznim trgovinama aplikacija poput Microsoft Store i App Store.

Sve ovo zapravo se svodi na to da je potrebno uložiti puno više vremena i truda za razvoj iste aplikacije za više platformi. Problem nastaje kod poduzeća koja se bave razvojem aplikacija i kojima vrijeme zapravo znači novac. Često se želi postići maksimalna razina dostupnosti aplikacije, što znači da postoji interes za to da se aplikacija razvije tako da radi na što više mobilnih i stolnih platformi. Isti problem, ali zbog malo drugačijeg razloga, nastaje kod samostalnog razvojnog programera koji samo želi što prije razviti aplikaciju koju će što više ljudi moći koristiti.

Tablica 1: Prikaz tehnologija za razvoj izvornih aplikacija po odabranim operacijskim sustavima

Operacijski sustav	Programski jezici	Grafičko sučelje	Distribucija
Windows	C/C++, C#, Java, Python	Win32, Windows Forms, WPF, UWP. Tehnologije uglavnom temeljene na XML jeziku za označavanje.	Izgradnja i dijeljenje *.exe* datoteka, postavljanje aplikacije na *Microsoft Store*.
MacOS	Objective-C, Java, Swift	Cocoa, SwiftUI. Moderne tehnologije koje spajaju programski kod i prezentacijski sloj aplikacije, odnosno grafičko sučelje.	Postavljanje aplikacije na Mac App Store, izgradnja i dijeljenje *.dmg* datoteka.
Linux	C/C++, Java, Python	GTK, Qt. U pravilu tehnologije otvorenog koda. Velika količina različitih distribucija sprječava nastajanje standarda.	Dijeljenje *snap* paketa. Izgradnja i dijeljenje instalacijskih paketa za specifičnu distribuciju.
Android	Java, Kotlin	Jetpack Compose. Google pruža jako pojednostavljene, ali moćne metode izgradnje sučelja.	Postavljanje aplikacija na Google Play Store. Moguće dijeljenje aplikacijskih paketa uz sigurnosne rizike.
iOS	Swift, Objective-C	SwiftUI. Apple pruža jako pojednostavljene, ali moćne metode izgradnje sučelja.	Postavljanje aplikacije na App Store. Moguće dijeljenje aplikacijskih paketa uz sigurnosne rizike.

(Izvor: [1][4][5][2][3])

3. Web aplikacija

Web aplikacija, kao i izvorna aplikacija, nema neku strogu definiciju. Usprkos tomu, može se dati općenita definicija što to znači da je neka aplikacija Web aplikacija. Web aplikacija je pojam koji je teže definirati od pojma izvorne aplikacije. Web aplikacija se, kao i izvorna aplikacija, tehnički izvršava na operacijskom sustavu uređaja. Razlika između izvorne i Web aplikacije je u tome što se sav sadržaj, odnosno sučelje i programski kod, aplikacije nalazi na udaljenom poslužitelju. Web preglednik služi samo kao klijent koji preuzima sve datoteke potrebne za izvršavanje Web aplikacije. Kako bi se Web aplikacija razlikovala od obične Web stranice mora biti dinamična. To znači da uvelike ovisi o programskom kodu i zahtjeva razvojnog programera koji će ju razviti. U pravilu svaki Web preglednik, radi sigurnosnih i sličnih razloga, stvara pješčanik za svaku aplikaciju koja se odvija u njemu. Tako ispada da je svaka Web aplikacija izolirana od ostatka sustava. [6]

U ovom poglavlju napraviti će se preslika prijašnjeg poglavlja, no baviti će se isključivo Web aplikacijama. Dakle, napraviti će se pregled najkorištenijih tehnologija za razvoj Web aplikacija. Također će se napraviti razrada koraka razvoja Web aplikacija, kao i najčešće i najutjecajnije poteškoće kod razvoja Web aplikacija.

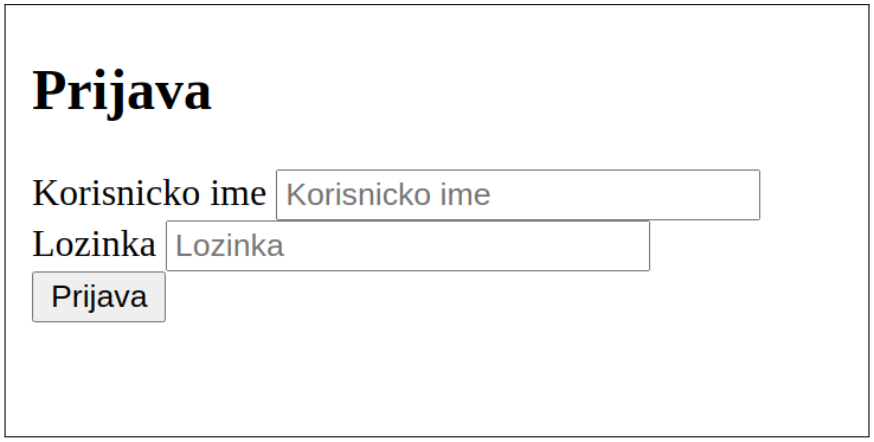
3.1. Glavne tehnologije za razvoj

Web je sam po sebi kao tehnologija od početka bio razvijen u svrhu da se s lakoćom mogu dijeliti razni dokumenti poput stručnih i znanstvenih radova. Uz to, Web je nelinearno strukturiran na način da svaki dokument može preko hiperlinka, odnosno poveznice, referencirati bilo koji drugi dokument. Korisnik jednim klikom miša dolazi do referenciranog dokumenta. Za izradu tih dokumenata također je razvijen novi jezik za označavanje, a danas je vjerojatno najpoznatiji takav jezik - HTML (*HyperText Markup Language*). HTML je i danas jedna od tri temeljne tehnologije na kojima je čitav Web izgrađen. Po strukturi i sintaksi sličan je XML jeziku za označavanje. Najnovija verzija jezika, HTML5, razvojnim programerima pruža ogroman broj ugrađenih funkcionalnosti. Te funkcionalnosti su često toliko napredne da za jednostavnije Web aplikacije nije potrebno napisati niti jednu liniju programskog koda. [6]

Kratak primjer obrasca za prijavu napisanog u HTML-u [autorski rad]:

```
<!DOCTYPE html>
<html lang="hr">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width,␣initial-scale=1.0">
  <link rel="stylesheet" href="index.css">
  <title>Prijava</title>
</head>
<body>

  <h2 class="title">Prijava</h2>

  <div class="center">
    <form>
      <label for="username">Korisnicko ime</label>
      <input placeholder="Korisnicko␣ime" type="text" name="username" id="username
          ">
      <br>
```

```
        <label for="password">Lozinka</label>
        <input placeholder="Lozinka" type="password" name="password" id="password">
        <br>

        <input id="submit" type="submit" value="Prijava">
      </form>
    </div>

  </body>
  </html>
```



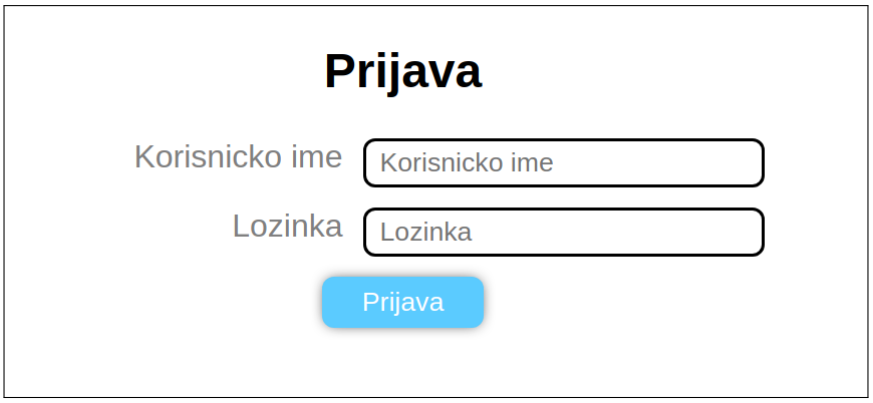
Slika 4: Jednostavni HTML obrazac za prijavu [autorski rad]

HTML sam po sebi služi isključivo za strukturiranje sadržaja - nije odgovoran za prezentaciju, odnosno izgled, samog sadržaja. U tu svrhu razvijen je drugi jezik koji, više od samog HTML-a, liči standardnom programskom jeziku, iako sam po sebi nije programski jezik - CSS (*Cascading Style Sheets*). CSS je, isto kao i HTML, jedna od tri temeljne tehnologije na kojoj je izgrađen Web. Najnovija verzija jezika je CSS3, a sam jezik usko je vezan uz HTML. Svaki HTML element može se, pomoću ogromnog broja CSS svojstava, stilizirati na specifičan način. Dva glavna HTML atributa koja pomažu u tome su *class* i *id*. Struktura CSS-a sastoji se od selektora, svaki od kojih selektira, odnosno označava, jedan ili više HTML elemenata ovisno o dva prije spomenuta HTML atributa. Svaki selektor u sebi sadrži jednu ili više definicija stilova koji će se primijeniti na svakom označenom HTML elementu.[6]

Prijašnji primjer HTML obrasca unaprijeđen je ovim CSS-om [autorski rad]:

```
body {
  font-family: sans-serif;
}

.title {
  text-align: center;
}

form {
  display: grid;
  grid-template-columns: 150px 200px;
  grid-template-rows: 1fr 1fr;
  gap: 10px;
  margin: auto;
  width: 360px;
}

label {
  text-align: right;
  color: gray;
}
```

```
input {
  padding: 3px 7px;
  border: 1.5px solid black;
  border-radius: 6px;
}

input[type=submit] {
  grid-column: 1 / span 2;
  justify-self: center;
  background-color: rgb(91, 203, 255);
  border: 0;
  padding: 5px 20px;
  border-radius: 6px;
  color: white;
  box-shadow: 0px 0px 7px -2px rgba(0,0,0,0.9);
}
```



Slika 5: Izgled HTML obrasca unaprijeđen CSS-om [autorski rad]

Treća tehnologija koja čini temelj modernog Weba je JavaScript. JavaScript je programski jezik opće namjene, a nastao je kao posljedica potrebe za dinamičnim Web sadržajem. Uglavnom se izvršava isključivo unutar Web preglednika. Danas postoje i drugačije varijante izvršavanja JavaScript koda izvan Web preglednika poput Node izvršnog okruženja. Popularnost JavaScript-a porasla je eksponencijalno od njegova prvog izdanja, a danas je jedan od najpopularnijih programskih jezika. Usko je povezan s prijašnje ostale dvije tehnologije, pogotovo što se tiče HTML-a. Sam jezik pruža direktan pristup modelu objekata dokumenta, takozvani DOM (*Document Object Model*), kojim je moguće u potpunosti upravljati. Na taj se način sa sadržajem dokumenta može raditi bilo što, a samo programiranje JavaScript-om temelji se na događajima. Primjerice, događaj može biti klik ili pritisak na neki gumb, ili micanje miša preko određenog elementa. Jezik se konstantno ažurira, pa je tako nadograđen s ograničenim pristupom raznim sustavima operacijskog sustava. Primjerice, razvojni programer može, pomoću JavaScript-a, od korisnika zatražiti pristup lokacijskim uslugama i dopuštenje za slanje sustavnih obavijesti. [6]

Iako se u pravilu odvija isključivo u jednoj dretvi, kod se često piše na asinkron način. To znači da se, primjerice, može poslati HTTP zahtjev na udaljeni poslužitelj bez da se blokira izvršavanje glavne dretve rada što znači da je ostatak aplikacije i dalje funkcionalan. U prijašnji primjer dodan je JavaScript kod kako bi se podaci obrasca slali na udaljeni poslužitelj, a odgovor poslužitelja obrađuje se na klijentskoj strani unutar Web preglednika [autorski rad]:

```
const loginButton = document.querySelector('#submit')
const usernameInput = document.querySelector('#username')
```

```
const passwordInput = document.querySelector('#password')

loginButton.addEventListener('click', () => {
  const username = usernameInput.value
  const password = passwordInput.value

  if (username && password) {
    const data = {
      username,
      password,
    }

    fetch('localhost:8080/login', {method: 'POST', body: data})
    .then(response => response.json())
    .then(json => {
      if (json.ok) {
        this.localStorage.setItem('user', json.user)
        window.location.replace('/home')
      }
    })
  }
  else {
    alert('Potrebno je ispuniti oba polja!')
  }
})
```

Osim standardnog JavaScript-a, užasno su popularni razni JavaScript razvojni okviri i biblioteke. Veći projekti se u pravilu nikad ne razvijaju bez nekog okvira ili veće biblioteke. Neki od danas najpopularnijih razvojnih okvira su: React, Angular, Vue i Svelte. Svaki od ovih okvira pruža slične funkcionalnosti za razvoj Web aplikacije, no svaki to pruža na svoj način. Svi navedeni okviri namijenjeni su razvoju korisničkog dijela Web aplikacije, odnosno korisničkog sučelja i poslovne logike koja se odvija isključivo na korisničkoj strani, unutar Web preglednika. Također postoje i razni JavaScript razvojni okviri i biblioteke za razvoj Web aplikacija na poslužiteljskoj strani.

3.2. Koraci razvoja

Razvoj Web aplikacije, kao i razvoj izvorne aplikacije, ima korake kojih se razvojni programer u pravilu drži. Ti koraci su u principu identični onima kod razvoja izvorne aplikacije, no imaju drugačije posljedice i drugačije utječu jedni na druge. Bitno je napomenuti kako se svaki Web preglednik, neovisno o platformi, u pravilu identično ponaša. Postoje Web standardi kojih se svaki Web preglednik treba držati i to osigurava da će se svaka Web aplikacija gotovo identično ponašati i izgledati na svakom Web pregledniku. Usprkos tomu, postoje sitne razlike u ponašanju različitih Web preglednika koje mogu do neke, ponekad i velike, mjere utjecati na razvijenu Web aplikaciju. Zbog toga se neki koraci, poput analize tržišta i odabira platforme za razvoj svode na to da se odabire Web preglednik kojeg će prosječni krajnji korisnik koristiti, a sama aplikacija se što više prilagođava tom Web pregledniku.

Razvoj ideje za samu aplikaciju je korak koji se nikako ne mijenja jer u pravilu ne ovisi previše o tome hoće li se razviti izvorna ili Web aplikacija. Naravno, svaka platforma za razvoj ima svoja ograničenja, pa tako ne bi imalo smisla razviti Web aplikaciju za nešto što nije prigodno pokretati unutar Web preglednika. Kao i kod izvorne aplikacije, ovaj korak do neke mjere utječe na sve ostale korake razvoja.

Analiza tržišta, kao i krajnjeg korisnika, puno manji utjecaj ima kod razvoja Web aplika-

cija. Svaki moderni Web preglednik dostupan je za korištenje na manje-više bilo kojoj platformi. Glavna razlika između platformi je zapravo to koji je Web preglednik postavljen kao zadani bez korisničkih promjena. Kod Apple platformi poput macOS i iOS operacijskih sustava zadani preglednik je Safari, dok je kod Windows platforme to Microsoft Edge. Odabir platforme za razvoj Web aplikacije u praksi ne postoji. Razlog je opisan u uvodu ovog potpoglavlja.

Odabir tehnologije na prvi se pogled može činiti nepotrebnim jer, kako je napomenuto ranije u poglavlju, postoje standardne tehnologije za razvoj Web aplikacija. Usprkos tomu postoji mnoštvo prilagođenih tehnologija koje pomažu kod razvoja Web aplikacija. Te tehnologije uvijek na kraju, kao rezultat izgradnje aplikacije, moraju generirati datoteke koje sadrže standardnih Web tehnologija. Odabir same tehnologije najviše se svodi na osobni odabir razvojnog programera, ovisno o njegovim preferencijama.

Razvoj aplikacije u principu je identičan razvoju izvorne aplikacije. Najveći utjecaj na sam razvoj Web aplikacije imaju odabrane tehnologije za razvoj jer se njima razvija aplikacija primjenom specifičnih metoda i konvencija razvoja. Nakon što je aplikacija razvijena, postoji mnogo opcija za distribuciju aplikacije. Distribucija aplikacija ovdje zapravo znači hosting aplikacije na nekom poslužitelju kojem javnost može pristupiti. Postoje posebne platforme koje se bave isključivo pojednostavljivanjem procesa hostinga. Pakiranje aplikacije, kao i sam razvoj, ovisi isključivo o odabranim tehnologijama za razvoj. Svaki razvojni okvir koristi svoju tehnologiju za pakiranje gotove aplikacije. Glavna točka ovdje je to da se pakirana aplikacija ne predaje krajnjem korisniku direktno, već ju korisnički Web preglednik interpretira i prikazuje na svoj način.

3.3. Glavne prepreke kod razvoja

Web aplikacije imaju potpuno drugačije prepreke od izvornih aplikacija. Kao što je već bilo napomenuto, glavna i najutjecajnija prepreka kod razvoja izvornih aplikacija je postojanje mnoštva različitih stolnih i mobilnih platformi za razvoj. Za svaku od tih platformi potrebno je učiti drugačije tehnologije za razvoj. Kod Web aplikacija ništa od ovog nije problem niti prepreka jer svaka Web aplikacija ovisi isključivo o Web pregledniku na kojem se odvija, a svaki Web preglednik se u pravilu drži dobro uspostavljenih standarda. Glavne prepreke kod razvoja web aplikacije svode se na nekoliko problema:

- specifične nepravilnosti Web preglednika
- nedostupnost temeljnih funkcionalnosti operacijskog sustava

Danas postoji mnogo različitih Web preglednika i, u pravilu, svi su kvalitetno razvijeni i održavani. Budući da se Web gradi na tri temeljne standardizirane tehnologije, svaki Web preglednik morao bi se držati tih standarda što bolje. Usprkos tome, svaki Web preglednik, bilo to namjerno ili slučajno, ne pridržava tih standarda u potpunosti. Razlog tome može biti greška u programskom kodu Web preglednika ili namjerna želja razvojnih programera da se Web preglednik tako ponaša. U pravilu te su nepravilnosti sićušne i utječu pretjerano na sam

razvoj Web aplikacije. Ponekad, doduše, te nepravilnosti mogu utjecati na razvoj do te mjere da je razvoj neke funkcionalnosti aplikacije potrebno promijeniti ili prilagoditi. Primjer nepravilnosti je nedostatak podrške za neko novije CSS svojstvo - jedan preglednik može dobro podržavati svojstvo, dok drugi preglednik uopće ne podržava isto svojstvo ili ga podržava s određenim greškama u prikazu. [6]

Web aplikacija se, za razliku od izvorne, ne odvija direktno na operacijskom sustavu računala. Svaki Web preglednik, kod pokretanja Web aplikacije, stvara pješčanik koji je izoliran od ostatka operacijskog sustava. Pješčanik se ponaša poput vrste virtualne mašine, a sam pješčanik se obično kreira iz sigurnosnih razloga. Posljedica toga je nedostupnost nekih temeljnih funkcionalnosti operacijskog sustava. Izvorna aplikacija može u potpunosti iskoristiti potpuni potencijal svog računalnog hardvera, dok Web aplikacija ne može. Osim toga, Web aplikacija ograničena je i uopće ne može pristupiti nekim podsustavima operacijskog sustava. [6]

4. Razvoj izvornih aplikacija korištenjem Web tehnologija

U ovom poglavlju biti će prikazani temeljni principi i tehnologije za razvoj izvornih stolnih i mobilnih aplikacija primjenom Web tehnologija. Osim toga, napraviti će se pregled progresivnih Web aplikacija (*PWA*) i općeniti pregled potencijalnih sigurnosnih problema kod razvoja izvornih aplikacija primjenom Web tehnologija.

4.1. PWA - korak do izvorne aplikacije

Progresivne Web aplikacije jedne su od najnovijih vrsta Web aplikacija. Iako se i dalje manje-više ponašaju identično standardnim Web aplikacijama, progresivne Web aplikacije posjeduju nekoliko bitnih svojstava koja ih uvelike razlikuju od njih. Glavno i najupečatljivije svojstvo progresivnih Web aplikacija je to što kako ih korisnik sve više koristi tako se one pretvaraju u aplikaciju koja sve više liči standardnoj izvornoj aplikaciji [7]. Primjer prilagodbe korisniku može biti smještanje prečaca za otvaranje progresivne Web aplikacije na radnu površinu operacijskog sustava. Iako se sama aplikacija i dalje otvara unutar Web preglednika, korisničko iskustvo korištenja same aplikacije više liči na korištenje izvorne aplikacije.

Najbitnija tehnologija za razvoj samih progresivnih Web aplikacija je takozvani uslužni radnik (engl. *service worker*). Ovo je temeljna tehnologija koja se nalazi u apsolutno svakoj progresivnoj Web aplikaciji. Glavna stvar koju omogućuje bilo koji uslužni radnik je izvršavanje programskog koda van standardnih okvira Web aplikacije. Točnije, to znači da bilo koji uslužni radnik nije direktno povezan s karticom unutar Web preglednika, već se odvija u pozadini, van samog Web preglednika. Svakog uslužnog radnika potrebno je prije korištenja registrirati, a svaki uslužni radnik može pristupiti, odnosno biti odgovoran, za specifičnu Web aplikaciju. Ovo otvara ogroman potencijal za bilo koju progresivnu Web aplikaciju. Primjerice, Web aplikacija može preko uslužnog radnika i dalje odrađivati neke učestale operacije. Na taj način Web aplikacija može funkcionirati čak i kada korisnik nema direktnu vezu na internet. [7]

Primjer jednostavnog uslužnog radnika [autorski rad]:

```
if ('serviceWorker' in navigator) {
  // Web preglednik podrzava usluzne radnike
  navigator.serviceWorker.register('usluzni_radnik.js').then(() => {
    // Usluzni radnik je registriran
  })
}

...

// unutar 'usluzni_radnik.js' datoteke

self.addEventListener('install', () => {
  // okida se kod instalacije usluznog radnika
})

self.addEventListener('fetch', (event) => {
  // okida se kod bilo kojeg standardnog HTTP zahtjeva Web preglednika
})
```

U ovom vrlo jednostavnom primjeru može se vidjeti na koji način se registrira uslužni

radnik unutar neke Web aplikacije. Sam kod uslužnog radnika nalazi se u zasebnoj datoteci. Programiranje uslužnog radnika temelji se na događajima koji se mogu okinuti u bilo koje vrijeme. Konkretno, u primjeru se stvaraju slušatelji na događaje instalacije samog uslužnog radnika i slanje, odnosno primanje, HTTP zahtjeva Web preglednika. Ovakvi događaji stvaraju temelj za daljnje proširivanje funkcionalnosti uslužnog radnika.

Što se tiče sigurnosne strane razvoja progresivnih Web aplikacija, središnje mjesto zauzima HTTPS protokol. HTTPS protokol zapravo je glavna sigurnosna zaštita za krajnjeg korisnika bilo koje Web aplikacije. Trenutno samo Web aplikacije koje se poslužuju preko sigurne HTTPS veze mogu registrirati i koristiti uslužne radnike. [7]

4.2. Najkorištenije tehnologije

Budući da se razvoj izvornih aplikacija Web tehnologijama svodi na korištenje spomenutih Web tehnologija, temeljne tehnologije za razvoj su identične. S druge strane, kako se ovdje zapravo radi o razvoju izvornih aplikacija, mora postojati neki međusloj koji "pretvara" Web aplikaciju u izvornu aplikaciju. Taj međusloj zapravo čine razne tehnologije koje bi se tako mogle nazvati temeljnim tehnologijama za razvoj izvornih aplikacija Web tehnologijama. U ovom će se potpoglavlju napraviti pregled nekoliko najpopularnijih takvih tehnologija.

Apache Cordova jedna je od tehnologija koje pružaju takav međusloj za razvoj izvornih aplikacija. Apache Cordova starija je tehnologija za razvoj izvornih mobilnih i stolnih aplikacija primjenom standardnih Web tehnologija. Jedna je od popularnijih tehnologija u svojoj kategoriji, no popularnost joj definitivno pada primarno zbog svoje starosti i promjena u trendovima razvoja aplikacija.

Apache Cordova podržava Windows, Android, iOS i MacOS operacijske sustave, odnosno platforme [8]. Ovo znači da pokriva veliku većinu modernih platformi, a pritom će svaka aplikacija razvijena Apache Cordova tehnologijom biti prisutna velikoj količini krajnjih korisnika. Što se tiče samog programskog koda, odnosno poslovne logike i razvoja korisničkog sučelja, temeljne tehnologije za razvoj su HTML, CSS i JavaScript. Unatoč tome, Apache Cordova ne ograničava razvojnog programera na odabir tehnologija. Drugim riječima, moguće je koristiti bilo koji zasebni razvojni okvir, pretprocesor i slično toliko dugo dok je krajnji rezultat čitavog razvoja isključivo HTML, CSS i JavaScript [8].

Jedna od najkorisnijih i unikatnih funkcionalnosti Apache Cordova tehnologije je podrška za predlošcima [8]. Predložak predstavlja temelj za izgradnju određene vrste aplikacije. Primjerice, postoje predlošci za izgradnju mobilne aplikacije sa standardiziranim početnim prikazom, tako da razvojni programer zapravo troši manje vremena na osnovne stvari kod početka razvoja. Osim toga, postoje i predlošci koji uvelike smanjuju vrijeme postavljanja projekta. Tako, primjerice, postoji predložak koji instalira sve potrebne pakete i ovisnosti kako bi se mogla razviti aplikacija primjenom Vue JavaScript okvira. Naravno, s relativnom lakoćom moguće je kreirati i vlastiti predložak za posebne potrebe.

React Native jedna je od novijih tehnologija za razvoj izvornih aplikacija primjenom Web tehnologija. Za razliku od Apache Cordova tehnologije, React Native može se koristiti isključivo

za razvoj mobilnih izvornih aplikacija. Tehnologija podržava isključivo Android i iOS platforme, što pokriva gotovo čitavo tržište mobilnih uređaja.

Sama React Native tehnologija temelji se na JavaScript programskom jeziku što znači da se sav programski kod potreban za razvoj aplikacije React Native tehnologijom također piše u JavaScript programskom jeziku. S obzirom na to, React Native stvara izoliranu okolinu za izvršavanje JavaScript koda na operacijskom sustavu mobilnog uređaja. Nadalje, stvara se most koji služi kao komunikacijski kanal između React Native JavaScript koda i samog operacijskog sustava. JavaScript kod se izvršava onakav kakav je, bez daljnjeg prevođenja ili pretvaranja u drugi format.

Veliki plus kod razvoja aplikacija React Native tehnologijom je to što je potrebno samo jednom napisati programski kod, a istovremeno se razvija izvorna aplikacija za dva najveća mobilna operacijska sustava. Još jedna pozitivna strana ove tehnologije je to što, uz određeno predznanje, postoji niska barijera za ulazak odnosno netko tko već poznaje React tehnologiju s lakoćom može početi razvijati aplikacije primjenom React Native tehnologije. S druge strane, često se spominju problemi s performansama aplikacija razvijenih React Native tehnologijom. S obzirom na to da postoji dodatan sloj između same aplikacije i operacijskog sustava, performanse aplikacije su do određene mjere smanjene. Ovo u pravilu ne stvara pretjerano veliki utjecaj na razvoj i korištenje aplikacije uzevši u obzir performanse današnjih mobilnih uređaja, no u određenim situacijama može stvarati velike probleme. U tom slučaju najbolje je koristiti neku izvornu tehnologiju za ciljani operacijski sustav jer ona pruža najbolje performanse.

React Native dijeli mnogo svojstava sa srodnom React tehnologijom. Isto kao i React, React Native razvijen je od strane Facebook-a, a primarno služi za izgradnju korisničkih sučelja. Usprkos tome što dijeli mnogo principa i svojstava s React tehnologijom, React Native je u srži potpuno drugačija tehnologija sa svojim skupom pravila, konvencija i ugrađenih funkcija i komponenti.

Neki od glavnih koncepata React Native tehnologije su komponente i svojstava (engl. props). Komponente su u samoj srži React Native tehnologije, a svaka komponenta predstavlja jedan element korisničkog sučelja. Ovisno o razvojnom programeru, React Native komponenta može biti nešto jednostavno poput poveznice koja ima nekakvo dodatno ponašanje ili pak složeni prikaz s mnogo ugniježđenih komponenti koje stvaraju veći dio aplikacije. Svojstva su usko vezana uz komponente, a koriste se tako da upotpunjuju komponente s potrebnim podacima, ili da promijene samo ponašanje komponente.

Electron je izrazito popularan razvojni okvir otvorenog koda. Služi za razvoj isključivo stolnih aplikacija Web tehnologijama. Temelji se na ostalim popularnim i pouzdanim tehnologijama otvorenog koda: Chromium i NodeJS. Chromium pruža arhitekturu procesa i upravljanje pojedinim pogledima, dok NodeJS pruža mnoštvo sučelja kojima se može direktno pristupati operacijskom sustavu na kojem se odvija aplikacija. Osim toga, NodeJS omogućuje razvojnom programeru pristup ogromnom broju javnih NodeJS paketa za razne funkcionalnosti i općenitu pomoć kod razvoja aplikacija. Što se tiče samog razvoja aplikacija, temeljne tehnologije koje Electron koristi su, naravno, HTML i CSS za izradu korisničkog sučelja i JavaScript za razvoj poslovne logike aplikacije.

Slično kao i Apache Cordova tehnologija, Electron ne nameće niti jednu posebnu tehnologiju za razvoj aplikacija. To znači da se za razvoj stolnih aplikacija Electron tehnologijom zapravo može koristiti bilo koja tehnologija koja se može koristiti za razvoj standardnih Web aplikacija unutar Web preglednika. Primjer toga je korištenje React biblioteke ili Angular razvojnog okvira uz jQuery i Lodash biblioteka za pomoćne funkcije. Dakle, ovo znači da, uz malu količinu učenja, Web razvojni programer može bez problema početi s razvojem stolnih aplikacija za bilo koju stolnu platformu.

Podržane platforme za razvoj su Windows, MacOS i Linux. Dakle, Electron podržava sve moderne stolne platforme što omogućuje razvojnom programeru da s lakoćom distribuira svoju aplikaciju svim potencijalnim korisnicima, neovisno o platformi.

Arhitektura same Electron tehnologije temelji se na Chromium arhitekturi. To znači da je svaki prozor aplikacije zapravo jedna kartica unutar Web preglednika. U Chromium arhitekturi, svaka kartica se zapravo odvija u jednom zasebnom procesu, dok se svim karticama (procesima) zapravo upravlja u glavnom procesu čitavog Web preglednika. To za Electron tehnologiju znači da će se svaki prozor odvija u svojem procesu, izoliran od ostalih, dok će uvijek postojati glavni proces aplikacije u kojem se upravlja ostalim prozorima.

Najčešća primjedba na Electron tehnologiju je korištenje više resursa nego što je to potrebno. U usporedbi s identičnom aplikacijom razvijenom korištenjem izvornih tehnologija za odabranu stolnu platformu, Electron aplikacije u prosjeku koriste značajno više resursa u pogledu radne memorije i procesorskih ciklusa. Dio krivice ovdje definitivno spada na razvojnog programera. Electron je posebna tehnologija s kojom treba znati baratati. Osim toga, Electron definitivno nije pogodan za razvoj bilo kakve stolne aplikacije. S obzirom na to da se sa svakom Electron aplikacijom pakira neka verzija Chromium tehnologije, jednostavnije aplikacije nije pogodno razvijati Electron tehnologijom. Loš primjer Electron aplikacije je upravitelj međuspremnika - aplikacija je jednostavna do te mjere da nije isplativo koristiti "težak" okvir za razvoj aplikacija kao što je to Electron. Sve što takva aplikacija treba raditi je spremati i pružati mogućnost čitanja povijesti međuspremnika. Korištenjem Electron okvira u ovom slučaju kao posljedicu stvara aplikaciju koja bespotrebno koristi više resursa poput memorije i procesorskih ciklusa, a sam paket aplikacije (pritom i veličina instalacije) je također znatno uvećan. [9]

4.3. Sigurnosni problemi kod razvoja izvornih aplikacija Web tehnologijama

Sigurnost aplikacija je glavna tema sadašnjice jer postoje razni načini koji žele tu sigurnost „srušiti“. Kako bi neka aplikacija bila sigurna mora se graditi tako da prolazi kroz četiri zahtjeva. [10] Prvi zahtjev je da se slijede sustavni procesi programskog inženjerstva. Svi procesi imaju određeni dio koji se bavi malim dijelovima sustava, te na kraju stvaraju cjelinu koja bi činila siguran sustav. Drugi zahtjev koji je koristan je da se provodi procjena rizika na svakom koraku izgradnju. Procjena rizika trebala bi se temeljiti na sigurnosnim odlukama koje će poboljšati sigurnost samog sustav. Treći zahtjev je da osoba koja provjerava sigurnost sustav ima ažurno znanje o svim sigurnosnim prijetnjama koje mogu zahvatiti sustav. Četvrti zahtjev je

da se osmisli sigurnosni mehanizam koji će braniti sustav od mogućih prijetnji koje ga zahvate. [10]

Što se tiče sigurnosti izvornih aplikacija, koje su razvijene primjenom Web tehnologija, može se reći kako je ona vrlo složena i uključuje više razmatranja nego što je potrebno za tradicionalnu aplikaciju. Kako bi ovakva aplikacija bila što sigurnija potrebno je razmatrati i poslužiteljsku i klijentsku stranu. Najvažnije je kod razvoja da se piše siguran kod jer je on najranjivija funkcionalnost cijele aplikacije. Napadači najviše napadaju sam kod aplikacije tako što ga mijenjaju i samim time cijela aplikacija više ne radi ono što joj je bila primarna svrha. Trebao bi se pisati što teži kod kojeg nije tako lako probiti. Također, sam kod bi trebalo ažurirati s vremena na vrijeme kako bi on postao snažniji. [12]

Treba biti oprezan i što se tiče korištenja biblioteka u aplikaciji. Obično ljudi koriste biblioteke koje je napisala neka osoba koja njima nije poznata, pa se ponekad pronađu i neke biblioteke koje nisu sigurne i napravljene su samo iz razloga da unište aplikaciju. Potrebno je provjeriti svaku biblioteku koja se koristi u aplikaciji kako bi se izbjegao napad. Enkripcija je isto dobar način kako bi se aplikacija mogla zaštiti od napada. Ukoliko se kriptiraju podaci manja je šansa da će nekome drugome biti korisni jer se ne mogu dohvatiti bez ključa osoba koja ih je kriptirala. Dakle, čak i ako dođe do napada neće biti velike štete jer napadač ne sadrži ključeve i ne može pristupiti aplikaciji. [10]

Napadi Cross-Site Scripting su vrsta napada pomoću napadači pokušavaju „ubrizgati“ svoje maliciozne skripte u aplikaciju kroz web stranicu. Ovakav napad je poznat kod stranica koje zahtijevaju unos od korisnika te nemaju provjeru za navedeni unos. Kako bi se izbjegli ovakvi napadi potrebno je napraviti provjere pri svakom unosu od korisnika. Postoji još jedan dobar sigurnosni način, a to je zamagljivanje koda. Zamagljivanje koda (eng. Code obfuscation) je proces modificiranja izvršne datoteke. Ovo je korisno je izvršna datoteka više neće biti korisna napadaču, ali će ostati potpuno funkcionalna. Kada napadač preuzme aplikaciju ona mu neće biti korisna, a aplikacija će biti izazovnija za napad.[10] Također, veliki nedostatak izvornih aplikacija koje su razvijene primjenom Web tehnologija je u komunikaciji sa serverom. Kada aplikacija komunicira sa servera najbolje je koristiti SSL (eng. Secure Sockets Layer) jer on štiti vezu između suradnika. [11]

Komunikacija između aplikacije i servera je jako bitna jer se šalju podaci koji su osjetljivi i nitko ne želi da ti podaci dođu u krive ruke. Kako bi se zaštitila aplikacija bilo bi najbolje da je razviju tehnike otkrivanja neovlaštenog pristupa. Ova metoda se koristi tako da se dobivaju obavijesti svaki put kada se kod aplikacije mijenja. Najbolje je imati dnevnik promjena koda kako bi se znalo koja osoba je kada mijenjala kod pa bi se lako otkrio napad. [12]

Nabrojene su bile neke sigurnosne mjere koje programer mora znati kako bi napravio što sigurniju aplikaciju. Posljednjih godina sigurnost aplikacija je dokazala svoju važnost i potrebno je kod razvoja aplikacija biti ažuran. Izvorne aplikacije koje su razvijene primjenom Web tehnologija zahtijevaju više truda oko svoj sigurnosti, ali u konačnici ako se dobro odrade svi aspekti može biti sigurna kao i neka standardna Web aplikacija.

5. Razvoj mobilnih aplikacija korištenjem *React Native* tehnologije

Kao što je već bilo spomenuto, React Native jedna je od najpopularnijih tehnologija za razvoj izvornih mobilnih aplikacija korištenjem standardnih Web tehnologija. Središnja i temeljna tehnologija koja se koristi unutar React Native tehnologije je React JavaScript biblioteka. Dakle, sama React Native tehnologija slična je standardnoj React biblioteci. S obzirom na to postoji podosta razlika između te dvije tehnologije, primarno zato jer je React Native tehnologija razvijena isključivo za razvoj mobilnih Android i iOS aplikacija. U ovom poglavlju biti će istraženi načini na koje React Native funkcionira, napraviti će se usporedba sa standardnom React Web bibliotekom i pregledati će se neki od korisnijih React Native sučelja za pristup operacijskom sustavu.

5.1. Usporedba s Web ekvivalentom

Web ekvivalenta React Native tehnologije je React JavaScript biblioteka. Osim toga, React biblioteka je također i temeljna tehnologija React Native tehnologije. Kao takva, React Native tehnologija dijeli mnoštvo koncepata s React bibliotekom. S druge strane, ove dvije tehnologije su potpuno drugačije u kontekstu vrsta aplikacija koje se mogu njima razviti i načina izgradnje razvijene aplikacije.

Temeljni koncept obje tehnologije definitivno je koncept komponenti. Bilo koja React ili React Native komponenta zapravo je obična JavaScript funkcija. Kao takva, u tijelo te funkcije može se pisati bilo kakav JavaScript kod. Ono po čemu je React komponenta posebna je to što funkcija mora vraćati jedan poseban tip podatka. Taj tip podatka predstavlja strukturu, odnosno vizualni izgled React komponente. Taj poseban tip podatka često se naziva JSX - akronim koji je nastao kao kombinacija akronima za JavaScript programski jezik i prvog slova u XML akronimu. Dakle, JSX bi se mogao shvatiti kao poseban jezik za označavanje koji u pozadini koristi JavaScript logiku kako bi mogao prikazati gotovo bilo što, uključujući i druge React komponente. [13]

Primjer jednostavne React komponente [autorski rad]:

```
import React from 'react'

const SimpleComponent = () => {
  return (
    <div>
      <h1>Jednostavna komponenta</h1>
    </div>
  )
}

export default SimpleComponent
```

S druge strane, React Native komponente se, za razliku od standardnih React komponenti, ne izgrađuju od temeljnih HTML elemenata. Za React Native postoje izvorne komponente koje se u pravilu mogu koristiti za bilo koji operacijski sustav. Nekolicina izvornih React Native komponenti specifična je za pojedini operacijski sustav, bio to Android ili iOS. Neke od najkori-

štenijih React Native komponenti su: *Text*, *View* i *Image*. Većina njih je poprilično jednostavna, slično kao što su i HTML elementi. To je tako po dizajnu same tehnologije - lakše je razviti vlastite i specifične komponente korištenjem nekoliko jednostavnih ugrađenih komponenti. *Text* React Native komponenta specifična je po tome što se samo unutar nje može smjestiti samostalni tekst. U pravilu ne koristi se za ništa drugo nego za prikazivanje teksta koji se također može stilizirati. Budući da je toliko jednostavna, ne podržava neka posebna svojstva. *View* React Native komponenta najbolje bi se mogla usporediti standardnom *div* HTML elementu. Sama po sebi, *View* komponenta služi samo kao spremnik za ostale komponente. Slično kao i *Text* komponenta, *View* komponenta jako je jednostavna i ne podržava neka posebna svojstva, osim svojstva za stiliziranje i nekoliko rijetko korištenih događaja. Kao što i samo ime komponente govori, *Image* React Native komponenta koristi se za prikazivanje slika u grafičkom sučelju. Naravno, najbolje bi se mogla usporediti s *img* HTML elementom. Najbitnije svojstvo *Image* komponente je *source* koje određuje koja će se slika prikazati unutar komponente. [14] [13]

Primjer jednostavne React Native komponente [autorski rad]:

```
import React from 'react'

const SimpleComponent = () => {
  return (
    <View>
      <Text>
        Jednostavna komponenta
      </Text>
    </View>
  )
}

export default SimpleComponent
```

React i React Native komponente su same po sebi poprilično statične - u pravilu uvijek rade jednu stvar bez velikih varijacija. Kako bi se ta monotonost razbila, postoji još jedan temeljni React koncept usko vezan uz React komponente. Taj koncept se često naziva svojstvima React komponente (engl. *properties*, skraćeno *props*). Svojstva React komponente prosljeđuju se samoj komponenti, gotovo identično kao i kod HTML elemenata, tako da se u JSX element proslijede atributi u obliku *ključ=vrijednost*. Sva proslijeđena svojstva akumuliraju se u jedan jedini ulazni parametar svake React komponente. Taj parametar se po konvenciji često naziva *props*. Vrijednost bilo kojeg svojstva React komponente može biti bilo što: znak, broj, polje, objekt, funkcija, React komponenta i ostalo. Sve što vrijedi za svojstva React komponenti vrijedi i za svojstva React Native komponenti. [14]

Kombinacijom React komponenti i njihovih svojstava mogu se razviti proširive komponente s velikim potencijalom za višestruku ponovnu iskoristivost komponente. Primjer prijašnje React i React Native komponente proširene korištenjem posebnih svojstava [autorski rad]:

```
// Standardna React komponenta
import React from 'react'

const SimpleComponent = ({ name, favoriteColor }) => {
  return (
    <div>
      <h1>Bok, moje ime je {name}</h1>
      <p>Moja najdraza boja je {favoriteColor}</p>
    </div>
  )
}
```

```
export default SimpleComponent
```

```
// React Native komponenta
import React from 'react'

const SimpleComponent = ({ name, favoriteColor }) => {
  return (
    <View>
      <Text>
        Bok, moje ime je {name}
      </Text>

      <Text>
        Moja najdraza boja je {favoriteColor}
      </Text>
    </View>
  )
}

export default SimpleComponent
```

Programiranje JavaScript jezikom često se svodi na programiranje raznih događaja unutar same aplikacije. Ti događaji mogu predstavljati mnoštvo stvari, neke od kojih su učitavanje elementa, unos podatka u obrazac od strane krajnjeg korisnika i klik na gumb. Veliki dio React, pa tako i React Native, programiranja također se svodi na programiranje tih istih događaja. Svi ugrađeni JavaScript događaji također su dostupni unutar React tehnologije, dok React Native pruža posebne događaje prilagođene za mobilne uređaje. Koriste se slično kao i posebna svojstva komponente, u obliku *nazivDogađaja={referenca na funkciju}*.

Primjer prijašnje React i React Native komponente proširene rukovanjem događajima [autorski rad]:

```
// Standardna React komponenta
import React from 'react'

const SimpleComponent = ({ name, favoriteColor }) => {

  const alertColor = () => {
    alert(favoriteColor)
  }

  return (
    <div>
      <h1>Bok, moje ime je {name}</h1>
      <p>Moja najdraza boja je {favoriteColor}</p>
      <button onClick={alertColor}>Klikni me!</button>
    </div>
  )
}

export default SimpleComponent
```

```
// React Native komponenta
import React from 'react'
import { Alert } from 'react-native'

const SimpleComponent = ({ name, favoriteColor }) => {

  const alertColor = () => {
    Alert.alert(favoriteColor)
  }

  return (
    <View>
      <Text>
        Bok, moje ime je {name}
      </Text>

      <Text>
        Moja najdraza boja je {favoriteColor}
```

```
        </Text>

        <Button onPress={alertColor} title="Klikni me!" />
      </View>
    )
  }

  export default SimpleComponent
```

Posljednji, i najnoviji koncept kojeg dijele React i React Native tehnologije su takozvane React "kopče" (engl. *hooks*). Naziv su dobile po tome što omogućuju razvojnom programeru da "prikvači" svoje komponente u samu srž React tehnologije. To znači da razvojni programer ima pristup funkcionalnostima kojima inače ne bi imao. Primjer toga je *useState* React kopča koja omogućuje pojedinoj React komponenti da pamti vrijednosti vezane uz nju kroz cijeli njezin životni ciklus. Primjer prijašnje React i React Native komponente proširene React kopčom [autorski rad]:

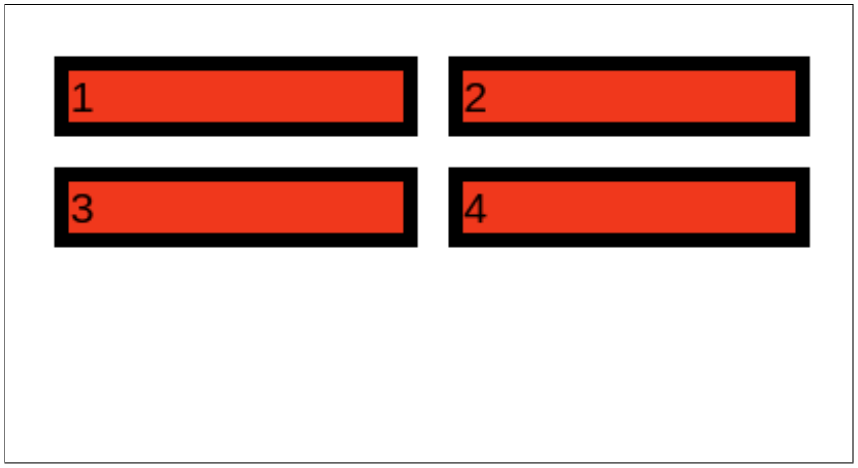
```
  // Standardna React komponenta
  import React, { useState } from 'react'

  const SimpleComponent = ({ name }) => {

    const [color, setColor] = useState('crvena')

    const selectRandomColor = () => {
      const colors = ['crvena', 'plava', 'zelena', 'zuta']
      const c = colors[Math.floor(Math.random() * colors.length)];

      setColor(c)
    }

    return (
      <div>
        <h1>Bok, moje ime je {name}</h1>
        <p>Moja najdraza boja je {color}</p>
        <button onClick={alertColor}>Klikni me!</button>
      </div>
    )
  }

  export default SimpleComponent
```

```
  // React Native komponenta
  import React , {useState }, from 'react'

  const SimpleComponent = ({ name }) => {

    const [color, setColor] = useState('crvena')

    const selectRandomColor = () => {
      const colors = ['crvena', 'plava', 'zelena', 'zuta']
      const c = colors[Math.floor(Math.random() * colors.length)];

      setColor(c)
    }

    return (
      <View>
        <Text>
          Bok, moje ime je {name}
        </Text>

        <Text>
          Moja najdraza boja je {color}
        </Text>

        <Button onPress={selectRandomColor} title="Klikni me!" />
      </View>
    )
  }

  export default SimpleComponent
```

React biblioteka namijenjena je za korištenje na Web platformi, tako da je i podrška za prilagođene stilove također namijenjena za korištenje na Web platformi. Drugim riječima, radi se o Web standardnoj CSS tehnologiji. React podržava CSS stilove u nekoliko različitih oblika, no najčešći oblik stiliziranja komponenti je uvozom stilova iz vanjske CSS datoteke. Osim toga, moguće je stilove pisati direktno u pojedinu komponentu, odnosno element. Rijetko korištena metoda stiliziranja React komponenti je korištenje posebnog *style* JSX elementa unutar kojeg se direktno može pisati CSS.

S druge strane, React Native tehnologija koristi poseban način stiliziranja svojih komponenti. Svi stilovi direktno se pišu unutar JavaScript datoteke, dakle nema direktne podrške za CSS tehnologijom. Usprkos tomu, sintaksa i samo ponašanje stilova unutar React Native tehnologije izrazito su slični standardnoj Web CSS tehnologiji. Primjer jednostavne React Native komponente koja posjeduje posebne stilove [autorski rad]:

```
import React from 'react'
import { StyleSheet, View, Text } from 'react-native'

const style = StyleSheet.create({
  container: {
    padding: '20px'
  },
  grid: {
    display: 'grid',
    gridTemplateColumns: '1fr 1fr',
    gridTemplateRows: '1fr 1fr',
    gap: '10px'
  },

  element: {
    backgroundColor: 'red',
    border: '5px solid black'
  }
})

const StyleApp = () => {
  return (
    <View style={style.container}>
      <View style={style.grid}>
        <View style={style.element}><Text>1</Text></View>
        <View style={style.element}><Text>2</Text></View>
        <View style={style.element}><Text>3</Text></View>
        <View style={style.element}><Text>4</Text></View>
      </View>
    </View>
  )
}

export default StyleApp
```



Slika 6: React Native komponenta s prilagođenim stilovima [autorski rad]

5.2. Načini kreiranja grafičkog sučelja

Što se tiče kreiranja grafičkog sučelja aplikacije, React Native u principu je identičan React biblioteci. Glavna ideja iza React tehnologije je korištenje JavaScript jezika za deklaraciju raznih komponenti, odnosno elemenata, grafičkog sučelja apsolutno neovisno o platformi. Glavni i najčešće korišteni, no ne i jedini, način deklaracije grafičkog sučelja unutar React tehnologije je prije spomenuti JSX jezik za označavanje koji kombinira JavaScript programski jezik s generičkim jezikom za označavanje nalik XML-u. Neovisnost od platforme drugi je najbitniji princip kojeg se React i React Native tehnologije drže. Zapravo je glavna razlika između React i React Native tehnologije to na koji način, odnosno u koji krajnji oblik, tehnologija pretvara dati JSX zapis. Sva ostala logika, izuzev pozivima specifičnim sučeljima vezanih uz određenu platformu, je pisana u JavaScript jeziku i isti se programski kod može bez problema koristiti u obje tehnologije. [14][15]

Sličnost između tehnologija nastavlja se kod konkretnog načina generiranja finalnog grafičkog sučelja, kao i kod manipulacija tog sučelja. React i React Native tehnologije obje stvaraju takozvani "virtualni DOM" (engl. *virtual Document Object Model*). Virtualni DOM zapravo predstavlja međusloj koji se nalazi između izvornog sustava, bio to Web preglednik ili operacijski sustav mobilnog telefona, i samog JavaScript i JSX koda. Ovaj međusloj postoji primarno kako bi unaprijedio performanse izvršavanja programskog koda. Poznata je činjenica da JavaScript nije najbolji programski jezik što se tiče brzine izvršavanja i općenitih performansi. Najskuplje su one operacije koje se upravo najviše koriste kod generiranja i manipulacijama grafičkog sučelja - upravo onaj zadatak kojeg imaju React i React Native tehnologije. Iako su performanse odličan bonus, potencijalno bitniji bonus ovakvog modeliranja sučelja je mogućnost stvaranja snažnih apstrakcija. Što to zapravo znači je da se temeljna React tehnologija može koristiti za generiranje raznih vrsta grafičkih sučelja. Budući da je React starija od ovih dviju tehnologija, virtualni DOM modeliran je po uzoru na standardni DOM unutar Web preglednika. Usprkos tomu, isti taj model moguće je pretvoriti u bilo kakav drugi model grafičkog sučelja. U konkretnom slučaju React Native tehnologije radi se o pretvaranju virtualnog DOM modela u modele koji su pogodni za prikaz na Android i iOS operacijskim sustavima. [14][16]

5.3. Pristup operacijskom sustavu i njegovim funkcionalnostima

Kako bi se aplikacije razvijene React Native tehnologijom mogle smatrati izvornim mobilnim aplikacijama, React Native mora moći pružati pristup izvornom sustavu i njegovim funkcionalnostima. React Native ima nekolicinu ugrađenih sučelja kojima se može pristupiti tim funkcionalnostima. Neka od njih su: *Alert*, *Vibration*, *Share* i *Platform* [15].

Alert sučelje omogućuje razvojnom programeru da otvori poseban dijaloški prozor, izgled i ponašanje kojeg ovisi o platformi. Ovo sučelje obično se koristi kao dodatna potvrda od strane korisnika za neku akciju. Primjer jednostavne React Native komponente koja koristi *Alert* sučelje [17][autorski rad]:

```
import * as React from 'react'
import { Button, View, Alert } from 'react-native'

export default function App() {
  const alertFunction = () => {
    Alert.alert(
      'Jeste li sigurni?',
      '',
      [
        {
          text: 'Da',
          onPress: () => Alert.alert('Pritisnut je "Da"')
        },
        {
          text: 'Ne',
          onPress: () => Alert.alert('Pritisnut je "Ne"')
        }
      ],
    )
  }

  return (
    <View>
      <Button
        title={"Klikni me!"}
        onPress={alertFunction} />
    </View>
  )
}
```

Komponenta iz primjera u sebi sadrži jedan gumb. Kad se klikne na gumb, otvara se dijaloški prozor s dvije opcije: "Da" i "Ne". Klikom na bilo koju opciju otvara se drugi dijaloški prozor u kojem se potvrđuje koja opcija je odabrana.

Vibration sučelje omogućuje razvojnom programeru da pokrene vibraciju na mobilnom telefonu, ukoliko uređaj podržava vibraciju. Sučelje pruža funkciju pod nazivom vibrate, a funkcija može kao parametar primiti nekoliko stvari. Najjednostavniji parametar je broj koji označava broj milisekundi koliko će dugo uređaj vibrirati. Osim toga, funkcija može kao parametar primiti i polje koje predstavlja uzorak od više uzastopnih vibracija, a svaku vibraciju ponovo predstavlja broj milisekundi koji označava trajanje vibracije. Primjer jednostavne React Native komponente koja koristi *Vibration* sučelje [17][autorski rad]:

```
import * as React from 'react';
import { Button, View, Vibration } from 'react-native';

export default function App() {
  const vibrateFunction = () => {
    Vibration.vibrate(2000)
  }
```

```
    const stopVibrate = () => {
      Vibration.cancel()
    }

    return (
      <View>
        <Button
          title={"Vibriraj!"}
          onPress={vibrateFunction} />

        <Button
          title={"Prestani vibrirati!"}
          onPress={stopVibrate} />
      </View>
    )
  }
```

Komponenta iz primjera sadrži dva gumba. Klikom na prvi gumb okida se funkcija koja pokreće vibraciju u trajanju od dvije tisuće milisekundi. Klikom na drugi gumb prekida se bila kakva vibracija ukoliko je prije bila pokrenuta.

Share sučelje specifično je po tome što nema direktne veze sa samom aplikacijom koja se razvija, već pruža jednostavan način dijeljenja sadržaja nekoj drugoj aplikaciji. Sučelje razvojnom programeru pruža jednu funkciju pod nazivom share koja, ovisno o platformi, otvara specifičan izbornik gdje se može odabrati odredišna aplikacija dijeljenog sadržaja.

```
  import * as React from 'react'
  import { Button, View, Share, Alert } from 'react-native'

  export default function App() {
    const shareFunction =  async () => {
      const result = await Share.share({
        message: 'Sadrzaj za dijeljenje',
      })

      if (result.action === Share.sharedAction) {
        Alert.alert('Sadrzaj podijeljen!')
      } else if (result.action === Share.dismissedAction) {
        Alert.alert('Sadrzaj nije podijeljen!')
      }
    }

    return (
      <View>
        <Button
          title={"Klikni me!"}
          onPress={shareFunction} />
      </View>
    )
  }
```

Platform sučelje daje pristup temeljnim informacijama o platformi, odnosno uređaju na kojem se izvršava aplikacija. Informacije daje u obliku jednog JavaScript objekta koji sadrži korisne i često korištene informacije o uređaju. Osim toga, Platform sučelje pruža korisnu select metodu koja omogućuje selekciju na temelju platforme uređaja. Dakle, tom se metodom primjerice može izvršiti jedna naredba ukoliko se radi o Android operacijskom sustavu, ili druga naredba ukoliko se radi o iOS operacijskom sustavu.

```
import React from 'react'
import { Platform, Button, Vibration, Alert, Text, View } from 'react-native'

const App = () => {
  const onButtonClick = () => {
    const os = Platform.OS
    if (os === 'android') {
      Vibration.vibrate(500)
    }
    else {
      Vibration.vibrate(2000)
```

```
    }
    Alert.alert(`Koristite ${os} operacijski sustav!`)
  }
  return (
    <View style={styles.container}>
      <Button title="Klikni me!" onPress={onButtonClick}/>
    </View>
  );
};

export default App
```

Komponenta iz primjera sadrži u sebi jedan gumb. Klikom na taj gumb okida se funkcija koja prvo iz *Platform* sučelja dohvaća i sprema podatak o kojem se operacijskom sustavu izvršava aplikacija. Moguće vrijednosti su "android" i "ios". Nakon toga, provjerava se vrijednost varijable i pokreće vibracija uređaja, dok je trajanje vibracije ovisno o operacijskom sustavu uređaja.

6. Razvoj stolnih aplikacija korištenjem *Electron* tehnologije

Kako React Native tehnologija služi za izradu izvornih mobilnih aplikacija, tako Electron služi za izradu izvornih stolnih aplikacija. Electron je JavaScript razvojni okvir za razvoj izvornih stolnih Windows, MacOS i Linux aplikacija. Za razvoj korisničkog sučelja aplikacije, Electron tehnologija podržava standardne Web tehnologije: HTML i CSS. U ovom će poglavlju biti napravljen općeniti pregled Electron tehnologije i usporedba s razvojem standardnih Web aplikacija. Osim toga, biti će istraženo kako sama tehnologija funkcionira i kako pristupa funkcionalnostima operacijskog sustava.

6.1. Usporedba s Web ekvivalentom

Za razliku od React Native tehnologije, Electron tehnologija ne temelji se direktno na nekoj drugoj tehnologiji kao što je to slučaj kod React Native tehnologije. Glavna ideja Electron tehnologije je dati razvojnom programeru potpun odabir Web tehnologija koje želi koristiti za razvoj izvorne stolne aplikacije. Stoga Electron tehnologija pruža Web platformu koja se može izgraditi na način da stvori stolnu aplikaciju. S obzirom na sve navedeno, jedina Web tehnologija s kojom valja usporediti Electron tehnologiju je standardan JavaScript. Sve usporedbe mogle bi se dalje povući za bilo koji JavaScript okvir ili biblioteku.

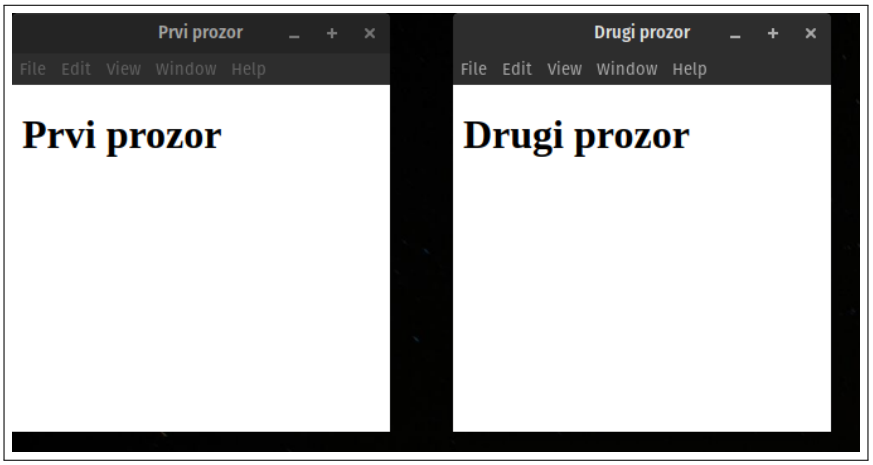
Otvaranje novog prozora jedna od temeljnih i najčešće korištenih radnji kod razvoja stolnih aplikacija Electron tehnologijom. Standardna Web aplikacija koja se izvršava unutar Web preglednika uopće nema funkcionalnost otvaranja novog prozora - čitava aplikacija ograničena je na jednu jedinu karticu Web preglednika. Ovo naravno stvara ogromno ograničenje kod razvoja Web aplikacija, bilo to programiranje same aplikacija ili dizajn korisničkog sučelja. S obzirom na to da je ovo ograničenje direktno vezano uz Web preglednik i Web tehnologije, Electron kao takav također nasljeđuje to ograničenje. Usprkos tomu, Electron ima ugrađeno rješenje za ovaj problem. Svaka Electron aplikacija razvijena je pomoću posebnog procesnog modela u kojem postoji jedan glavni proces koji upravlja s više sporednih procesa. Ovaj model će detaljnije biti odrađen u sljedećem potpoglavlju. [18] [19]

U pravilu svaka Electron aplikacija započinje otvaranjem početnog prozora. Taj prozor može primjerice biti prozor za prijavu, odnosno registraciju, ili općeniti središnji prozor aplikacije u kojem je moguće otvoriti sve ostale prozore i pristupiti svim funkcionalnostima aplikacije. Kako bi se otvorio novi Electron prozor potrebno je ispuniti nekoliko preduvjeta. Prvo, potrebno je kreirati objekt tipa *BrowserWindow*. Taj objekt, kao što i ime govori, predstavlja referencu na jedan prozor aplikacije. U konstruktor objekta potrebno je proslijediti neke osnovne informacije o prozoru, poput njegove visine i širine u pikselima. Osim toga, u konstruktor moguće je proslijediti i neke opcionalne parametre poput putanje do JavaScript datoteke koja će se izvršiti prije samog učitavanja, odnosno prikazivanja, prozora. Nakon kreiranja samog objekta otvara se prazan prozor bez ikakvog sadržaja. Objekt prozora prazna je šablona koju je potrebno upotpuniti nekom HTML datotekom kako bi se mogao prikazati neki konkretni sadržaj. [20] [21]

Jednostavni primjer otvaranja prozora korištenjem Electron razvojnog okvira [autorski rad]:

```
const { BrowserWindow } = require('electron')

const firstWindow = new BrowserWindow({
  width: 300,
  height: 300,
  x: 0,
  y: 0
})

const secondWindow = new BrowserWindow({
  width: 300,
  height: 300,
  x: 0,
  y: 350
})

firstWindow.loadFile('firstWindow.html')

secondWindow.loadFile('secondWindow.html')
```

Izvršavanjem ovog programskog koda nastaju dva prozora visine i širine od 300 piksela. Prvi prozor nalazi se u gornjem lijevom kutu, dok se drugi prozor nalazi desno od prvog. U svaki od prozora se zatim učitavaju odgovarajuće HTML datoteke.



Slika 7: Otvaranje prozora Electron tehnologijom [autorski rad]

Kao što je već prije bilo spomenuto, JavaScript programiranje se često temelji na programiranju raznih događaja koji se mogu dogoditi tokom izvršavanja i korištenja aplikacije. Naravno, isto bi se moglo reći i za Electron razvojni okvir. Jedna razlika kod Electron tehnologije je to što posjeduje poseban skup događaja koji je smješten unutar posebnog modula svake aplikacije. Taj modul koristi se u svakoj kompleksnijoj Electron aplikaciji, a naziv mu je, naravno, *app*. Ovaj modul podržava ogroman broj događaja koji su direktno vezani uz samu aplikaciju, odnosno glavni proces aplikacije. Neki od tih događaja vezani su uz specifičan operacijski sustav, dok ih je većina univerzalno, odnosno nepovezano s bilo kojim specifičnim operacijskim sustavom. Neki od korisnijih događaja koji se često koriste kod razvoja Electron aplikacija su *ready*, *activate*, *will-quit* i *window-all-closed*. Događaj *ready* okida se samo jednom i to kada je aplikacija završila s početnom inicijalizacijom. Ovaj događaj koristi se u gotovo svakoj Electron aplikaciji, a najčešća primjena je otvaranje početnog prozora nakon što je aplikacija postala spremna za korištenje. Događaj *activate* okida se u nekoliko slučaja: aplikacija se po prvi puta

pokreće, pokretanje aplikacija kad je ona već pokrenuta, klik na ikonu aplikacije. Događaj *will-quit* okida se nakon što su zatvoreni svi prozori aplikacije što obično označava da se aplikacija treba zatvoriti. Slično prijašnjem događaju, *windows-all-closed* okida se kad se zatvore svi prozori aplikacije, no okida se prije *will-quit* događaja. [20]

Kratki primjer Electron aplikacije koja koristi neke događaje [autorski rad]:

```
const { app } = require('electron')

app.on('before-quit', () => {
  // okida se prije zatvaranja aplikacije
  // obicno se koristi za zatvaranje otvorenih resursa
})

app.on('window-all-closed', () => {
  console.log('svi prozori aplikacije zatvoreni')
})

app.on('activate', () => {
  console.log('aplikacija aktivirana')
})

app.on('ready', () => {
  console.log('aplikacija spremna za koristenje')
})

app.on('open-file', () => {
  console.log('korisnik zeli otvoriti datoteku s racunala')
})
```

Osim raznih događaja, *app* Electron modul pruža i velik broj funkcija kojima se može upravljati samom aplikacijom. Slično kao i kod događaja, neke funkcije se mogu koristiti samo na specifičnim operacijskim sustavima, dok su ostale neovisne o operacijskom sustavu. Neke češće korištene funkcije *app* modula su: *quit*, *relaunch* i *whenReady*. Funkcija *quit* služi za zatvaranje svih prozora aplikacije, pa tako i same aplikacije. Za razliku od jako slične ali rigoroznije *exit* funkcije, *quit* funkcija prvo provjerava mogu li se svi prozori zatvoriti bez izazivanja problema. Funkcija *relaunch* u principu ponovno pokreće aplikaciju, uz jedan uvjet. Taj uvjet je da se aplikacija prvo mora zatvoriti pozivom prije spomenutih funkcija. U *relaunch* funkciju je moguće proslijediti opcionalne parametre koji definiraju ulazne parametre nove instance aplikacije koja će se otvoriti. Korištenjem funkcije *whenReady* moguće je pozvati određenu funkciju koja će se pozvati kada je Electron aplikacija završila s inicijalizacijom i spremna je za korištenje, a često se koristi kao alternativa za *ready* događaj. [20]

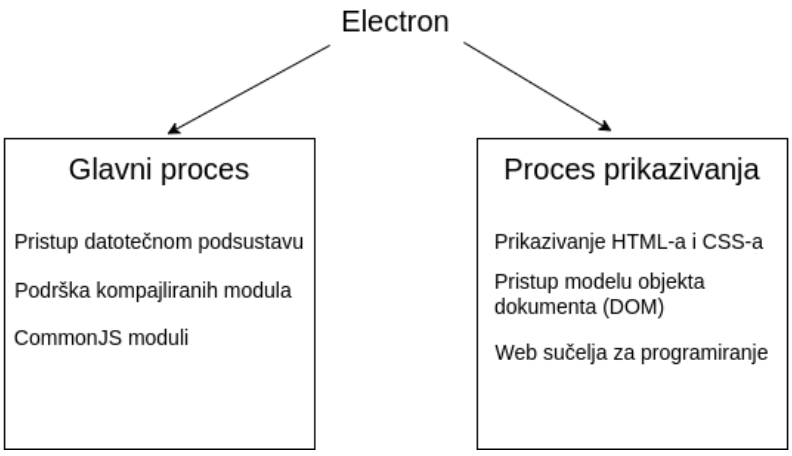
Kratki primjer Electron aplikacije u kojoj se koriste neke funkcionalnosti *app* modula [autorski rad]:

```
const { app } = require('electron')

const openStartWindow = () => {
  const window = new BrowserWindow({
    width: 300,
    height: 300
  })

  window.loadFile(start.html)
}

app.on('window-all-closed', () => {
  console.log('svi prozori aplikacije zatvoreni')
  app.quit()
})

app.on('activate', () => {
  console.log('aplikacija aktivirana')
})
```

```
app.on('ready', () => {
  console.log('aplikacija spremna za koristenje')
  openStartWindow()
})
```

6.2. Načini kreiranja grafičkog sučelja

Svaka Electron aplikacija izvršava se u dvije vrste procesa: glavni proces i proces prikazivanja. Glavni proces aplikacije je samo jedan, dok procesa prikazivanja može biti više. Broj procesa prikazivanja određen je brojem otvorenih prozora unutar aplikacije. Konkretno, svaki otvoreni prozor unutar aplikacije upravlja jedan proces prikazivanja. Glavni proces može komunicirati sa svakim procesom prikazivanja, a svaki proces prikazivanja može komunicirati s glavnim procesom. Procesi prikazivanja međusobno ne mogu komunicirati. Primarni zadaci glavnog procesa su upravljanje životnim ciklusom aplikacije, otvaranje novih prozora, odnosno pokretanje procesa prikazivanja i pružanje raznih NodeJS modula procesima prikazivanja. Osim toga, glavni proces može direktno pristupiti svim ugrađenim i dodatno instaliranim NodeJS modulima i paketima, što otvara vrata velikih mogućnosti. S druge strane, glavni zadatak procesa prikazivanja je prikazivanje sadržaja određenog prozora aplikacije. To znači da proces mora moći prikazivati HTML sadržaj uređen CSS stilovima. S obzirom na to, svaki se prozor Electron aplikacije zapravo temelji na Chromium tehnologiji. Chromium je tehnologija otvorenog koda za razvoj Web preglednika razvijena od strane Google-a. Chromium se koristi kao temelj za mnogo modernih Web preglednika, a implementacija JavaScript stroja unutar Web preglednika je jako dobra. Electron tehnologija upravo zato i koristi Chromium tehnologiju kao temelj za prikazivanje sadržaja aplikacije. Kod koji se izvršava unutar procesa prikazivanja ponaša se kao da se izvršava u nekom Web pregledniku. [22][23]



Slika 8: Dijagram Electron modela procesa [23]

Prednost ovakvog modela je to što je moguće upotpuniti okruženje Web preglednika s moćima standardnih izvornih aplikacija. Ovo se može postići iz razloga što su glavni proces i procesi prikazivanja povezani. Glavni proces ima pristup izvornim sučeljima koje pruža NodeJS okruženje za izvršavanje JavaScript koda, dok procesi prikazivanja posjeduju svojstva

najmodernijih Web preglednika. Kombiniranjem ovakva dva procesa dobije se aplikacija koja se izvršava u i poštuje sva pravila okruženja Web preglednika, a pritom ima i sve funkcionalnosti standardne izvorne aplikacije. Primarni način na koji se ovo može postići je korištenjem takozvane "predučitane skripte" (engl. *preload script*). Kod unutar ove JavaScript skripte izvršava se prije otvaranja novog prozora. Svaki prozor može imati svoju skriptu, a skripta se određuje opcionalnim parametrom unutar konstruktora *BrowserWindow* klase. Iako se kod unutar skripte izvršava unutar konteksta novog prozora, taj kod ima pristup svim izvornim NodeJS modulima. Ti moduli mogu se izložiti tako da i programski kod koji se izvršava unutar procesa prikazivanja ima pristup njima. Poveznica između skripti preko koje se ti moduli mogu izložiti je globalni objekt prozora (engl. *window*). Kako bi se to izlaganje odvilo u što sigurnijem okruženju, Electron pruža poseban *contextBridge* modul koji u sebi sadrži funkcije za izlaganje bilo čega procesu prikazivanja. [23][24]

Kratki primjer izlaganja nekih NodeJS izvornih modula procesu prikazivanja [20]:

```
// glavni proces

const {app, BrowserWindow} = require('electron')

function createWindow () {
  const firstWindow = new BrowserWindow({
    width: 300,
    height: 300,
    webPreferences: {
      preload: `${__dirname}/preload.js`,
    }
  })

  firstWindow.loadFile('firstWindow.html')
}

app.whenReady().then(() => {
  createWindow()
})


//preducitana skripta

const { contextBridge } = require('electron')
const fs = require('fs')

contextBridge.exposeInMainWorld('node', {
  fs
})


// proces prikazivanja

const { fs } = window.node
// fs NodeJS modul je dostupan
```

6.3. Pristup operacijskom sustavu i njegovim funkcionalnostima

Electron, s obzirom na to da se njime mogu razviti izvorne stolne aplikacije, razvojnom programeru pruža mnoštvo modula koji predstavljaju sučelja do različitih funkcionalnosti operacijskog sustava. Ovdje će se napraviti pregled nekoliko Electron izvornih modula koji se često koriste kod razvoja aplikacija Electron tehnologijom.

Modul *clipboard* bavi se svim operacijama vezanim uz međuspremnik računala. Opera-

cije koje podržava su kopiraj i zalijepi. Ovaj modul ima više metoda, a jedna od njih je *readText* koja kao opcionalni parametar prima znakovni niz. Ova metoda čita i kao rezultat vraća tekst koji se nalazi u međuspremniku. Metoda *writeText* kao parametar prima znakovni niz koji zatim sprema u međuspremnik. [20]

Kratki primjer korištenja *Clipboard* modula [autorski rad]:

```
const { app, clipboard } = require('electron')

app.whenReady().then(() => {
  clipboard.writeText('tekst spremljen u meduspremnik')

  const text = clipboard.readText()

  console.log(text)
})
```

Modul *globalShortcut* može registrirati globalni prečac operacijskog sustava. Prečac se može prilagoditi prema korisnikovim preferencijama i može se koristiti kada tipkovnica nema trenutni fokus na aplikaciju. Najbitnija metoda ovog modula je *register*. Metoda kao prvi parametar prima znakovni niz koji označava određenu kombinaciju tipki koju je potrebno pritisnuti kako bi se prečac okinuo. Drugi parametar *register* metode je funkcija koja će se izvršiti prilikom uspješnog pritiska kombinacije tipki definirane u prvom parametru. Metoda modula koja se veže na *register* metodu je *unregister*. Iz samog naziva metode vidi se da *unregister* metoda zapravo uklanja prije definirani prečac. [20]

Na sljedećem isječku može se vidjeti primjer korištenja *globalShortcut* Electron modula [autorski rad]:

```
const { app, globalShortcut, clipboard } = require('electron')

app.whenReady().then(() => {

  const ret = globalShortcut.register('Control+Shift+C', () => {
    console.log('pritisnut je precac')

    clipboard.writeHtml('<h1>Globalni precac</h1>')
  })

  app.on('will-quit', () => {
    globalShortcut.unregister('Control+Shift+C')
  })

})
```

Modul *safeStorage* pruža sučelje za sigurnosno spremanje osjetljivih podataka unutar pohrane računala. Modul pruža samo nekoliko metoda, od kojih su dvije najbitnije *encryptString* i *decryptString*. Obje metode primaju samo jedan parametar, tip kojeg se razlikuje ovisno o metodi. Očito je da metoda *encryptString* radi šifriranje čistog znakovnog niza. Druga metoda, *decryptString*, radi dešifriranje već prije šifriranog znakovnog niza. Kombinacijom ove dvije metode s lakoćom se sigurno mogu spremiti osjetljivi podaci, poput lozinke korisničkog računa. [20]

Primjer korištenja *safeStorage* modula [autorski rad]:

```
const { app, safeStorage, globalShortcut clipboard } = require('electron')

app.whenReady().then(() => {
  let storage = ''
```

```
    globalShortcut.register('Control+Shift+C', () => {
      storage = safeStorage.encryptString(clipboard.readText())
    })

    globalShortcut.register('Control+Shift+V', () => {
      clipboard.writeText(safeStorage.decryptString(storage))
    })

    app.on('will-quit', () => {
      globalShortcut.unregister('Control+Shift+C')
      globalShortcut.unregister('Control+Shift+V')
    })

  })
```

U primjeru se definiraju dva globalna prečaca. Pritiskom prvog prečaca uzima se tekst koji se nalazi u međuspremniku, šifrira se i sprema u lokalnu varijablu. Pritiskom drugog prečaca čita se šifrirani tekst iz lokalne varijable. Šifrirani tekst se zatim dešifrira i ponovo sprema u međuspremnik.

Electron pruža poseban modul koji se isključivo bavi slanjem HTTP zahtjeva na udaljene poslužitelje. Naziv modula je *net*. Modul pruža samo dvije metode: *request* i *isOnline*. Metoda *request* temeljna je metoda čitavog modula, a služi za kreiranje i slanje HTTP zahtjeva na udaljeni poslužitelj. Kao parametar metoda prima skup vrijednosti koje opisuju HTTP zahtjev za slanje. Navedena metoda kao rezultat vraća objekt tipa *ClientRequest*. Taj objekt se dalje može koristiti kako bi se, između ostalog, obradio događaj primitka odgovora od udaljenog poslužitelja. Druga metoda, *isOnline*, vraća istinu, odnosno laž, ovisno o tome ima li računalo vezu na internet. Bitno je napomenuti kako se ovaj modul može učitati tek kad je aplikacija spremna za korištenje. [20]

Primjer korištenja *net* modula [autorski rad]:

```
  const { app } = require('electron')

  app.whenReady().then(() => {
    const { net } = require('electron')

    const request = net.request({
      method: 'GET',
      protocol: 'https:',
      hostname: 'jsonplaceholder.typicode.com',
      path: '/posts/1'
    })

    let jsonResponse = {}

    request.on('response', response => {
      response.on('data', data => {
        jsonResponse = data
      })
    })

    request.end()
  })
```

7. Razlike između izvornih i Web aplikacija

Razvoj aplikacija u pravilu je veliki i dugotrajni proces. Naravno tu uvijek postoje iznimke, no jako teško da će se neka ozbiljnija aplikacija razviti u kratkom vremenu. Kao što je već bilo spomenuto, odabir tehnologija kojima će se aplikacija razviti jedan je od najbitnijih koraka kod razvoja aplikacije. Ovaj odabir direktno i značajno utječe na ostatak razvoja aplikacije i, između ostalog, određuje hoće li aplikacija biti izvorna ili Web aplikacija. U ovom će se poglavlju napraviti pregled najvećih prednosti i nedostataka kod razvoja izvornih, odnosno Web aplikacija.

7.1. Izvorna aplikacija

Neke od glavnih prednosti kod razvoja izvornih aplikacija su:

- najbolje moguće performanse,
- najbolja podrška za funkcionalnosti operacijskog sustava,
- visoka kvaliteta i razina detalja dokumentacije,
- veliki izbor tehnologija za razvoj,
- olakšana distribucija aplikacije

Potreba za visokom razinom performansi aplikacije se često pojavljuje kod razvoja bilo kakve aplikacije. Izvorne aplikacije su očito najbolji izbor za takve aplikacije. Izvorne aplikacije su odlične što se tiče performansi iz par razloga. Jedan od njih je to što se izvorne aplikacije u pravilu razvijaju primjenom programskih jezika koji se fokusiraju na visoke performanse. Neki od takvih jezika su C/C++, C#, Objective-C i Java. Ova četiri jezika mogu se koristiti za razvoj izvornih aplikacija za bilo koji moderni operacijski sustav, bio on za stolna računala ili mobilne telefone. Drugi razlog visokih performansi izvornih aplikacija je odlična podrška od strane razvojnih programera samih operacijskih sustava. To znači da će, primjerice, Microsoft uložiti puno truda u to da će Windows izvorne aplikacije imati najbolje performanse ukoliko su razvijene s izvornim Windows tehnologijama. Ista stvar vrijedi i za bilo koji drugi operacijski sustav.

Druga točka direktno je povezana s prošlom točkom. Točnije, izvorne aplikacije imaju najbolju moguću podršku za sve funkcionalnosti operacijskog sustava za kojeg su razvijene. Razlog tomu je sličan onom u prijašnjoj točci. Razvojnim programerima koji razvijaju i održavaju operacijski sustav je u interesu da aplikacije razvijene za taj operacijski sustav mogu pristupiti svim aspektima i funkcionalnostima operacijskog sustava. Zato i postoje razne izvorne tehnologija poput Microsoft-ove UWP tehnologije za Windows operacijski sustav ili Google-ove Jetpack tehnologije za Android operacijski sustav.

Treća točka nema najveći utjecaj kod razvoja izvornih aplikacija, no definitivno utječe na razvoj svake izvorne aplikacije. Čitljiva, jasna, ažurna i detaljna dokumentacija za izvorne

tehnologije korisna je svim razvojnim programerima, bili oni apsolutni početnici ili ekspertni programeri s puno godina iskustva. Pisanje kvalitetne dokumentacije za neki softver je dugotrajni i neprekidni proces koji zahtijeva ulaganje velike količine resursa, pogotovo kada su u pitanju kompleksni operacijski sustavi i tehnologije. Iz tog razloga najkvalitetnije dokumentacije obično pišu najveća poduzeća poput Microsoft-a, Apple-a i Google-a.

Operacijski sustavi u pravilu podržavaju veliku količinu različitih tehnologija za razvoj izvornih aplikacija. Ovo se u pravilu manifestira u obliku podrške starijih tehnologija. Kad se razvije nova tehnologija za razvoj izvornih aplikacija za neki operacijski sustav, obično se starije tehnologija i dalje ažuriraju i održavaju. Posljedica toga je da nije potrebno učiti najnovije tehnologije samo zato da bi se mogla razviti izvorna aplikacija, a već postojeće aplikacija razvijene starijim tehnologijama su i dalje podržane i održive. Primjer starije tehnologije koju je i dalje moguće koristiti je Windows Forms za Windows operacijski sustav. Ova tehnologija je nešto starija, no Microsoft razvojnim programerima i dalje pruža podršku za razvoj izvornih aplikacija njome.

Distribucija izvornih aplikacija uglavnom je relativno jednostavan proces. Svaki operacijski sustav pruža jednostavnu metodu za pakiranje i distribuciju. Ovo u pravilu uključuje posebnu trgovinu koja pruža hosting usluge i usluge instalacije same aplikacije. Primjeri takvih trgovina su Microsoft Store i App Store. Naravno, uvijek postoje alternativne metode distribucije aplikacija, no one su obično kompliciranije.

Neki od glavnih nedostataka kod razvoja izvornih aplikacija su:

- višestruki razvoj kod razvoja aplikacije za više platformi,
- Jednostavnost korištenja tehnologija
- Vrijeme razvoja aplikacije

Kad je potrebno razviti aplikaciju za više platformi, izvorne aplikacije nisu uvijek idealne. Osim u specifičnim slučajevima, razvoj aplikacije za više platformi svodi se na razvoj više aplikacija, svaka od kojih je razvijena za specifični operacijski sustav. S obzirom na to da je svaki operacijski sustav specifičan i drugačiji od ostalih, potrebno je puno više pažnje uložiti u razvoj. Ovo je potpuna suprotnost naspram Web aplikacija koje mogu funkcionirati na bilo kojoj platformi.

Prije navedena točka naravno uvelike produljuje vrijeme razvoja i resursni trošak poduzeća ili pojedinca. Nikome nije u interesu utrošiti više vremena i novaca nego što je to potrebno. Izvorne aplikacije, kad se razvijaju za više platformi, zahtijevaju veliki vremenski ulog jer se za svaki operacijski sustav, odnosno platformu, mora razviti posebna aplikacija. S druge strane, svaki operacijski sustav posjeduje posebna svojstva i funkcionalnosti što znači da je potrebno prilagoditi aplikaciju tim svojstvima, što ponovo povećava potreban vremenski ulog.

Osim toga, kod razvoja izvornih aplikacija za više platformi postoji mnoštvo tehnologija koje je potrebno poznavati. Iako su neke od tih tehnologija relativno slične, puno njih funkcionira po potpuno drugačijim principima i tehnološkim temeljima. Povećan vremenski i financijski ulog

ponovo uskače u igru ovdje jer je učenje potpuno nove tehnologije dugi proces bez kojeg je nemoguće krenuti u razvoj za specifičnu platformu.

7.2. Web Aplikacija

Neki od glavnih prednosti kod razvoja Web aplikacija su:

- jednom napisana aplikacija radi na više platformi,
- jednostavnost korištenja tehnologija,
- kraće trajanje razvoja,
- lakoća distribucije aplikacije

Glavna i najveća prednost kod razvoja Web aplikacija je to što se jednom napisana Web aplikacija može izvršavati na svim modernim platformama. Svaki moderni Web preglednik u pravilu podržava sve moderne operacijske sustave. S obzirom na to da su Web standardi poput HTML-a osnova samog Weba, svaki Web preglednik mora podržavati te Web standarde kako bi se mogao nazvati Web preglednikom. To osigurava da će bilo koja Web aplikacija u pravilu dobro funkcionirati na bilo kojem Web pregledniku. Ovo očito uzrokuje mnogo pozitivnih posljedica i prednosti. Web aplikacija s lakoćom može raditi na apsolutno svim modernim operacijskim sustavima, neovisno o tome koje izvorne tehnologije taj operacijski sustav podržava.

Prijašnja točka služi kao potpora za sve ostale prednosti razvoja Web aplikacija. Kako je Web temeljen na nekoliko standardnih i temeljnih tehnologija, te tehnologije su u pravilu dobro dizajnirane i jednostavne za korištenje. Usprkos svojoj jednostavnosti, temeljne Web tehnologije lako su proširive, a sam Web je pun raznih kvalitetnih tehnologija koje proširuju temeljne Web tehnologije.

Kad je potrebno razviti aplikaciju koja se mora moći izvršavati na više ciljanih platformi, očiti izbor za razvoj je Web aplikacija. To uvelike skraćuje vrijeme razvoja same aplikacije, što dopušta razvojnim programerima da se više fokusiraju na kvalitetu aplikacije i otklanjanje grešaka u programskom kodu. Osim toga, omogućava da se jedan dizajn aplikacije može na jednak način prikazati na svim modernim operacijskim sustavima, što dalje skraćuje vrijeme pripreme i razvoja aplikacije. Dodatna prednost ovog je smanjen financijski i resursni trošak poduzeća ili pojedinca koji razvija aplikaciju. Smanjen trošak omogućuje manjim poduzećima, pa čak i pojedincima, da u kraćem vremenu svoje ideje pretvore u stvarne aplikacije.

Web aplikacija je definitivno jedna od prilagodljivih oblika aplikacije. U praksi to znači da se Web aplikacija može razviti i distribuirati u raznim različitim okruženjima. Kod izgradnje i distribucije aplikacije razvojni programer nije ograničen na neku Web trgovinu, već s lakoćom može na svojem, ili unajmljenom, Web poslužitelju posluživati svoju aplikaciju svima koji ju žele koristiti.

Neki od glavnih nedostataka kod razvoja izvornih aplikacija su:

- ovisnost o implementacijama različitih Web preglednika,
- slabije performanse kod zahtjevnijih aplikacija,
- potreba za internetskom vezom

Iako se bilo koja Web aplikacija može izvršavati na gotovo bilo kojem Web pregledniku, opet postoje neke prepreke. Glavna prepreka kod toga je to što svaki Web preglednik ima vlastiti implementaciju svih Web standarda. To znači da je, primjerice, neko CSS svojstvo drugačije implementirano u dva različita Web preglednika. Te razlike su uglavnom minorne, no u specifičnim slučajevima može doći do problema kod razvoja Web aplikacije, što znači da može doći do dužeg trajanja razvoja Web aplikacija ili neželjenih promjena u implementaciji aplikacije.

Kao što je već bilo spomenuto, visoka razina performansi je često jedna od željenih svojstava bilo koje aplikacije. Web aplikacije su nažalost u ovom području nešto slabije. Ovo u pravilu nije problem jer je velika većina aplikacija koje se razviju orijentirano ka poslovnom svijetu. U slučaju kada to je problem, kao kod video igara ili softvera za obrađivanje video sadržaja. U tim slučajevima bolja varijanta je razvoj izvorne aplikacije.

Jedan od utjecajnijih nedostataka kod Web aplikacija je taj da je za korištenje razvijene Web aplikacije potrebno imati vezu na Internet. Razvijena Web aplikacija može biti najbolja aplikacija na svijetu, no to ništa ne znači ako korisnik ne može uopće pristupiti aplikaciji. Uz to, slaba i nestabilna internetska veza, iako tehnički omogućuje pristup Web aplikaciji, može uvelike negativno utjecati na općeniti doživljaj korištenja Web aplikacije.

8. Praktični dio

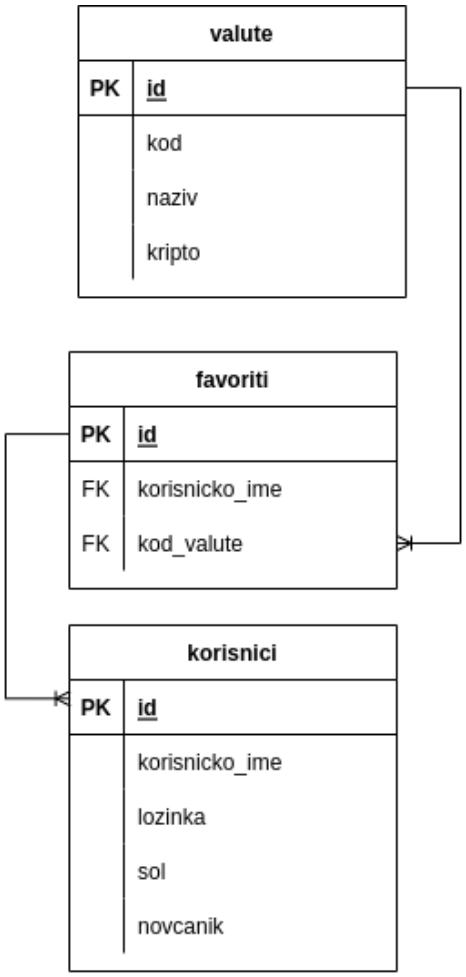
Sva prijašnja poglavlja bavila su se pregledom nekih od najpopularnijih tehnologija za razvoj aplikacija za više platformi primjenom modernih Web tehnologija. U ovom poglavlju slijedi primjena tih tehnologija u svrhe razvoja praktičnog dijela ovog rada.

Dakle, za praktični dio ovog diplomskog rada razvijene su tri različite aplikacije primjenom različitih tehnologija:

- aplikacija za stolna i prijenosna računala,
- aplikacija za pametne mobilne telefone,
- REST servis kojeg koriste ostale aplikacije.

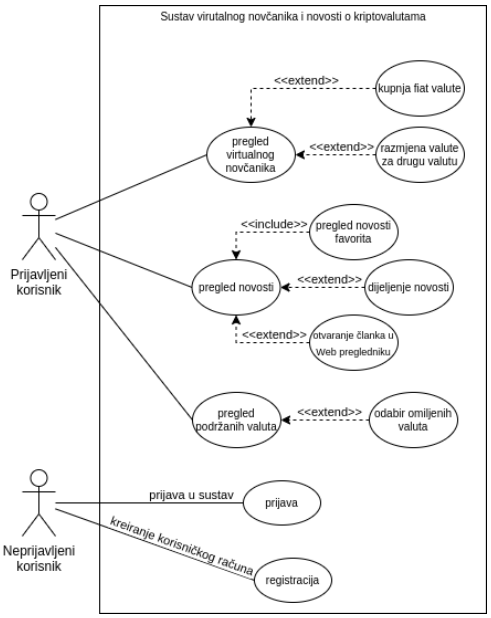
Sve tri aplikacije dio su jednog sustava i pokrivaju identične funkcionalnosti za različite platforme. Sustav koje čine ove tri aplikacije je sustav za vođenje virtualnog novčanika i praćenje novosti vezanih uz kriptovalute. Vođenje virtualnog novčanika uključuje nekoliko funkcionalnosti. Omogućuje kupnju podržanih fiat valuta (na primjer HRK, EUR, USD) i razmjenu fiat valuta za druge podržane fiat valute ili podržane kriptovalute. Osim toga, razmijenjene kriptovalute mogu se razmijeniti u suprotnom smjeru, za fiat valute. Naravno, virtualni novčanik pruža i kohezivni prikaz svih povezanih podataka. Osim virtualnog novčanika sustav pruža funkcionalnost pregleda najnovijih novosti vezanih uz kriptovalute. Novosti su u sustavu prikazane kao lista najnovijih članaka iz uglednih izvora novosti. Novosti se mogu otvoriti u Web pregledniku za detaljnije čitanje, a mogu se i dijeliti s drugim osobama. Sustav novosti upotpunjen je određenom razinom personalizacije. Naime, svaki korisnik sustava može odabrati kriptovalute koje su njegovi "favoriti", odnosno one kriptovalute za koje želi čitati više novosti. Novosti za takve kriptovalute se učestalije prikazuju u novostima. Nije moguće pratiti novosti vezane uz standardne, fiat valute. Posljednja funkcionalnost koju pruža sustav je funkcionalnost prijave u, odnosno registracije korisničkog računa. Bilo tko može izraditi račun, a svim ostalim funkcionalnostima može se pristupiti isključivo u slučaju uspješne prijave u sustav.

REST servis kojeg koriste ostale aplikacije razvijen je u JavaScript programskom jeziku, primjenom popularnog *Express* razvojnog okvira. Servis se sastoji od više krajnjih točaka do kojih bilo koji klijent može poslati standardni HTTP zahtjev. Također, unutar Express razvojnog okvira postoji modul koji se bavi spremanjem i općenitim upravljanjem korisničkim sjednicama, kako bi aplikacija mogla pratiti koji korisnici su trenutno prijavljeni u sustav. Osim *Express* razvojnog okvira koristili su se i drugi JavaScript moduli. Najznačajniji korišteni JavaScript moduli su: *redis*, *sqlite* i *sequelize*. REST servis se direktno spaja na bazu podataka, a model baze podataka nalazi se u slijedećoj slici. Model je prilično jednostavan, a sastoji se od tri relacije: valute, favoriti i korisnici. Relacija valute sadrži zapise o svim podržanim valutama sustava, dok relacija korisnici sadrži zapise o svim korisnicima sustava. Relacija favoriti objedinjuje ostale relacije na način da sadrži zapise o tome koje valute su favoriti kojeg korisnika.



Slika 9: ERA model baze podatka sustava [autorski rad]

Dijagram slučajeva korištenja u sažetom obliku prikazuje sve funkcionalnosti sustava, od prijave i registracije, do kupnje i razmjene raznih valuta. Sustav prepoznaje dvije vrste korisnika: prijavljenog i neprijavljenog. Neprijavljeni korisnik može napraviti, odnosno registrirati, novi korisnički račun ili iskoristiti postojeći korisnički račun za prijavu u sustav. Tek nakon što se korisnik prijavi u sustav dobije pristup svim ostalim funkcionalnostima sustava.



Slika 10: UML dijagram slučajeva korištenja sustava [autorski rad]

Svi navedeni moduli bave se spremanjem podataka u neku bazu podataka. Konkretno, *redis* se koristi za spremanje privremenih podataka koji se često mijenjaju poput podataka o trenutnim tečajevima svih valuta. Baza podataka podržava postavljanje vremena isteka valjanosti podataka. Iz tog razloga praćenje najnovijih tečajeva postaje trivijalno jer se stari tečajevi automatski obrišu iz baze podataka, pa je potrebno dohvatiti ažurne podatke. Primjer privremenog spremanja podataka o tečajevima neke valute [autorski rad]:

```
const cacheRates = async (base) => {
  const twoHours = 7200
  const response = await requests.getCurrencyRates(base, supportedCurrencies.map(c
      => c.code))
  const data = response.data.rates
  await redisClient.set(base, JSON.stringify(data), {
    EX: twoHours
  })

  return data
}

app.get('/currency/rates/:base', authMiddleware, async (req, res, next) => {
  const { base } = req.params
  if (!supportedCurrencies.some((el) => el.code === base)) {
    res.reason = 'Invalid currency'
    return next()
  }
  const redisData = await redisClient.get(base)
  let data = {}
  if (!redisData || JSON.parse(redisData).length === 0) {
    data = await cacheRates(base)
  }
  else {
    data = JSON.parse(redisData)
  }

  res.json({rates: data})
})
```

Krajnja točka */currency/rates/:base* koristi *redis* klijent kako bi provjerila postoje li ažurni podaci o tečajevima za neku valutu. Ukoliko ti podaci nisu lokalno spremljeni, poziva funkciju *cacheRates* koja dohvaća najnovije podatke o tečajevima i sprema ih u lokalnu *redis* bazu podataka. Vrijeme valjanosti dohvaćenih podataka je dva sata, nakon čega se podaci brišu i

potrebno je dohvatiti nove podatke za istu valutu.

Modul *sqlite* koristi se za instanciranje lokalne *SQLite* baze podataka, a tu bazu podataka koristi *Express* razvojni okvir kako bi spremao podatke o postojećim korisničkim sjednicama unutar sustava. Sve što je potrebno napraviti kako bi *Express* razvojni okvir mogao koristiti *SQLite* bazu podataka je ubaciti *session* funkciju u *Express* tok. Primjer upravljanja korisničkim sjednicama [autorski rad]:

```
const session = require('express-session')
const SqliteStore = require("better-sqlite3-session-store")(session)
const express = require('express')
const app = express()

app.use(session({
  genid: uuidv4,
  secret: 'mandel_diplomski',
  name: 'sessionId',
  saveUninitialized: false,
  resave: false,
  store: new SqliteStore({
    client: database,
    expired: {
      clear: true,
      intervalMs: 1000000
    }
  }),
}))

const authMiddleware = (req, res, next) => {
  if (!req.session?.login) {
    return res.status(401).send('Not authorized')
  } else {
    next()
  }
}

app.post('/auth/login', async (req, res, next) => {
  const { username, password } = req.body
  if (!username || !password) {
    res.reason = 'Invalid username or password!'
    res.status(400)
    next()
  }
  try {
    const user = await User.findOne({ where: { username: username } })
    if (user) {
      const hashed = await bcrypt.hash(password, user.salt)
      if (user.password === hashed) {
        req.session.login = true
        req.session.username = username
        ...
      }
    }
  }
})

app.post('/logout', authMiddleware, async (req, res, next) => {
  req.session.destroy(() => {
    res.sendStatus(200)
  })
})
```

U primjeru su prikazane sve funkcije i krajnje točke koje na neki način upravljaju korisničkom sjednicom. U početku se u *Express* aplikaciju uključi *session* funkcija koja generira objekt sjednice i postavlja ga u objekt zahtjeva (*req.session*). U konfiguracijski parametar potrebno je postaviti nekoliko vrijednosti, no najbitnija je vrijednost *store* u koju je potrebno postaviti podržanu način i lokaciju spremanja podataka o sjednicama. U početku, podaci o sjednici nisu trajno spremljeni u memoriju aplikacije. Kako bi se podaci trajno spremili, potrebno je dodati neku vrijednost u objekt sjednice. U primjeru, ovo se radi u */auth/login* krajnjoj točci gdje se,

prilikom uspješne prijave, postavljaju vrijednosti *login* i *username*. Kako bi se neprijavljenim korisnicima mogle zabraniti određene krajnje točke, napravljena je funkcija *authMiddleware* koja provjerava postoji li *login* vrijednost u objektu sjednice. Za brisanje sjednice koristi se */logout* krajnja točka unutar koje se poziva *destroy* funkcija objekta sjednice.

8.1. Stolna aplikacija

Stolna aplikacija razvijena je za sve moderne operacijske sustave za stolna i prijenosna računala (Windows, Linux, MacOS). Osim toga, razvijena stolna aplikacija pruža sve funkcionalnosti spomenute u dijagramu slučajeva korištenja. Kako bi mogla pružiti sve funkcionalnosti koristi prije spomenuti razvijeni REST servis. Stolna aplikacija razvijena je Electron programskim okvirom, a za grafičko sučelje aplikacije korištena je *React* JavaScript biblioteka. Electron okvir, kao što je već bilo objašnjeno, koristi specifičan model procesa koji se sastoji od jednog glavnog procesa i jednog ili više procesa prikazivanja. Glavni proces bavi se svim pozadinskim poslovima, poput učitavanja aplikacije u memoriju. S obzirom na to da se ne radi o Web aplikaciji koja se odvija u Web pregledniku, potrebno je na neki način upravljati samim prozorom, odnosno prikazom, aplikacije. Upravljanje prikazom aplikacije u potpunosti se obavlja unutar Electron okvira. Dakle, Electron okvir, osim što može stvoriti, odnosno otvoriti, novi prozor aplikacije, može i smanjiti prozor ili ga u potpunosti isključiti. Nekakav prvi korak kod pokretanja aplikacije je uvijek otvaranje novog prozora kako bi se aplikacija zapravo mogla prikazati. Primjer upravljanja prozorom unutar glavnog procesa aplikacije [autorski rad]:

```
const createWindow = async () => {

  mainWindow = new BrowserWindow({
    width: 1024,
    height: 728,
    webPreferences: {
      preload: `${__dirname}/preload.ts`
    }
  })

  mainWindow.loadURL(resolveHtmlPath('index.html'))

  mainWindow.on('ready-to-show', () => {
    if (!mainWindow) {
      throw new Error('"mainWindow" is not defined')
    }
    if (process.env.START_MINIMIZED) {
      mainWindow.minimize()
    } else {
      mainWindow.show()
    }
  })

  mainWindow.on('closed', () => {
    mainWindow = null
  })

  const menuBuilder = new MenuBuilder(mainWindow)
  menuBuilder.buildMenu()

  mainWindow.webContents.setWindowOpenHandler(edata => {
    shell.openExternal(edata.url)
    return { action: 'deny' }
  })

}

app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') {
    app.quit()
```

```
  }
})

app
  .whenReady()
  .then(() => {
    createWindow()
    app.on('activate', () => {
      if (mainWindow === null) createWindow()
    })
  })
  .catch(console.log)
```

U primjeru se nalazi *createWindow* funkcija koja kreira novi prozor aplikacije i postavlja nekoliko slušača događaja nad tim prozorom. Kreiranjem novog *BrowserWindow* objekta kreira se novi prozor aplikacije, a kao argument prima veličinu novostvorenog prozora i skriptu koja će se pokrenuti prije samog otvaranja prozora. Događaji za koje se postavljaju slušači su *ready-to-show* i *closed* koji se okidaju kad je prozor spreman za prikaz, odnosno kada se prozor zatvori. Osim slušača događaja nad prozorom postavljaju se dva slušača nad čitavom aplikacijom: *window-all-closed* i *whenReady*. Slušač *whenReady* izrazito je bitan jer omogućuje otvaranje prozora i pokretanje same aplikacije točno onda kada je aplikacija spremna za to, odnosno kada su svi resursi spremni za korištenje. Korištenjem ovog slušača izbjegavaju se potencijalne greške prilikom pokretanja aplikacije poput nepotpuno učitanih resursa.

Ostatak stolne aplikacije se u potpunosti odvija unutar procesa prikazivanja, a za razvoj tog dijela aplikacije korištena je *React* JavaScript biblioteka. Biblioteka se temelji na izradi vlastitih i korištenjem ugrađenih komponenata za izgradnju korisničkog sučelja. Glavna, odnosno ulazna, komponenta aplikacije zove se *App*, a unutar nje ugniježđene su sve ostale komponente koje čine sučelje aplikacije. S obzirom da se radi o stolnoj aplikaciji, navigacija kroz aplikaciju trebala se uzeti u obzir. Nije bilo moguće koristiti standardni *<a>* HTML element, pa je za navigaciju korištena *react-router* biblioteka. Ta biblioteka pruža sve potrebne komponente za upravljanje i rukovanje svom navigacijom unutar aplikacije. Korištenje *react-router* biblioteke unutar stolne aplikacije [autorski rad]:

```
import { HashRouter } from 'react-router-dom'

render(<HashRouter><App /></HashRouter>, document.getElementById('root'))

function App() {
  const [loggedUser, setLoggedUser] = useState({} as any)

  return (
    <AuthContext.Provider value={{loggedUser, setLoggedUser}}>
      <Routes>
        <Route path='/' element={<MainView />}>
          <Route path='/login' element={<LoginForm setUser={setLoggedUser} />} />
          <Route element={<RequireAuth />}>
            <Route path='/news' element={<NewsView />} />
            <Route path='/wallet' element={<WalletView />} />
            <Route path='/wallet/buy/:currency' element={<BuyCurrencyView />} />
            <Route path='/wallet/exchange/:currency' element={<
                ExchangeCurrencyView />} />
            <Route path='/currencies' element={<CurrenciesView />} />
          </Route>
        </Route>
      </Routes>
    </AuthContext.Provider>
  )
}
```

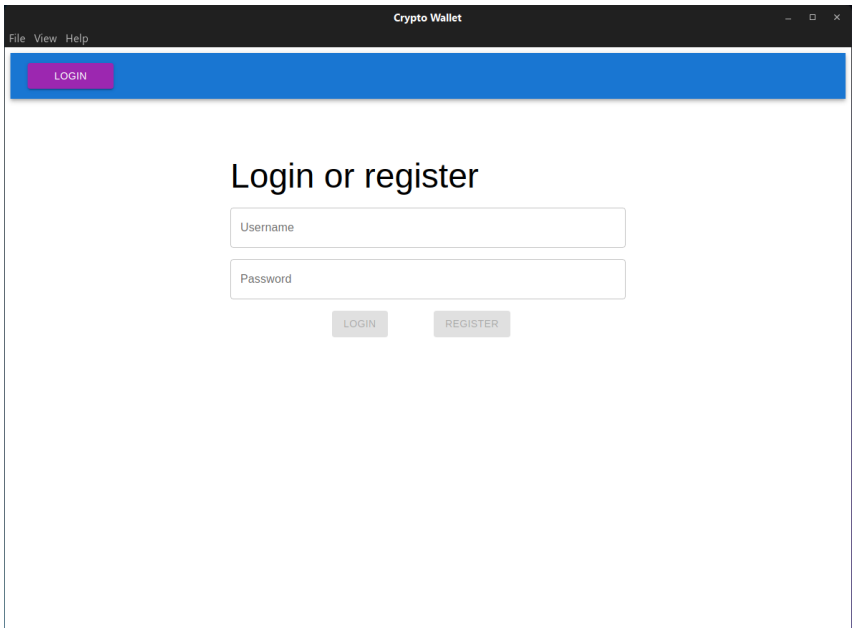
App komponenta smještena je unutar *HashRouter* komponente koja je uvezena iz *react-router* biblioteke. Na taj način omogućeno je korištenje ostalih komponenata iz *react-router* bi-

blioteke poput *Routes* i *Route*. *Route* komponenta definitivno je najbitnija što se tiče navigacije, a funkcionira tako da se u komponentu proslijedi par atributa: *path* i *element*. *Path* atribut označava rutu, a *element* atribut označava koji će se element, odnosno komponenta, prikazati kada je aktivna ruta koja se nalazi u *path* atributu. Kao što se može vidjeti, gotovo sve rute zaštićene su posebnom rutom koja nema *route* atribut, već samo *element* atribut u kojeg je smještena *RequireAuth* komponenta koja se bavi provjerom korisničke sjednice kako bi se osiguralo da samo prijavljeni korisnik može pristupiti ostatku aplikacije. Pregled *RequireAuth* komponente [autorski rad]:

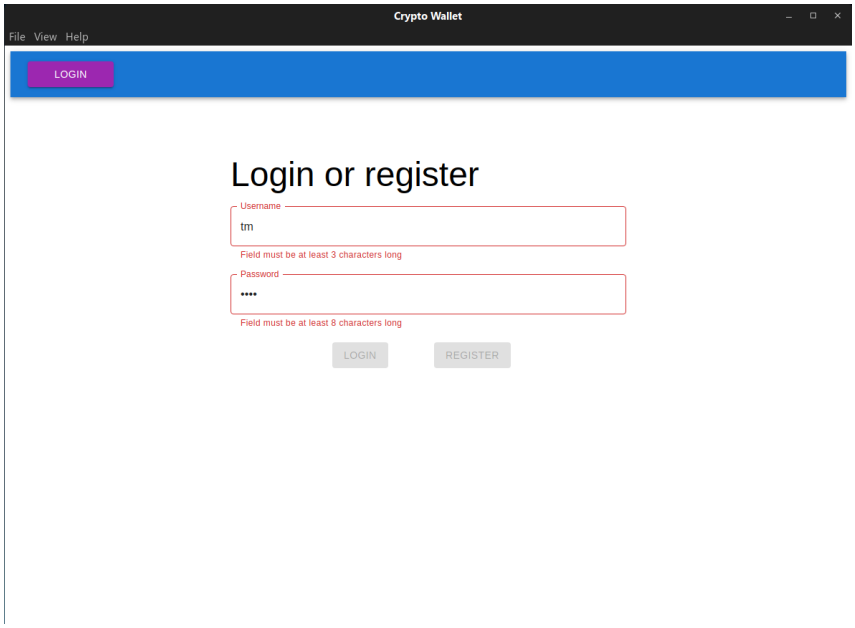
```
type Currency = {
  amount: number,
  code: string,
  crypto: boolean
}

export type Wallet = {
  [key: string]: Currency
}

export type User = {
  wallet: Wallet,
  username: string
}

const AuthContext = React.createContext({
  loggedUser: {},
  setLoggedUser: (user: User) => {}
} as { loggedUser: User, setLoggedUser: (user: User) => void })


const RequireAuth = () => {
  const location = useLocation()
  const navigate = useNavigate()
  const authContext = useContext(AuthContext)

  useEffect(() => {
    if (!authContext.loggedUser.username) {
      navigate('/login')
    }
  }, [location])

  return (
    <Outlet />
  )
}
```

RequireAuth komponenta koristi nekoliko ugrađenih *React* funkcija kako bi uspješno obavljala svoj posao. Uz ugrađene funkcije, također koristi i *useLocation* funkciju iz *react-router* biblioteke koja vraća objekt koji služi za dohvaćanje podataka o navigaciji i upravljanje navigacijom aplikacije. Funkcija *useEffect* koristi se kako bi se iz konteksta aplikacije moglo provjeriti je li korisnik prijavljen u aplikaciju. Ta se funkcija pokreće kod svake promjene objekta navigacije kako bi se osiguralo da neprijavljen korisnik ne može pristupiti aplikaciji.

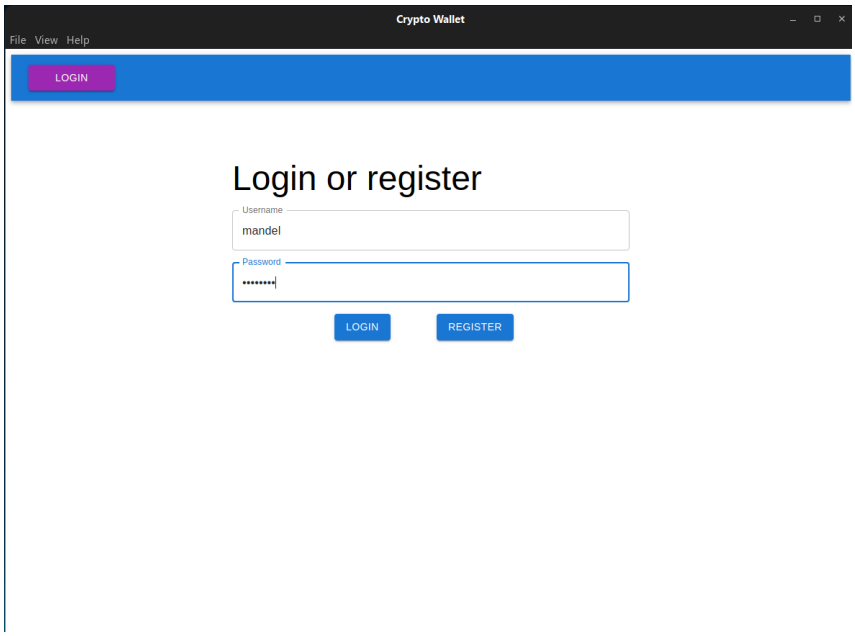
Jedina ruta, odnosno komponenta, koja je dostupna neprijavljenim korisnicima je komponenta za prijavu i registraciju. Ta komponenta naravno mora biti dostupna neprijavljenim korisnicima kako bi mogli potvrditi svoj identitet i pristupiti ostatku aplikacije. Čitava komponenta je zapravo standardni obrazac s provjerom korisničkog unosa, a podržava unos za korisničko ime i lozinku.



Slika 11: Prikaz obrasca za prijavu i registraciju [autorski rad]



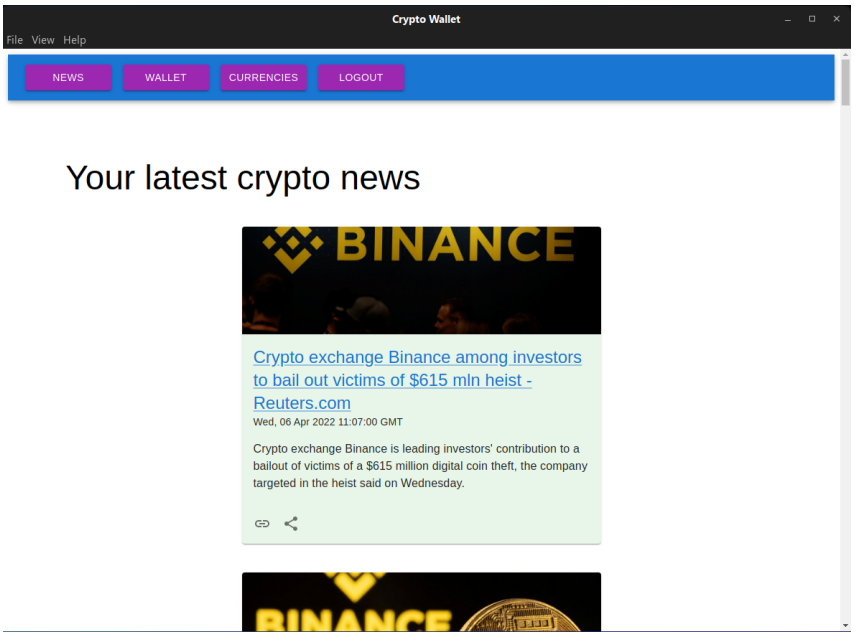
Slika 12: Provjera korisničkog unosa - neispravan unos [autorski rad]



Slika 13: Provjera korisničkog unosa - ispravan unos [autorski rad]

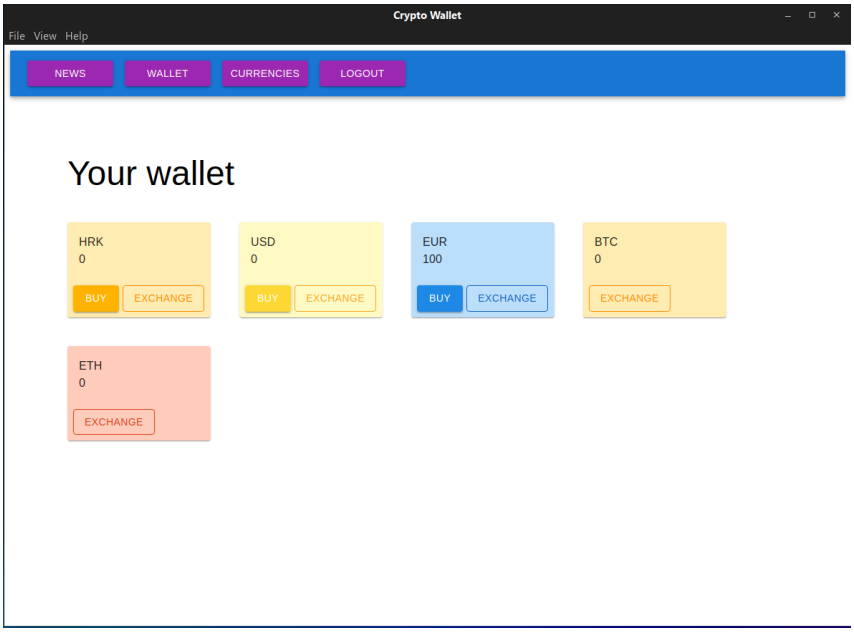
Nakon uspješne prijave u aplikaciju otvara se početni prikaz - prikaz novosti o kriptovalutama. Komponenta prikaza novosti prvotno provjerava postoje li valjani podaci o novostima u lokalnoj pohrani. Ako novosti ne postoje, komponenta radi zahtjev na REST servis kako bi dohvatila najnovije novosti i spremila ih u lokalno pohranu. Valjanost tih novosti postavlja se na jedan sat kako bi se izbjeglo pretjerano terećene poslužitelja na kojem je pokrenut REST servis. Ova logika može se vidjeti na sljedećem isječku [autorski rad]:

```
// ...
const localNews = localStorage.getItem('news')
const newsJson = localNews && JSON.parse(localNews)
if (!newsJson || new Date().getTime() > newsJson.expiration) {
  getNewsRequest().then(response => {
    const articles = response.data.articles
    setNews(articles)
    localStorage.setItem('news', JSON.stringify({
      expiration: new Date().getTime() + 3600000,
      articles
    }))
    spinner.setOpen(false)
  })
} else {
  setNews(newsJson.articles)
}
// ...
```

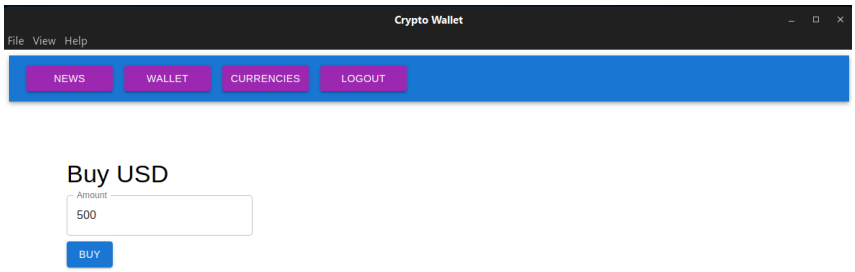


Slika 14: Prikaz najnovijih novosti o kriptovalutama [autorski rad]

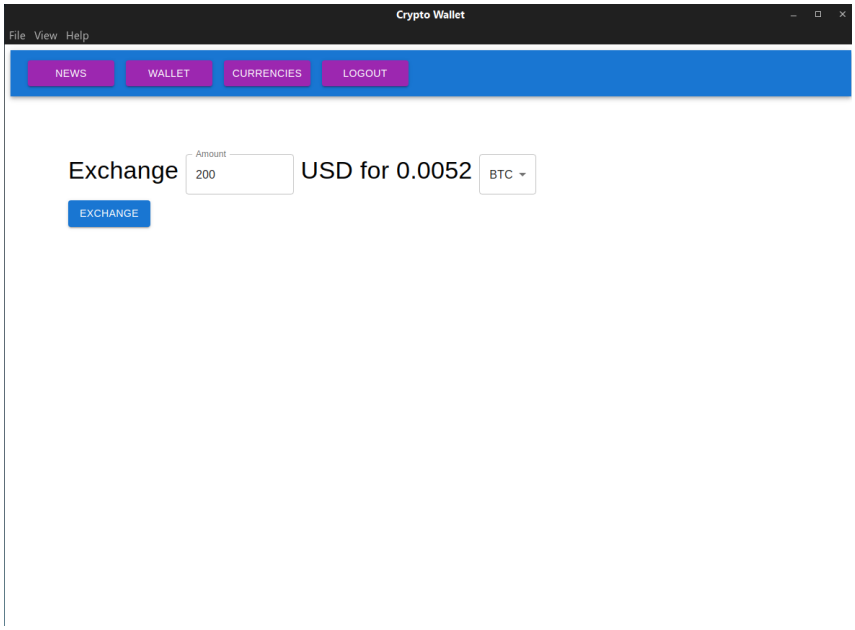
Prikaz novčanika sastoji se od prikaza kartica valuta na kojima se nalazi podatak o količini pojedine valute na računu. Svaka kartica na sebi ima i gumbe - jedan za kupnju valute ako se radi o fiat valuti i jedan za razmjenu valute za neku drugu valutu. Klikom na prvi gumb otvara se prikaz s obrascem za kupnju određene fiat valute. Klikom na drugi gumb, koji je dostupan za sve vrste valuta, otvara se obrazac za razmjenu odabrane valute za neku drugu valutu. Konverzija vrijednosti jedne valute u drugu događa se u pravom vremenu.



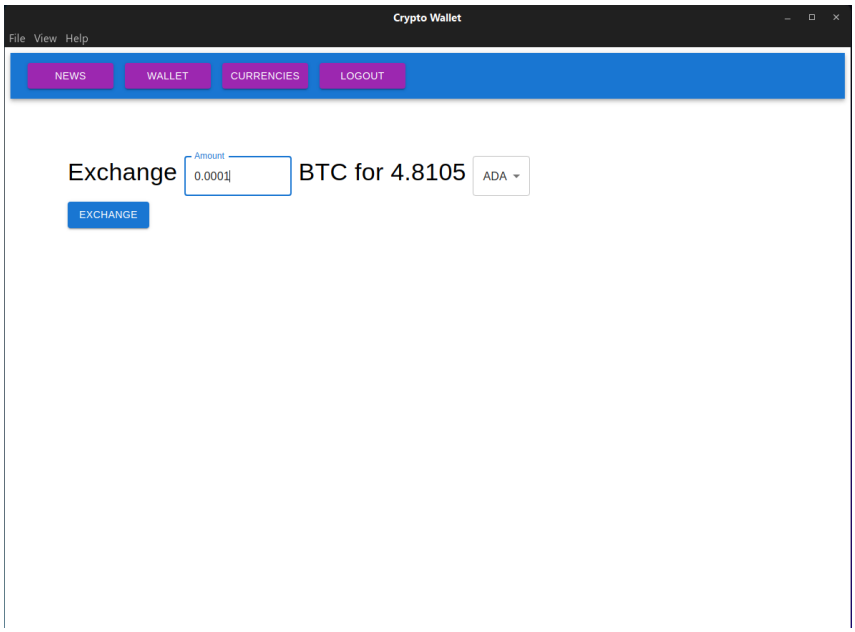
Slika 15: Prikaz virtualnog novčanika [autorski rad]



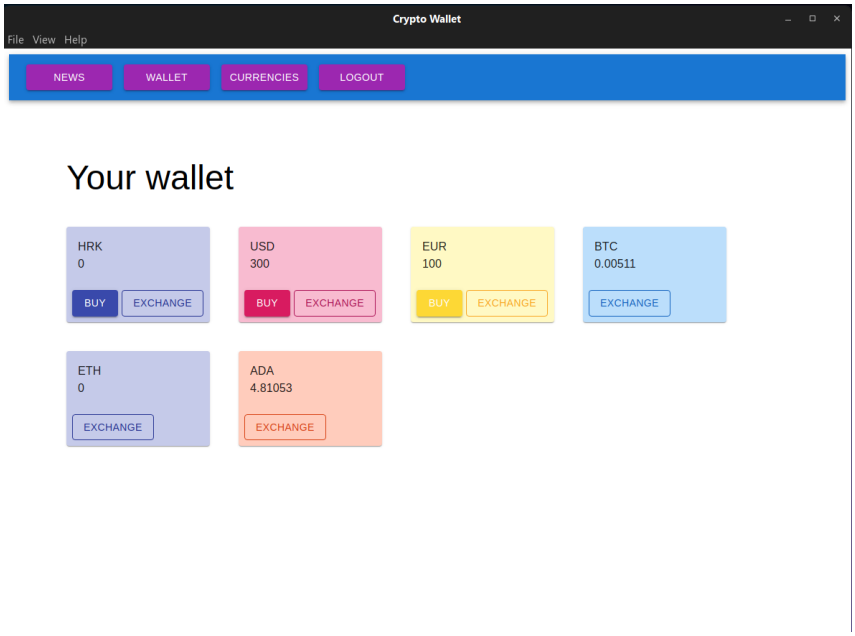
Slika 16: Prikaz kupnje fiat valute [autorski rad]



Slika 17: Prikaz razmjene fiat valute za kritpovalutu [autorski rad]

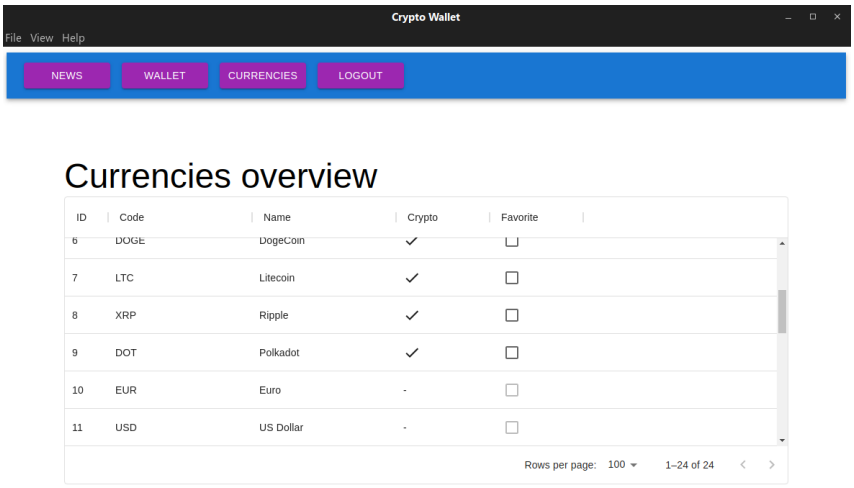


Slika 18: Prikaz razmjene kriptovaluta [autorski rad]

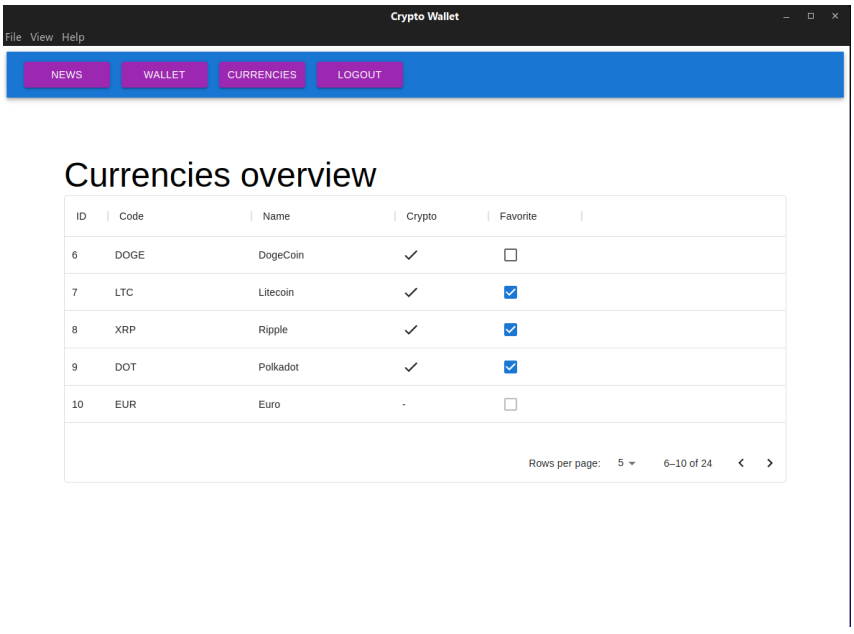


Slika 19: Prikaz virtualnog novčanika nakon svih promjena [autorski rad]

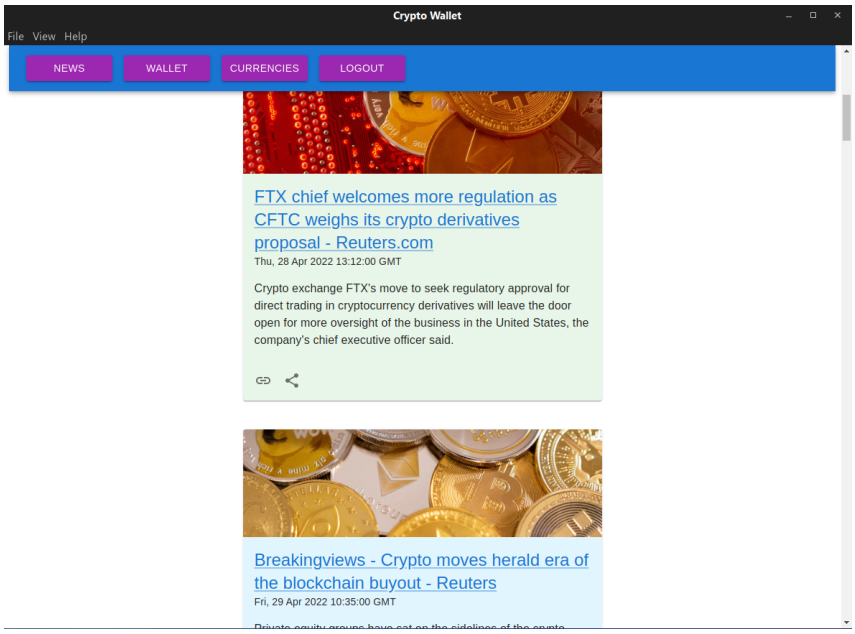
Posljednji prikaz aplikacije je prikaz svih podržanih valuta sustava. Ovaj prikaz sastoji se od tabličnog prikaza svih podržanih valuta uz mogućnost odabira bilo koje podržane kriptovalute kao favorita. Odabirom favorita šalje se zahtjev na REST servis, a postojeći podaci o novostima spremljeni u lokalnoj pohrani se brišu kako bi se, po povratku na prikaz novosti mogli dohvatiti najnoviji podaci.



Slika 20: Prikaz liste podržanih valuta [autorski rad]



Slika 21: Prikaz liste podržanih valuta - odabir favorita [autorski rad]



Slika 22: Prikaz novosti nakon odabira favorita [autorski rad]

8.2. Mobilna aplikacija

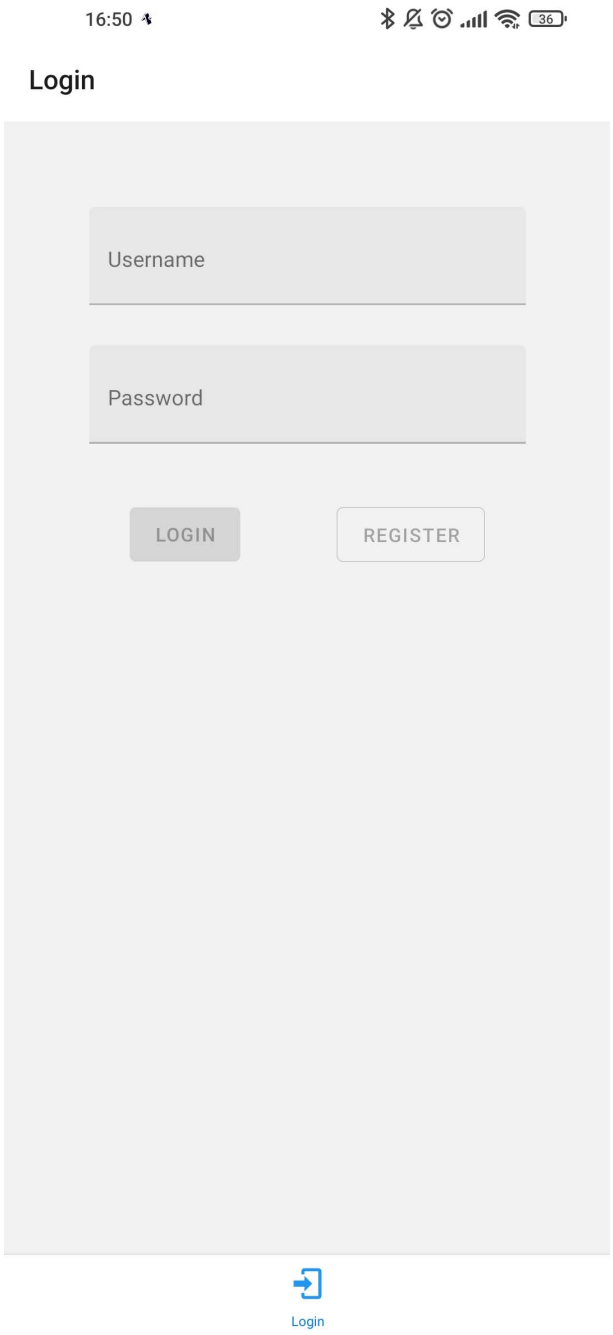
Mobilna aplikacija razvijena je za dva najpopularnija operacijska sustava za pametne mobilne uređaje - Android i iOS. Aplikacija je razvijena korištenjem *React Native* tehnologije. Tehnologija je izrazito slična *React* JavaScript biblioteci korištenoj kod izrade stolne aplikacije. Mobilna aplikacija pokriva identične slučajeve korištenja, a poslovna logika same aplikacije napravljena je da što više nalikuje poslovnoj logici stolne aplikacije. S obzirom da se ne radi o istim tehnologijama, neke biblioteke korištene kod izrade stolne aplikacije nisu mogle biti korištene kod izrade mobilne aplikacije. Najveća razlika u implementaciji aplikacija nastala kao posljedica razlika u bibliotekama je navigacija aplikacije. Naime, kod razvoja mobilne aplikacije nije bilo moguće koristiti *react-router* biblioteku jer ona ne podržava *React Native* tehnologiju. Najprigodnija alternativa za *react-router* biblioteku, ispostavilo se, je *React Navigation* biblioteka koja je namijenjena za korištenje isključivo s *React Native* tehnologijom. *React Navigation* biblioteka pruža jako slične, ako ne i identične, funkcionalnosti kao i *react-router* biblioteka, a način korištenja biblioteke je također poprilično sličan. Mobilna se aplikacija, kao i stolna, temelji na korištenju komponenata, a glavna komponenta se također naziva *App*. Implementacija navigacije u mobilnoj aplikaciji [autorski rad]:

```
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs'
// ...
const Stack = createBottomTabNavigator()

// ...
<NavigationContainer>
  <Stack.Navigator>
    {!user.username &&
      <>
        <Stack.Screen name='Login' component={LoginForm}
          options={{ tabBarIcon: ({ focused }) => {
            return <IconButton color={focused ? Colors.blue500 : Colors.grey600}
                icon='login' />
          } }}
        />
```

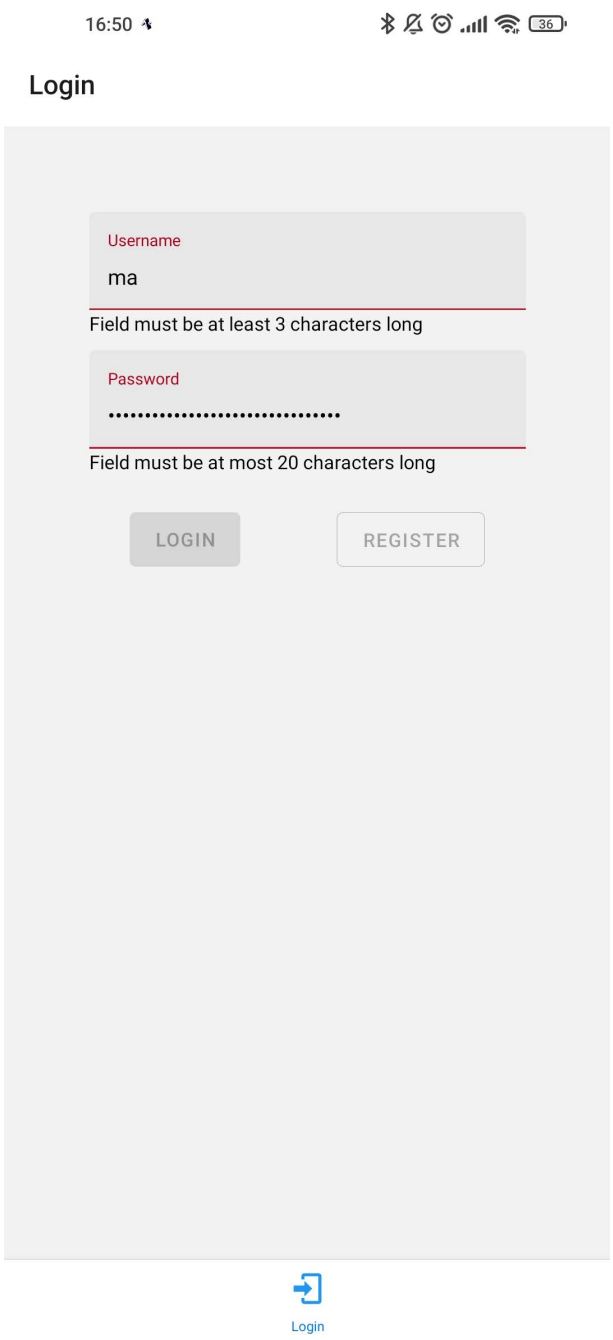
```
        </>
      }
      {user.username && <>
        <Stack.Screen name='News' component={NewsView}
          options={{ tabBarIcon: ({ focused }) => {
            return <IconButton color={focused ? Colors.blue500 : Colors.grey600}
                icon='newspaper' />
          } }}
        />
        <Stack.Screen name='Wallet' component={WalletView}
          options={{ tabBarIcon: ({ focused }) => {
            return <IconButton color={focused ? Colors.blue500 : Colors.grey600}
                icon='wallet' />
          } }}
        />
        <Stack.Screen name='Currencies' component={CurrenciesView}
          options={{ tabBarIcon: ({ focused }) => {
            return <IconButton color={focused ? Colors.blue500 : Colors.grey600}
                icon='currency-btc' />
          } }}
        />
        <Stack.Screen name='BuyCurrency' options={{ tabBarButton: () => null }}
            component={BuyCurrencyView} />
        <Stack.Screen name='ExchangeCurrency' options={{ tabBarButton: () => null }}
             component={ExchangeCurrencyView} />
    </>}
    </Stack.Navigator>
  </NavigationContainer>
  // ...
```

Dakle, za implementaciju navigacije potrebno je uvesti dvije komponente - *NavigationContainer* koja se uvozi neovisno o vrsti navigacije, i, u ovom slučaju, *Stack.Navigator* koja određuje o kakvoj će se vrsti navigacije raditi. Konkretno, u ovoj aplikaciji radi se o navigaciji koja se nalazi na dnu ekrana u obliku reda u kojem se nalaze ikone koje se ponašaju kao poveznice na različite dijelove aplikacije.

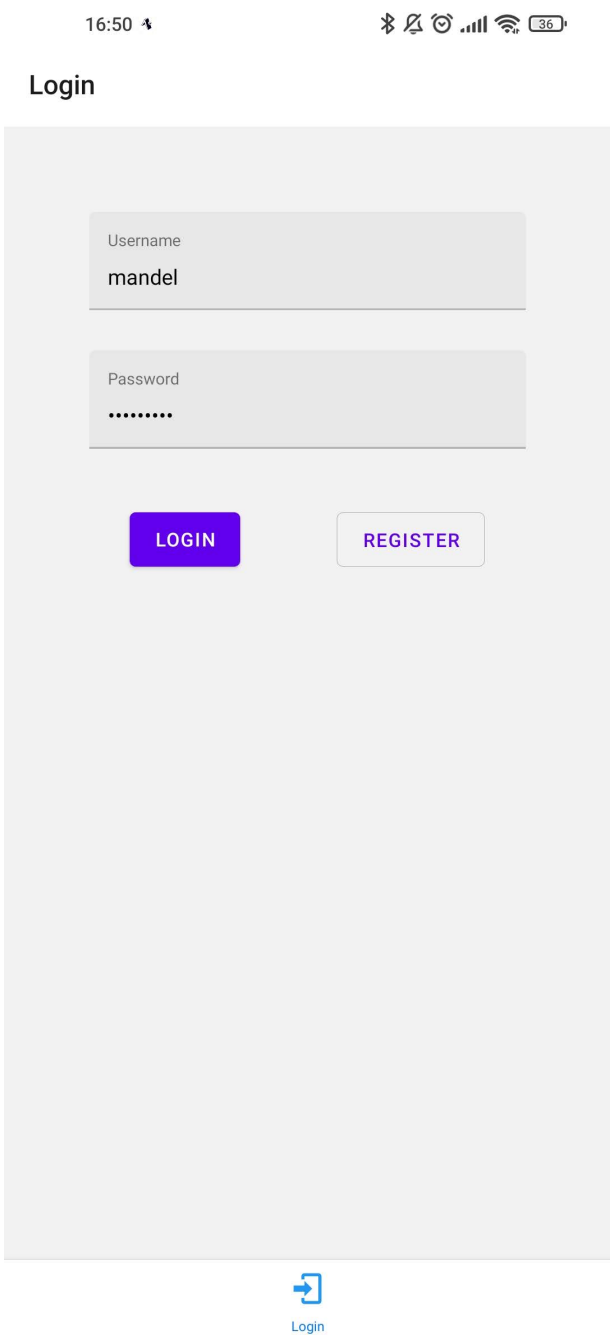
Kao i kod stolne aplikacije, jedini prikaz dostupan neprijavljenom korisniku je prikaz obrasca prijavu i registraciju. Nakon uspješne prijave, ostali prikazi postaju dostupni.



Slika 23: Mobilna aplikacija - obrazac za prijavu [autorski rad]



Slika 24: Mobilna aplikacija - obrazac za prijavu, neispravan unos [autorski rad]

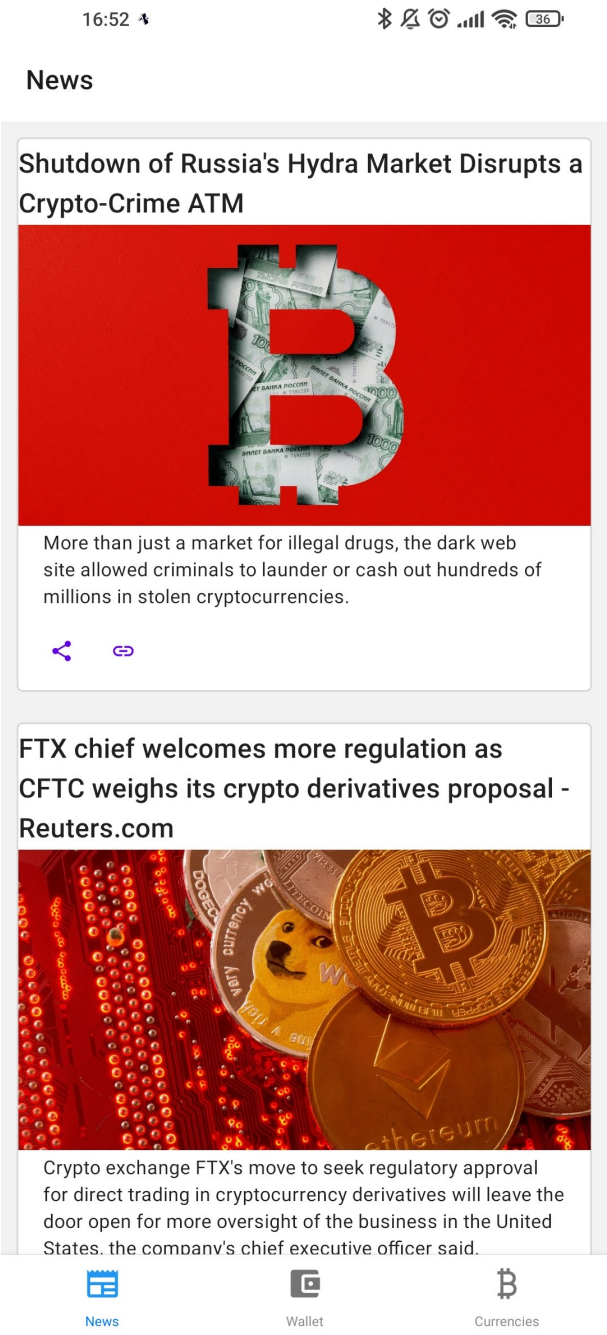


Slika 25: Mobilna aplikacija - obrazac za prijavu, ispravan unos [autorski rad]

Prikaz novosti je gotovo identičan onom u stolnoj aplikaciji. Jedina razlika je u implementaciji funkcionalnosti dijeljenja određenog članka. Naime, za razliku od stolnih operacijskih sustava, mobilni operacijski sustavi pružaju poseban prozor za dijeljenje sadržaja s drugim aplikacijama. Iz tog razloga bilo je potrebno implementirati posebnu funkciju koja otvara prozor za dijeljenje sadržaja, dok se samo dijeljenje sadržaja odvija u pozadini, ovisno o operacijskom sustavu. *React Native* tehnologija pruža posebnu funkciju koja se bavi dijeljenjem sadržaja, a potrebnu ju je samo uvesti u komponentu u kojoj je potrebno dijeliti sadržaj. Implementacija funkcionalnosti dijeljenja sadržaja [autorski rad]:

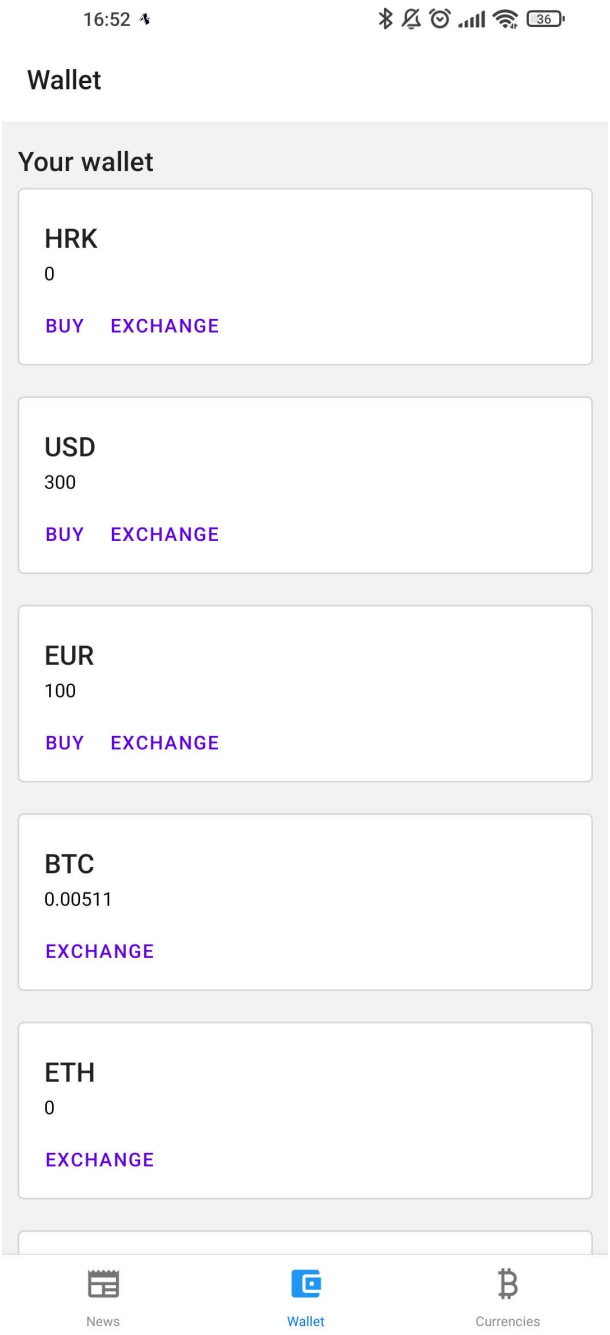
```
import { Share } from 'react-native'
```

```
// ...

const shareArticle = () => {
  Share.share({
    message: url
  })
}

// ...

<Card onPress={openInBrowser} style={style} mode='outlined'>
  <View style={style.title}>
    <Title>{title}</Title>
  </View>
  <Card.Cover source={{ uri: urlToImage }} />
  <Card.Content>
    <Paragraph>{description}</Paragraph>
  </Card.Content>
  <Card.Actions>
    <Button onPress={shareArticle} icon='share-variant'>

    </Button>
    <Button onPress={openInBrowser} icon='link'>

    </Button>
  </Card.Actions>
</Card>
```

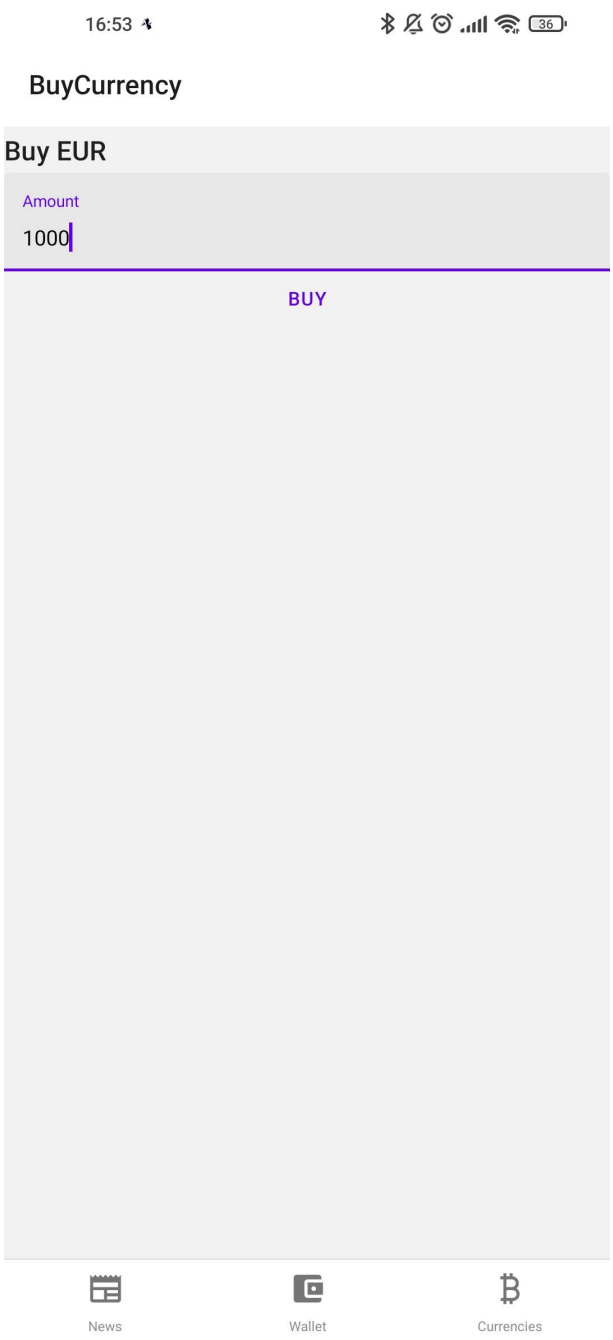


Slika 26: Mobilna aplikacija - prikaz najnovijih novosti o kriptovalutama [autorski rad]

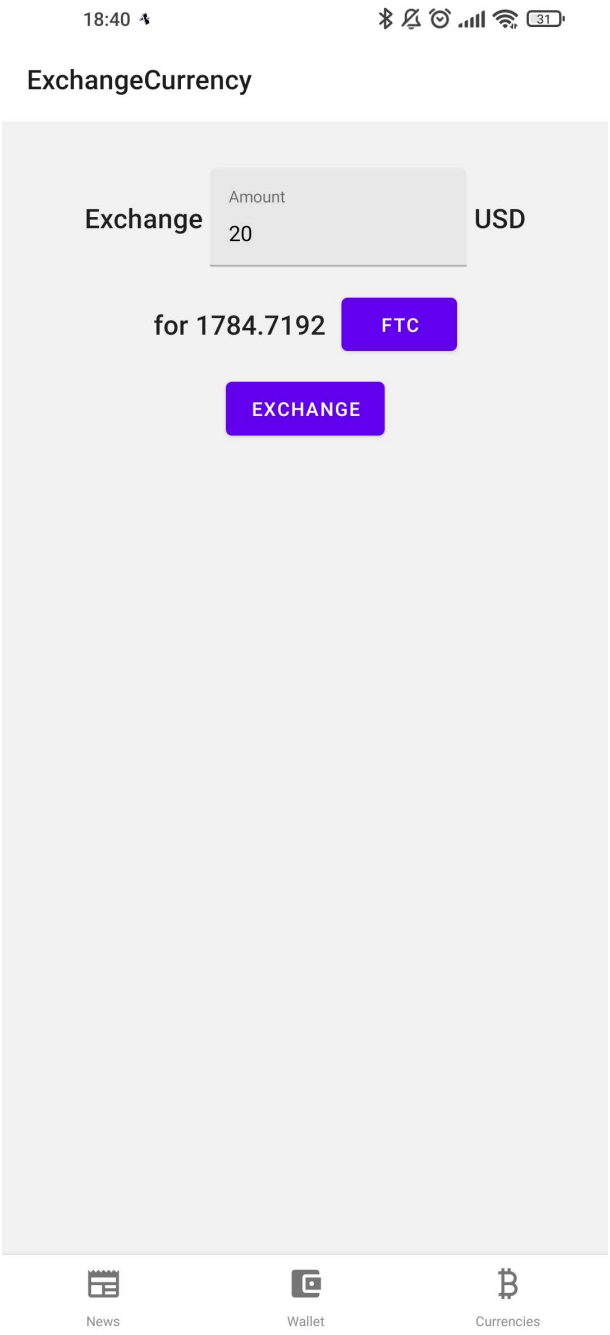
Ostatak aplikacije, ne uzimajući u obzir izgled sučelja, gotovo je identičan stolnoj aplikaciji. Mobilna aplikacija također pruža pregled novčanika koji se sastoji od kartica valuta i gdje se mogu kupiti fiat valute, odnosno razmijeniti različite valute. Osim toga, aplikacija također pruža prikaz svih podržanih valuta u obliku tablice, a u tablici se mogu označiti favoriti korisnika, na temelju kojih će se prikazivati personalizirane novosti.



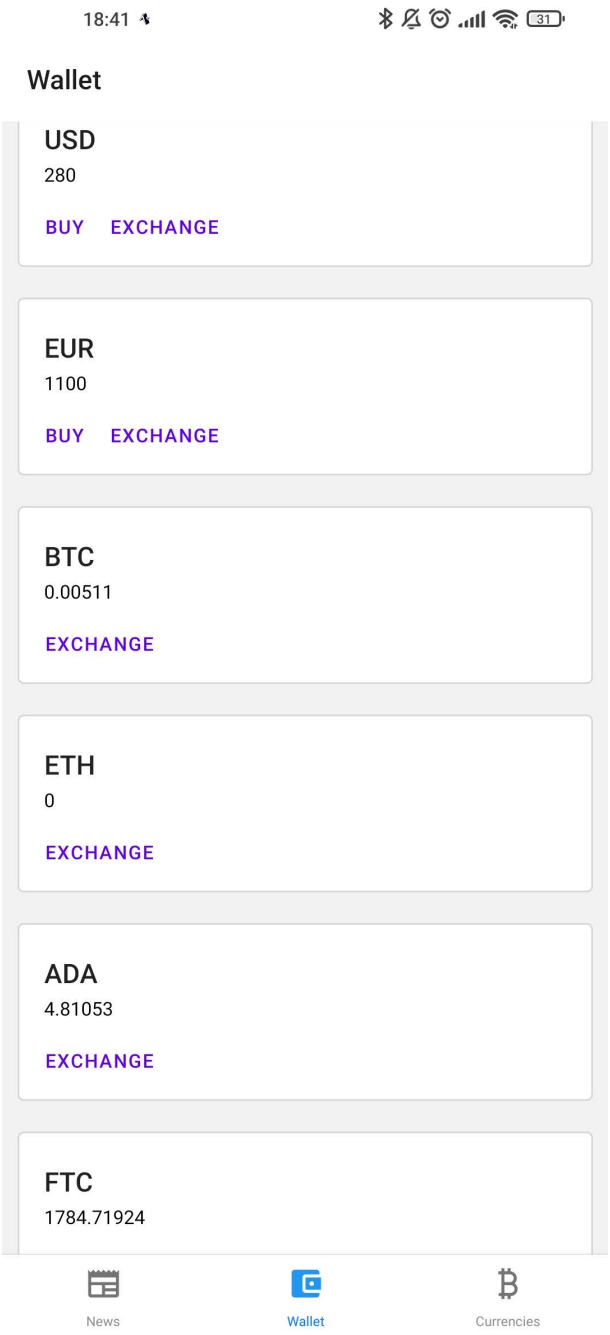
Slika 27: Mobilna aplikacija - prikaz novčanika [autorski rad]



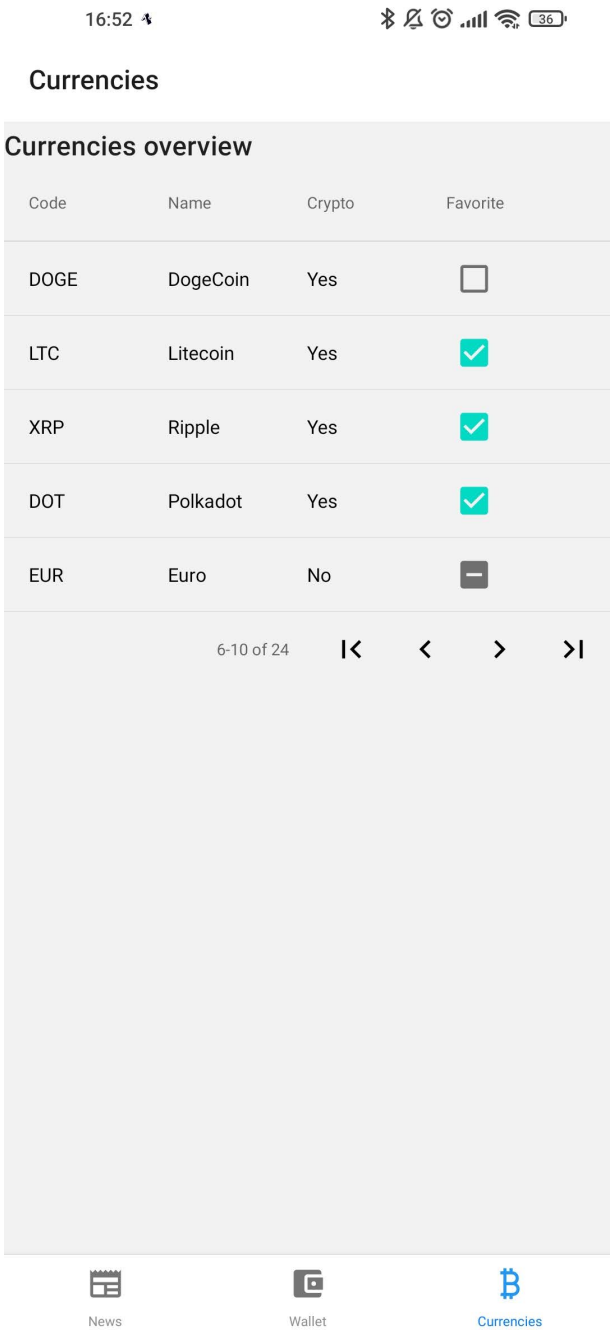
Slika 28: Mobilna aplikacija - prikaz kupnje fiat valute [autorski rad]



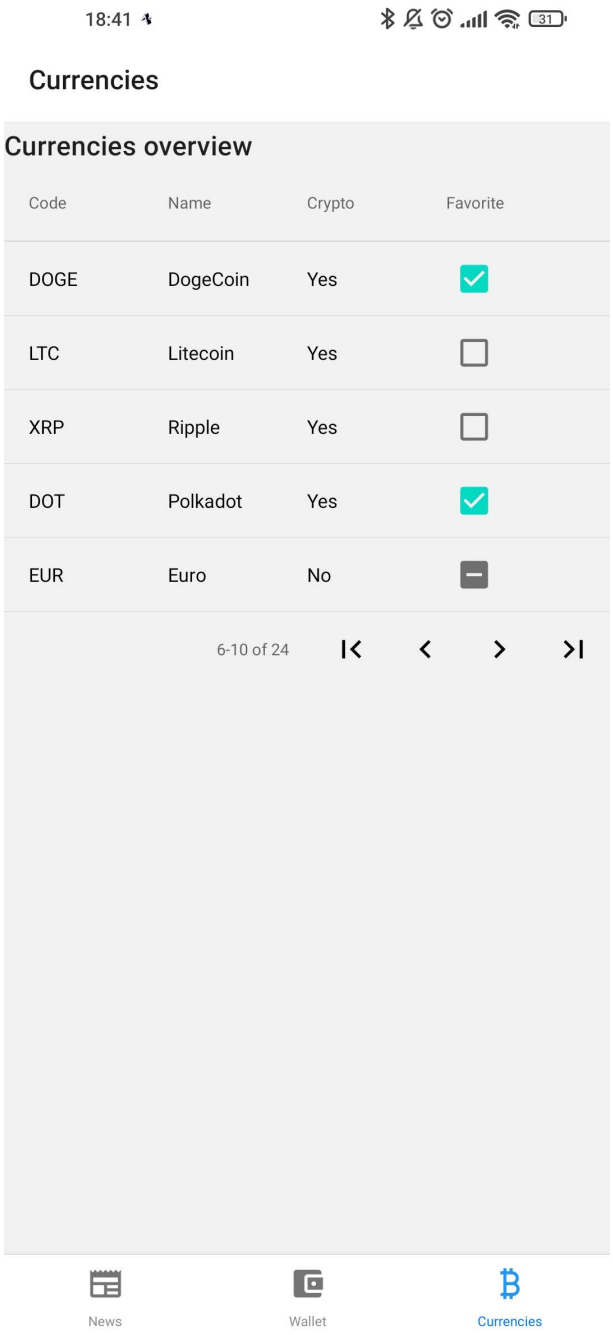
Slika 29: Mobilna aplikacija - prikaz razmjene valuta [autorski rad]



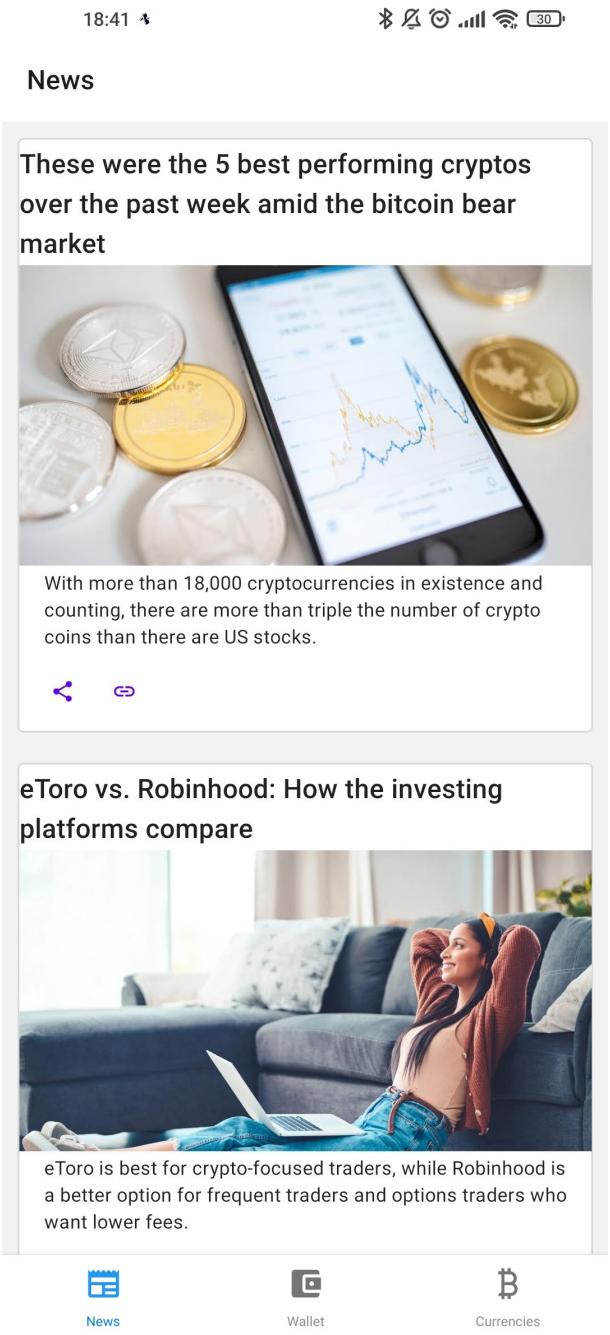
Slika 30: Mobilna aplikacija - prikaz novčanika nakon akcija [autorski rad]



Slika 31: Mobilna aplikacija - prikaz popisa podržanih valuta [autorski rad]



Slika 32: Mobilna aplikacija - označavanje favorita [autorski rad]



Slika 33: Mobilna aplikacija - prikaz novosti nakon odabira favorita [autorski rad]

9. Zaključak

Moderan Web postao je jedna od najimpresivnijih dostignuća čovječanstva. S porastom prosječne računalne snage, svako računalo danas može pokretati napredne aplikacije bez previše muke. S obzirom na popularnost Weba kao platforme, broj Web razvojnih programera definitivno je u rastu što znači da je sve više spretno u korištenju modernih Web tehnologija. Naravno, Web aplikacija nije uvijek najbolje rješenje kod razvoja programskog proizvoda. Za puno vrsta aplikacija primjerenije je da se razviju kao izvorne aplikacije, bile one stolne ili mobilne aplikacije. Ako se sve ovo uzme u obzir, izvorne aplikacije razvijene modernim Web tehnologijama postaju primamljivo rješenje za velik broj slučajeva razvoja novog programskog proizvoda.

U ovom su radu bile razvijene dvije aplikacije - jedna stolna i jedna mobilna. Obje aplikacije su izvorne aplikacije, no razvijene su modernim Web tehnologijama. Stolna aplikacija razvijena je *Electron* JavaScript okvirom, dok je mobilna aplikacija razvijena *React Native* tehnologijom. Obje tehnologije jako su jednostavne za korištenje, pogotovo kad se razvojni programer bolje upozna s tehnologijama. Znanje i iskustvo razvoja Web aplikacija definitivno se prenosi na razvoj izvornih aplikacija primjenom ovih tehnologija i uvelike olakšava učenje i sam razvoj aplikacija. Osim toga, podrška zajednice na Webu je ogromna i nije problem pronaći dobro dokumentirane primjere korištenja navedenih tehnologija. Uz to, službena dokumentacija obje tehnologije izrazito je detaljna, kvalitetna i lako razumljiva što uvelike utječe na iskustvo korištenja tehnologija.

S druge strane, performanse aplikacija su definitivno slaba strana ovih tehnologija što se očituje u tome da je potrebno više ciklusa procesa i više memorije da se obavi neki posao. Ovdje konkretno to nije bio problem jer se radi o relativno jednostavnim aplikacijama, no lako je vidljivo kako bi ovakav nedostatak utjecao na razvoj, a i korištenje, neke naprednije aplikacije poput aplikacije za uređivanje video sadržaja.

Najveća prednost korištenja modernih Web tehnologija za razvoj izvornih aplikacija definitivno je bila lakoća korištenja. Nakon uvodnog perioda upoznavanja tehnologija, razvoj je bio izrazito tečan i nije se pretjerano razlikovao od standardnog Web razvoja. Alati koji su dostupni svim razvojnim programerima su besplatni i funkcioniraju jako dobro. Jednostavno je testirati aplikacije tokom razvoja, a izgradnja finalnog proizvoda svodi se na upisivanje jedne naredbe u sučelje konzole.

Popis literature

[1] Microsoft. „Microsoft Documentation.” (2021.), adresa: https://docs.microsoft.com/en-us/windows/apps/desktop/ (pogledano 10. 12. 2021.).

[2] GTK Team. „GTK Documentation.” (2022.), adresa: https://www.gtk.org/docs/ (pogledano 5. 12. 2021.).

[3] The Qt Company. „Qt Documentation.” (2022.), adresa: https://doc.qt.io/ (pogledano 5. 12. 2021.).

[4] Apple. „Apple developer documentation.” (2021.), adresa: https://developer.apple.com/documentation (pogledano 11. 12. 2021.).

[5] Google. „Android developer documentation.” (2021.), adresa: https://developer.android.com/docs (pogledano 11. 12. 2021.).

[6] Mozilla. „MDN Web Docs.” (2022.), adresa: https://developer.mozilla.org/en-US/ (pogledano 10. 1. 2022.).

[7] T. Ater, *Building progressive web apps: bringing the power of native to the browser*. " O'Reilly Media, Inc.", 2017.

[8] Apache Software Foundation. „Apache Cordova documentation.” (2022.), adresa: https://cordova.apache.org/docs/en/latest/ (pogledano 29. 12. 2021.).

[9] D. DeVault. „Electron considered harmful.” (2016.), adresa: https://drewdevault.com/2016/11/24/Electron-considered-harmful.html (pogledano 4. 1. 2022.).

[10] I. Flechais, M. A. Sasse i S. M. Hailes, „Bringing security home: a process for developing secure and usable systems,” *Proceedings of the 2003 workshop on New security paradigms*, 2003., str. 49–57.

[11] I. Malavolta, „Beyond native apps: web technologies to the rescue!(keynote),” 2016.

[12] C. James. „Hybrid Mobile App Security.” (2022.), adresa: https://www.deloittedigital.com.mt/blog/securing-hybrid-mobile-applications (pogledano 12. 1. 2022.).

[13] J. Lebensold, *React native cookbook: bringing the web to native platforms*. " O'Reilly Media, Inc.", 2018.

[14] E. Masiello i J. Friedmann, *Mastering React Native*. Packt Publishing Ltd, 2017.

[15] N. Dabit, *React Native in action: developing iOS and Android apps with JavaScript*. Manning, 2019.

[16] B. Eisenman, *Learning react native: Building native mobile apps with JavaScript*. " O'Reilly Media, Inc.", 2015.

[17] Facebook. „React Native Documentation." (2022.), adresa: `https://reactnative.dev/`.

[18] D. Sheiko, *Cross-platform Desktop Application Development: Electron, Node, NW. js, and React: Build desktop applications with web technologies*. Packt Publishing, 2017.

[19] A. D. Scott, *JavaScript everywhere: building cross-platform applications with GraphQL, React, React Native, and Electron*. O'Reilly Media, 2020.

[20] OpenJS Foundation. „Electron Documentation." (2021.), adresa: `https://www.electronjs.org/docs/latest`.

[21] P. Jensen, *Cross-platform desktop applications: Using node, electron, and Nw. js*. Simon i Schuster, 2017.

[22] M. Jasim, *Building cross-platform desktop applications with electron*. Packt Publishing Ltd, 2017.

[23] C. Griffith i L. Wells, „Electron: From Beginner to Pro," *Electron: From Beginner to Pro*, 2017.

[24] S. Kinney, *Electron in Action*. Simon i Schuster, 2018.

Popis slika

Popis tablica