UDC: 004.738.5:004.9 Info M str. 32-38

MODULARNI PRISTUP U RAZVOJU VEB INTERFEJSA SA REACT.JS
MODULAR APPROACH IN WEB DEVELOPMENT WITH REACT.JS

Autor: Dario Jahić

REZIME: Kroz konkretan projekat se prikazuje nov način razvoja modularnih veb aplikacija. Ceo projekat se deli na module koju su međusobno nezavisni ali imaju mogućnost međusobnog komuniciranja. Na taj način se olakšava timski rad na projektu, proces upoznavanja sa projektom je znatno kraći, održavanje projekta je olakšano i brzina razvoja je veća. Pored toga, ukazaće se na ključne tačke koje su usko grlo u ovom pristupu kao i kada nije pogodno koristiti ovaj pristup. Kroz rad se može videti način integracije modula, kao i način na koji se projekat deli na određene module. Dokaz koncepta će biti urađen na studiji slučaja veb bankarske aplikacije.

KLJUČNE REČI: modularni razvoj veb interfejs react.js

ABSTRACT: Following development of concrete project, a new way of developing modular web applications is presented. The whole project is divided into independent modules which will be able to communicate with each other. This facilitates teamwork on the project, the process of understanding project is significantly shorter, project maintenance is easier and the speed of development process is higher. Besides, key points that are a bottleneck in this approach as well as when it is not convenient to use this approach will be pointed out. The process of module integration, as well as how the project is divided into modules will be explained. Proof of the concept will be done in a case study of a web banking application.

KEY WORDS: modular principles web interface react.js.

1. UVOD

Ukoliko posmatramo razvoj složene veb aplikacije, ne postoji neki opšte prihvaćeni standard kad je u pitanju arhitektura projekta. Svaki programski jezik ima svoju predefinisanu strukturu projekta koja je dosta fleksibilna i čest je slučaj da se ona znatno razlikuje od projekta do projekta, čak i za projekte koji su relativno slični [1].

Kao i za sve ostale složene probleme bilo u programiranju ili u nekim drugim životnim situacijama, najbolji pristup rešavanju velikih problema jeste podela problema na više malih. Vodeći se tom logikom, nastala je ideja za deljenje monolitnog projekta na više manjih celina odnosno modula, i na taj način podele odgovornosti između članova tima i olakšavanje procesa razvoja i održavanja projekata [2].

Jedan od uzroka problema je to što je ključna prednost u razvoju veb aplikaciije izdati gotovo rešenje u što kraćem vremenskom roku. To dovodi ne samo do toga da se piše nepregledan kod, već i da sam projekat ulazi u fazu održavanja i pre nego što se objavi finalna verzija. Uvođenje novog programera u takav projekat je jako težak proces koji dugo traje. To dovodi do situacije da se novom programeru prvo daje određeni vremenski rok da pokuša sam da protumači projekat nakon čega je neophodno i izdvajanje vremena drugog člana tima da u ne tako kratkom vremenskom periodu uvede novog kolegu u proces razvoja. Ovaj problem, pored vremena koji je jedan od ključnih resursa kada govorimo o razvoju veb aplikacija takođe zahteva i dodatne finansijske troškove koji mogu biti izbegnuti. Na bekend aplikacijama, možemo da vidimo da su se programeri suočavali sa dosta sličnim problemima. Kao rešenje ovih problema na bekendu su nastali takozvani mikroservisi [3].

Cilj istraživanja jeste da se predstavi novi način za razvoj frontend aplikacija. Kroz konkretan primer bankarske aplikacije, biće prikazan ceo proces uspostavljanja modularne strukture, počevši od analize projekta i potrebe za uvođenjem fleksibilnijeg pristupa u razvoju aplikacije pa sve do konkretnih problema koje ovaj pristup nosi sa sobom. Kako su bankarske aplikacije uglavnom monolitnog tipa, ovo predstavlja korak dalje u pristupanju rešavanja problema sa kojima se monolitni sistemi suočavaju.

Potencijalni nastavak ovog istraživanja jeste širenje koncepta modularne strukture tako da se moduli pišu u različitim programskim jezicima a da aplikacija i dalje funkcioniše bez ikakvih problema. Ovaj koncept je relativno nov na internetu i naziva se Microfrontends [4].Modularni pristup svakako otvara vrata za ideje ovog tipa, čiji je cilj da olakšaju programerima razvoj i proces održavanja projekta. Deljenje većeg problema na manje je pristup koji se koristi u raznim aspektima, pa dati pristup možemo da sretnemo i u dizajn paternima, kroz takozvane SOLID principe. Upravo prateći ove principe možemo da vidimo da je na jako jednostavan način rešen veliki broj opštih problema i to na način koji je lako primenjiv i predstavlja dugotrajno rešenje.

Mane koje ovaj pristup nosi sa sobom su upravo potrebno dodatno vreme za postavljanje ove strukture. Trenutno se u ovom istraživanju koriste određene biblioteke kao što su Redux i Redux Saga, koje su često korišćene u React i React-Native programskim jezicima, ali njih je moguće koristiti i u drugim Javascript jezicima.

Ideja celokupnog rada, je pokušaj da se istraži novi pristup u razvoju frontend aplikacija, u nadi da će u budućnosti prerasti u standard ili u najmanju ruku biti dodatna opcija za pristup rešavanju problema u zavinosti od potreba korisnika kao što je to slučaj sa mikroservisima u bekend aplikacijama.

2. PROBLEMI KOJI SE REŠAVAJU

Loša komunikacija između članova tima i nepregledan kod sa sobom povlače poteškoće u rešavanju novonastalih problema prilikom korišćenja aplikacije. Jako je bitno biti ažuran u

ovim situacijama i biti u mogućnosti da za što kraći vremenski period rešite sve greške koje dovode do prestanka rada aplikacije. Ovaj pristup nam upravo to omogućava, jer deljenjem projekta na nezavisne module, znatno je lakše locirati problem i rešiti ga bez uticaja na ostale delove aplikacije. Kod monolitnih sistema to često nije slučaj, već zbog isprepletenosti samog koda, rešavanjem jednog problema često prouzrokuje problem na drugom mestu i na takvim stvarima se gubi dosta vremena što nije slučaj sa modularnim sistemima.

Još jedan od glavnih razloga za primenu modularnog sistema je upravo fleksibilnost u primeni koju data struktura pruža. Ukoliko firma pruža jedan glavni proizvod sa svojim predefinisanim funkcionalnostima, često će taj proizvod delimično varirati od klijenta do klijenta. Tu se javlja nekoliko velikih problema koje ovaj pristup rešava. Problem u brzini potrebnoj da se napravi novi proizvod prilagođen za datog klijenta se drastično smanjuje jer se ovim pristupom izdvaja glavni deo projekta kao zasebna celina koja ima mogućnost jednostavnog prilagođavanja putem dodatnih modula. Sam proces održavanja, koji kod monolitnih projekata funkcioniše tako što se svaki projekat zasebno održava, je znatno olakašan. Bez modularnog sistema, čak i delove koji su zajednički za sve projekte, neophodno je izmeniti u svakom podprojektu pojedinačno. Ovom strukturom je to optimizovano, jer je glavni deo projekta izdvojen i koristi se u svim projektima sa mogućnošću izbora konkretne verzije koju će dati podprojekti da koriste. Pored toga, dodavanje nove funkcionalnosti je znatno lakše jer prateći postojeću strukturu može se samo dodati novi modul koji će zadovoljiti željene potrebe klijenta ne razmišljajući o tome da li će se novom funkcionalnošću uticati na već postojeće u vidu uzrokovanja nekih novih problema.

Ukoliko bi sumirali sve probleme koji se rešavaju modularnim pristupom i pokušali ukratko da ih navedemo to bi bili:

- Poboljšana komunikacija između članova tima koji rade uporedno na projektu
- Jednostavnija podela zadataka i odgovornosti između članova tima
- Smanjena je međusobna zavisnost između rada samih programera, tako što se do određene mere sprečava situacija da jedan programer zavisi od drugog
- Generisanje novih projekata, baziranih na jedan osnovni projekat, je znatno optimizovan
- Proces održavanja više projekata je takođe optimizovan, jer se održavanjem jednog projekta održavaju i većinski delovi svih njegovih podprojekata
- Proces upoznavanja novih programera sa projektom je znatno kraći
- Generisanje drugih tipova frontend aplikacija kao što je razvijanje mobilne aplikacije na osnovu postojeće veb aplikacije i obrnuto je znatno skraćeno
- Stvara se mogućnost jednostavne primene već postojećih funkcionalnosti i na projekte drugog tipa
- Olakšano je pisanje testova za projekat

Bitno je napomenuti da modularni pristup date probleme rešava u određenim tipovima projekta. To su projekti koji su po prirodi malo kompleksniji i od kojih se očekuje određena fleksibilnost, zato je bitno najpre uraditi određene analize o primeni samog modularnog pristupa o kojima će biti više reči kasnije u radu.

3. REACT.JS

Danas postoji zaista veliki izbor tehnologija za razvoj veb interfejsa. Iako se čini kao ne toliko bitna stavka kod izrade određenog projekta, neke kompanije troše i po više dana samo za donošenje odluke o pravoj tehnologiji. Kod odabira tehnologije za razvoj projekta ulogu igra više parametara, kako tip aplikacije koji treba da se izgradi tako i resursi sa kojima se raspolaže. S obzirom da je čest slučaj da za napravljenu veb aplikaciju klijenti uglavnom očekuju i mobilne aplikacije, React.js se činio kao idealno rešenje iz razloga što uz poznavanje React.js jako je lako proširiti znanje i naučiti i ReactNative kao jednu od danas najpopularnijih tehnologija za izradu mobilnih aplikacija. Sve bankarske aplikacije danas pored veb stranice zahtevaju i aplikacije kako za Android tako i za iOS operativni sistem [5]. Korisnici danas mnogo češće koriste telefon kao sredstvo za komunikaciju i obavljanja određenih aktivnosti kao što su plaćanje, slanje novca i drugih, što je jako bitan faktor za izbor programskog jezika kad je u pitanju razvoj bankarskih aplikacija [6] [7].

Pored toga, bitno je uveriti se da je tehnologija koja će se koristiti za razvoj veb i mobilnih aplikacija pouzdana, da ima dobru podršku, da bude popularna u smislu da je koriste velike kompanije, jer to automatski sa sobom donosi i niz drugih pogodnosti kao što su raspoloživost biblioteka, veličina zajednice, jednostavnost pronalaženja rešenja za određene probleme i drugih.

4. ANALIZA PROJEKTA ZA PRIMENU MODULARNOG PRISTUPA

Kad je reč o programiranju, nijedan pristup nije idealan za sve situacije. Problemi na koje programeri nailaze su jako raznovrsni i često je potreban neki specifičan vid prustupa problemu kako bi se on rešio. Sa druge strane, ukoliko posmatramo određene dizajn paterne i neke druge vidove najboljih praksi, možemo da vidimo da oni znatno olakšavaju pronalaženje rešenja za date probleme ili ih čak i preduprede [8]. Primenu dizajn paterna možemo da vidimo u skoro svim modernim tehnologijama, bilo da je u pitanju frontend ili bekend aplikacija [9].

Dokaz modularnog pristupa će biti prikazan kroz konkretnu e-banking aplikaciju. Poznato je da su aplikacije i platforme koje razvijaju banke ili firme iz Fintech[1] branše često velike. Samim tim potrebno je od samog početka uzeti u obzir faktore kao što su skalabilnost, održavanje, fleksibilnost, performanse i dr. U fintech svetu, veliki broj aplikacija su suštinski iste ili se razlikuju u određenim sitnicama. Prva ideja za optimizaciju

1 Reč nastala od termina Finansije i Tehnologije i odnosi se na svako preduzeće koje koristi tehnologiju za poboljšanje ili automatizaciju finansijskih usluga i procesa.

rada na ovakvim projektima i njihovom održavanju je da se izdvoji osnova (core) projekta i da se omogući njeno prilagođavanje u zavisnosti od projekta. Ovaj pristup izdvajanja zajedničkog rešenja odnosno izbegavanje ponavljanja koda se provlači kroz skoro sve paterne i predstavlja jednu vodilju koju treba pratiti i u drugim segmentima razvoja aplikacije kao što su i razvijanje samih komponenti.

Idući korak dalje, nakon što se definiše osnovni deo projekta koji će biti zajednički za sve podprojekte odnosno instance datog projekta, potrebno je osmisliti način na koji će data osnova biti razvijena. Često je moguće na osnovu bekenda predvideti celine koje će biti velikim delom nezavisne i na frontend aplikaciji. U slučaju da je bekend izgrađen kao sklop mikroservisa, za početak može i frontend deo da se podeli na module koji će pratiti date servise.

Pored bekenda, kao jedan od potencijalnih smernica šta bi moglo da se razdvoji kao poseban modul, mogu da budu putanje veb stranica ili rute u okviru navigacije. Rute se kreiraju često na način da prate određenu logiku između ostalog i zbog SEO[2] optimizacije. Na osnovu grupisanja tih ruta moguće je prepoznati potencijalne celine kao zasebne module.

Kad bi sumirali, ono što je ključno u primeru projekta koji se razvija primenom modularnog pristupa i zbog čega je odlučeno za taj pristup su sledeća svojstva projekta:

- Postoji glavni projekat koji će se dalje prilagođavati raznim klijentima uz određene dorade
- Glavni deo projekta je dovoljno kompleksan da je moguće podeliti ga na manje celine odnosno module, koji su međusobno nezavisni ili delimično zavise od drugih modula.
- Projekat zahteva dobru organizaciju i podelu između članova tima tako da se omogući jednostavan rad više ljudi na različitim segmentima projekta.
- Projekat će potencijalno imati specifične funkcionalnosti koje neće biti karakteristične za ostale instance istog projekta.
- Instance projekta će koristiti određene module u zavisnosti od ugovora sa klijentom (lite verzija / full verzija / custom verzija).

5. POSTAVLJANJE MODULARNOG PRISTUPA

Nakon analize projekta i uočavanja potrebe za fleksibilnijim pristupom u razvoju same aplikacije i odluke o odabiru modularnog pristupa na red dolazi njegova implementacija. Ono što je prednost kod ovog pristupa je da se jednom postavljena struktura lako prilagođava drugim projektima. Moguće je iskoristiti tkz. Boilerplate[3] koji predstavlja osnovu dovoljnu za početak rada na razvijanju veb platforme.

Druga prednost je da ukoliko već postoji projekat koji prati dati modularni pristup, moguće ga je integrisati sa novim projektima tj. iskoristi određene module koji su već implementirani uz male dorade. Uzmimo za primer Authentication modul, to je modul koji sadrži osnovne funckionalnosti vezane za autentikaciju korisnika kao što su registracija, prijavljivanje i drugi. S obzirom da bankarski sistemi moraju da ispoštuju određe regulative kao što je PSD2[4], u okviru modula je moguće postaviti određenu konfiguraciju koja će po potrebi uključiti dodatne korake za autentikaciju korisnika kao što su 2FA (two-factor authentication), SCA (Strong customer authentication), KYC (Know your customer) i drugi. Na taj način dobijamo određenu fleksibilnost gde na jednom mestu imamo kompletno funkcionalni modul za autentikaciju a po potrebi ostaje mogućnost da se isključe odnosno uključuje određene funkcionalnosti.

5.1. *Podela projekta na module*

Da bi na pravi način postavili strukturu projekta, potrebno je da se najpre dobro upoznamo sa projektom. Potrebno je detaljno pročitati specifikaciju projekta i razumeti što je više moguće šta će dati veb interfejs da omogućava i koje će funkcionalnosti da sadrži. Ovde je jako bitno dobro razumeti potrebe klijenta, jer klijenti često imaju problem da na jasan način izraze ono što žele. Iz tog razoga je popularna izjava: „Razumevanje klijenta i dobro napisana specifikacija je pola posla.“ jako česta u IT svetu. Nakon razumevanja projekta potrebno je prvo definisati osnovne funkcionalnosti koja će se koristiti u svim projektima [10]. U daljem tekstu će biti prikazana kratka speficikacija projekta na kome će biti primenjen modularni pristup.

Projekat predstavlja platformu koja služi za rešavanje problema malih i srednjih preduzeća kada je u pitanju digitalno bankarstvo. U pitanju su funkcionalnosti vezane za kreiranje naloga, poštujući sve regulative kao što je PSD2, upravljanje nalozima, integraciju više valuta, upravljanje karticama, praćenje statistike prometa, podržavanje P2P (Person-to-person) i P2C (Person-to-company) transfera, višejezičnost, notifikacije i obaveštenja i druge. Iz perspektive klijenta, korisnik će imati sledeće funkcionalnosti:

1. Kreiranje naloga i logovanje
2. Kreiranje naloga, uvid u listu naloga sa osnovnim informacijama kao i pristup pojedinačnom nalogu i pregledu detalja istog.
3. Upravljanje nalozima u vidu izmena podataka, brisanja, povezivanja kartica na određene naloge.
4. Kreiranje kartica, uvid u listu kartica sa osnovnim informacijama kao i prustup pojedinačnoj kartici i pregledu detalja iste.
5. Upravljanje karticama u vidu izmena podataka, brisanja, povezivanje i linkovanje za određene naloge.
6. Pristup isplatama, kreiranje novih isplata kompaniji, kreiranje novih isplata drugim korisnicima kao i zakazivanje isplata u određenim vremenskim intervalima.
7. Prikaz liste korisnika kojima se najčeće vrše isplate, kao i mogućnost isplate više od jednom korisniku odjednom.

2 SEO (Search engine optimization) – proces poboljšanja veb stranice kako bi se povećala vidljivost za relevantne pretrage.

3 Šablon ili deo koda koji se može ubaciti u određeni projekat i neometano radio uz malo ili nimalo promena.

4 PSD2 (second Payment services directive) – regulativa koja omogućava preduzećima da preuzmu podatke o računu korisnika uz njegovu dozvolu.

8. Mogućnost transfera sredstava između naloga i kartica
9. Prikaz statistike za pojedinačne kartice i naloge u vidu prometa sredstava
10. Opšta podešavanja naloga kao što su promena lozinke i slično
11. Lista notifikacija
12. Prikaz profila korisnika i mogućnost njegovog ažuriranja

Gore navedena lista predstavlja glavnu (core) funkcionalnost i ona bi trebala da se nalazi izdvojena i podeljena na module sa mogućnošću njenog proširenja. Evo kako su gore navedene stavke podeljene na module:

1. Authentication module (stavka 1)
2. Accounts module (stavke 2 i 3)
3. Cards module (stavke 4 i 5)
4. Payments module (stavke 6 i 7)
5. Transfers module (stavka 8)
6. Statements module (stavka 9)
7. Settings module (stavka 10)
8. Notifications module (stavka 11)
9. Profile module (stavka 12)

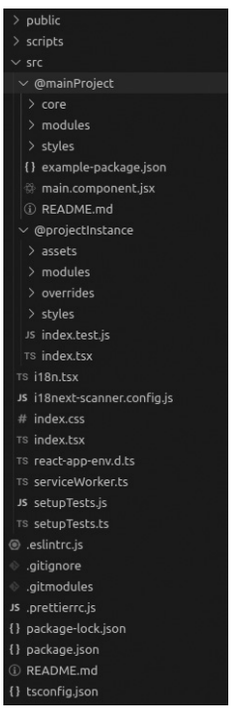
Modularnim pristupom ostavljamo prostora da i osnovna funkcionalnost ne mora nužno da bude korišćena za sve instance projekta. To znači da je ostavljen prostor da za određene projekte budu uključeni samo određeni moduli npr: Autentication, Accounts i Cards. U tom slučaju korisnici neće biti u mogućnosti da vrše isplate i transfere između naloga i kartica, već samo da imaju uvid u balans na računima kao i da upravlja nalozima ili karticima, npr. da zamrzne određenu karticu ukoliko je izgubljena i slično. Ukoliko bi u monolitnoj strukturi želeli da ubacimo funkcionalnost da određeni korisnici imaju mogućnost plaćanja a drugi ne, morali bi iz korena da menjamo logiku samih plaćanja kao i načina prikazivanja povezanih komponenti. To bi oduzelo mnogo vremena i verovatno učinilo kod dosta nepreglednijim i težim za dalje održavanje ili ubacivanje novih funkcionalnosti.

Modularnim pristupom se ovo jednostavno rešava tako što se prilikom učitavanja aplikacije sa bekenda prosledi određena konfiguracija modula za datog korisnika. Na osnovu te konfiguracije se generiše aplikacija i na taj način se zadržava puna kontrola nad aplikacijom i načinom na koji će se ona prikazivati i šta će omogućavati korisniku.

5.2. *Uspostavljanje osnove modularnog pristupa*

Kao što je navedeno, prilikom uspostavljanja modularnog pristupa vodiće se računa o dve glavne stvari: razdvajanje glavnog (core) projekta i njegovo rastavljanje na zasebne module. Razdvajanjem glavnog projketa od njegovih potencijalnih instanci ostavljamo prostora za dodatna prilogađavanja projekta drugim klijentima (generisanje novih instanci projekta). U glavnom projektu će biti postavljene sve vrednosti koje će se koristi kao demo projekat i glavni projekat bi trebalo da može da funkcioniše bez nekih dodatnih konfiguracija u podprojektu. Ideja je da se iz podprojekta poziva glavni projekat i prosleđuje određena konfiguracija preko koje će se override-ovati osnovni projekat tako da se njegov kod ne menja. Na taj način zadržavamo glavnu strukturu projekta a sve izmene i dorade koje su potrebne se vrše preko konfiguracije koja se prosleđuje glavnom projektu. Ovim pristupom omogućavamo jednostavno održavanje svih instanci glavnog projekta. S obzirom da će svi podprojekti koristi isti core projekat, ukoliko se u istom javi neki problem, rešavanjem tog problema će se automatski rešiti u svim instancama datog projekta.

React.js dolazi sa određenm strukturom projekta i nizom predefinisanih fajlova koji omogućavaju njegovo funkcionisanje. Programeri uglavnom date fajlove ne menjaju već sav svoj kod smeštaju u src folder. Takav je slučaj i sa modularnim pristupom. Na sledećoj slici možemo da vidimo kako izgleda struktura projekta.



Slika 1 – *Struktura aplikacije*

Kao što vidimo na slici, postoje dva glavna foldera: @mainProject i @projectInstance. MainProject folder predstavlja glavnu (core) funkcionalnost našeg projekta, dok se u projectInstance folderu nalazi konfiguracija koja se tiče konkretne instance Main projekta. Ukoliko prilikom pozivanja Main projekta ne prosledimo nikakvu konfiguraciju dobićemo osnovni odnosno demo projekat sa svom glavnom funkcionalnošću. Folder @projectInstance će da sadrži ono što će biti jedinstveno za datog klijenta. Pored slike logoa, fajla sa osnovnim bojama tu će se nalaziti i specifični moduli koji će biti jedinstveni za datog klijenta. Tu uviđamo još jednu prednost kad je u pitanju modularni pristup. Ukoliko klijent zatraži neki vid nove funkcionalnosti kao što je na primer integracija sa digitalnim valutama kao što su BitCoin, Etherium i drugi, uvek lako možemo da dati modul uvrstimo u glavnu funkcionalnost našeg projekta, i da nju ponudimo drugim klijentima bez ikakvih većih dorada.

Način na koji komuniciraju glavni deo projekta i podprojekat je tako što se u okviru podprojekta poziva glavna komponenta osnovnog projekta i prosleđuje joj se konfiguracija koja se koristi dalje kako bi izmenila određene delove. Konfiguracija može da sadrži sledeće elemente:

1. Reducers: objekat koji sadrži listu specifičnih reducera nevezanih za određene module
2. Modules: niz modula jedinstavenih za datu instancu projekta
3. Overrides: konfiguracioni fajl, koji sadrži listu komponenti koje su override-ovane iz osnovnog projekta.
4. Styles: konfiguracioni fajl sa specifičnim stilovima kao što su osnovne boje i drugi.

Na sledećoj slici možemo da vidimo kako izgleda pozivanje glavnog projekta iz podprojekta i način na koji se prosleđuje određena konfiguracija.

```
import React from 'react';
import modules from './modules/index';
import overrrides from './overrides/index';
import { IConfiguration } from '../@mainProject/core/types/index';
import Main from '../@mainProject/main.component';
import './styles/custom.scss';

const BaseProject = () => {
  const configuration: IConfiguration = {
    modules,
    overrides: overrrides
  };

  return (
    <div>
      <Main configuration={configuration} />
    </div>
  );
};

export default BaseProject;
```

Slika 2 – *Prikaz pozivanja glavnog projekta iz podprojekta i prosleđivanje konfiguracije*

Prilikom inicijalizacije aplikacije poziva se InitializerService-e koji prolazi kroz listu svih definisanih modula i proverava da li se oni nalaze u listi modula koja je dovučena sa bekenda. Ukoliko se nalaze, prolazi se kroz sve module i vrši se „ubrizgavanje“ glavnih komponenti modula kao što su Reducer-i i Saga-e i drugih ukoliko su definisani u okviru samih modula. To znači da svaki modul može biti posmatran kao zasebna celina odnosno aplikacija koja može biti sastavljena od komponenti, ekrana, navigacije, redux stanja i saga. Svi ovi elementi koji su eksplicitno definisani u okviru ModulContainer-a su javni i može im se pristupiti uz pomoć određenih komponenti koje se koriste za komunikaciju.

Da bi moduli funkcionisali na pravi način neophodno je definisati neki vid standarda odnosno interfejsa koji će dati moduli da prate. Da bi uveli tipizaciju, u sam projekat je ubačena i typescript biblioteka koja bi trebala da olakša razvoj aplikacije [11]. Nakon uvođenja tipova, lakše je definisati intefejse kao i tipove koji moraju biti ispraćeni kako bi se napravili moduli koji prate strukturu, odnosno data predefinisana pravila. ModulContainer predstavlja pristupačnu tačku modula i uz pomoć njega se pristupa određenim elementima i komponentima iz samog modula. ModuleContainer je sastavljen od sledećih komponenti:

1. ModuleName: naziv modula
2. Reducers: lista reducera specifični za konkretan modul
3. Sagas: lista saga middleware-a specifični za konkretan modul
4. Components: niz komponenti, kojima će biti moguće pristupiti van modula
5. Routes: lista ruta koje se odnose na konkretan modul

Konkretan primer zajedno sa definisanim tipovima ovih promenjivih može da se vidi na sledećoj slici.

```
/**
 * Module container is access point for each module.
 * Every module should have class that extends ModuleContainer.
 * App is injecting modules elements such as reducers, routes etc. on app start.
 * All classes that are extending ModuleContainer should be exported as object instance.
 */
class ModuleContainer implements IModuleContainer {
  reducers: ReducerType = {};
  sagas: SagaType = {};
  moduleName = '';
  components: ComponentType = {};
  routes: Array<RouteType> = [];

  getComponent(name: string): React.FC | React.ComponentType {
    return this.components[name];
  }

  getRoutes(): Array<RouteType> {
    // TODO: Return route base on permission
    return this.routes;
  }
}

export default ModuleContainer;
```

Slika 3 - *ModuleContainer - komponenta koja služi kao pristupna tačka za elemente konkretnog modula*

Da bi drugi moduli mogli da pristupe određenim komponentama iz modula, date komponente moraju biti navedene u okviru ModulContainer-a. Uzmimo za primer modul plaćanja (Payments module). U okviru njega je definisana komponenta koja na osnovu prosleđenog id-a prikazuje dugme za plaćanje i vrši isto ukoliko se na dato dugme pritisne. Da bi prikazali ovo dugme u okviru pojedinačne stranice naloga (Accounts module) iskoristićemo ExternalComponent komponentu koja će ukoliko je Payment modul uključen da izgeneriše dato dugme, a ukoliko je modul isključen neće prikazati ništa. Na taj način se osiguravamo da ne dolazi do grešaka prilikom komunikacije između modula i održavamo njihovu nezavisnost.

Konkretan primer Payments modula koji nasleđuje navedeni ModuleContainer može se videti na sledećoj slici.

```
src > @launchpad > modules > payments > TS PaymentsContainer.ts > ...
  import ModuleContainer from '../../core/modules/ModuleContainer';
  import PaymentsScreen from './pages/payments-screen.component';
  import NewPaymentButton from './components/new-payment/new-payment-button.component';
  import paymentsReducer from './redux/payments.reducer';

  class PaymentsContainer extends ModuleContainer {
    reducers = {
      payments: paymentsReducer
    };

    routes = [{ path: '/payments', component: PaymentsScreen, exact: true }];

    components = {
      'new-payment-button': NewPaymentButton
    };
  }

  export default new PaymentsContainer();

```

Slika 4 - *PaymentContainer - pristupna tačka za Payment module*

Kao što možemo da vidimo na slici, dovoljno je da definišemo klasu koja će naslediti ModuleContainer i u okviru date klase, stavke koje se odnose na dati modul. U ovom primeru, vidimo da Payment modul ima svoj reducer, svoju rutu i komponentu kojoj može da se pristupi na osnovu njenog unikatnog naziva. Na osnovu ovoga možemo da zaključimo da će ruta sa url-om /payments biti pristupačna samo ukoliko

je modul uključen. Da na datoj ruti se renderuje komponenta PaymentsScreen koja verovatno sa bekenda povlači određene informacije o paymentima i njih čuva u svom stanju. U variabli components je sačuvan objekat koji sadrži unikatni naziv komponente i njenu vrednost. U ovom slučaju to je new-payment-button koji kao što možemo da predpostavimo, renderuje dugme koje će prikazati niz modala za odabir korisnika ili kompanije kome će data isplata biti izvršena.

Lista svih modula se nalazi u ModulesService klasi, koja pored toga sadrži i određene statičke metode kao što su:

1. getModule – vraća konkretan kontejner modula na osnovu prosleđenog imena modula
2. getModules – vraća listu svih definisanim modula
3. addModules – služi sa ubacivanje novih modula iz podprojekta

5.3. *Prikaz načina komunikacije između modula*

U praksi će čest slučaj biti da je potrebno na neki način vršiti komunikaciju iz samih modula. Uzmimo za primer e-banking aplikaciju. Ukoliko smo je podelili na module Accounts, Cards, Payments i druge, vrlo je verovatno da će nam u okviru Accounts modula biti potrebno da izvršimo neku funkciju iz Payment modula ili da prikažemo dugme New Payment koje će na osnovu prosleđenog ID-a naloga da izvrši isplatu na određeni račun. Sa druge strane, treba imati u vidu da neće svi korisnici imati mogućnost da vrše plaćanje sa svog naloga ili određene instance aplikacije neće imati Payment modul uključen.

Da bi ovo moglo da funkcioniše na pravi način, potrebno je napraviti određene komponente koje će vršiti komunikaciju između datih modula. Komunikacija se vrši tako što se najpre proveri da li korisnik ima permisije za datu komponentu kao i da li je modul koji se zahteva uključen u datom projektu na osnovu konfiguracije koja se prosleđuje s bekenda. Tek nakon toga se proverava da li je komponenta koja je tražena javna (da li drugi moduli mogu da joj pristupe) i onda se, ukoliko su svi ovi uslovi ispunjeni, data komponenta prikazuje.

ExternalComponent je komponenta koja će se koristiti za pristup komponentama iz drugih modula. Imali smo gore navedeni primer PaymentContainer-a koji omogućava pristup komponenti koja generiše dugme za vršenje isplate. Na sledećoj slici možemo da vidimo kako izgleda navedena komponenta.

```
/**
 * External component is used to render component from other module if module is enabled.
 * It will return null otherwise.
 */
const ExternalComponent: React.FC<IExternalComponent> = ({
  moduleName,
  componentName,
  componentProps,
  children,
  ...otherProps
}) => {
  // TODO: Check if user has permission for module specific data
  const EComponent = ModulesService.getModule(moduleName)?.getComponent(
    componentName
  );
  // If module not exist return null
  if (EComponent === undefined) {
    return null;
  }
  return (
    <EComponent {...componentProps} {...otherProps}>
      {children}
    </EComponent>
  );
};

export default ExternalComponent;
```

Slika 5 - *ExternalComponent - komponenta koja služi za komunikaciju između modula*

Vidimo da je u pitanju relativno prosta komponenta, koja od parametra prima naziv modula, naziv komponente, parametre kao i podkomponente ukoliko postoje. Data komponenta uz pomoć ModuleService-a proverava da li modul postoji kao i zadata komponenta i ukoliko ne postoji vraća null. Ovo je jako bitan deo, jer se na ovaj način izbegava prikaz greške i dobijamo situaciju da moduli mogu besprekorno da funkcionišu bez obzira da li je neki drugi modul uključen ili nije. Upravo ovo predstavlja jedan deo nezavisnosti modula, gde moduli mogu da funkcionišu bez obzira da li oni koriste određene elemente iz drugih modula ili ne. Ukoliko je modul uključen i komponenta definisana u ModuleContainer-u, ExternalComponent će prikazati datu komponentu i proslediti joj parametre ukoliko ona to zahteva.

Na sledećoj slici možemo da vidimo kako se u okviru Accounts modula poziva Payments modul koji treba da vrati dugme za izvršavanja novog plaćanja.

```
<ExternalComponent
  moduleName="payments"
  componentName="new-payment-button"
  componentProps={{
    accountID: account.id,
    disabled: isDisabled
  }}
/>
```

Slika 6 - *Prikaz komponente iz drugog modula uz pomoć ExternalComponente*

Vidimo da se pored naziva modula i naziva komponenti prosleđuju i parametri accountID i disabled, koji redom treba da označe sa kog naloga se vrši isplata kao i da li je dugme uključeno ili isključeno u zavisnosti od određenih uslova.

5. PRIKAZ PRIMENE PRISTUPA U REACTNATIVE TEHNOLOGIJI

Bitna činjenica za primenu modularnog pristupa u izradi mobilne aplikacije je upravo da ReactNative kao i React.js ima integraciju sa Redux i Redux Saga bibliotekama. Način na koji ove biblioteke funkcionišu u ReactNative-u je identičan kao i kod React.js-a. To znači da za postavljanje modularnog pristupa kod razvoja mobilnih aplikacija može da se iskoristi većinski deo strukture koja je korišćena prilikom izgradnje veb interfejsa. To znači da sam način pokretanja i funkcionisanja aplikacije ostaje isti. I u ReactNative-u su kreirana dva glavna foldera gde je u jednom smešten glavni projekat a u drugom podprojekat koji će na isti način koristiti identičnu komponentu (ExternalComponent) da vrši izmene na projektu i komunikaciju između modula. S obzirom da su pristupne tačke i servisi koji su korišćeni za funkcionisanje modularnog pristupa na veb klase, i njih je u celosti moguće iskopirati i prebaciti da rade sa ReactNative-om.

Sve ove činjenice navode na to da je modularni pristup jedan korak ka jednostavnijem i bržem načinu razvijanja frontend projekata. Prateći istu logiku, dolazimo u situaciju da iz jednog glavnog projekta proizlaze različite veb i mobilne aplikacije. Funkcionalnost i suština na koje ove dve platforme funkcionišu je skoro identična, jedina razlika koja može da se pronađe je upravo u tipovima komponenti koje koriste za prikaz određenog

sadržaja. Ono što je u veb aplikaciji bilo prikazano uz pomoć div taga, u mobilnoj aplikaciji je prikazano preko View komponente i tako dalje. To su činjenice koje suštinski ne utiču na razvoj i održavanje celokupnog sistema. Čak šta više, prebacivanje modularnog pristupa sa veb aplikacije i generisanje od istog mobilnu aplikaciju dovelo je do toga da je mobilna aplikacija videla svoju prvu verziju za manje od mesec dana. Treba napomenuti da se za razvoj iste mobilne aplikacije bez modularnog pristupa, period njenog razvijanja procenjivao i na nekoliko meseci.

6. ZAKLJUČAK

Primena modularnog pristupa na nekoliko projekata je ukazala na sve potencijalne probleme kao i koristi dobijene njegovom implementacijom. Deljenjem projekta na module, povećava se fleksibilnost, znatno je lakše raditi na razvoju projekta kao i održavati isti. Lakše je uočiti uzroke problema i mnogo lakše testovima prekriti ceo projekat. Modularni pristup daje tu mogućnost jednostavnog generisanja niza sličnih projekata baziranih nad istom funkcionalnošću. To omogućava bolje plasiranje na tržište u svetu gde se veliki broj kompanija utrkuje da ponudi svoj proizvod. Održavanjem glavnog projekta, vrši se održavanje svih podprojekata nastalih iz istog. Na taj način lako se preduprede greške, koje većina klijenata i ne stigne da primeti, što znatno povećava njihovo zadovoljstvo.

Isto tako, modularni pristup sam po sebi nije savršeno rešenje. Razumevanje funkcionisanja ovog pristupa, učenje biblioteka uz pomoć kojih ovaj pristup funkcioniše mogu da oduzmu određeni vremenski period, te je jako verovatno da je za projekte manje kompleksnosti neisplativo koristiti ovaj pristup. U ovom radu se moglo videti nekoliko saveta i smernica koje bi trebalo uzeti u obzir pre nego što se krene sa razvijanjem nekog projekta. Kao i kod odabira programskih jezika, odabir arhitekture koja će se koristiti za izgradnju datog projekta, često zavisi od samog projekta. U nekim situacijama će modularni pristup znatno olakšati posao i ubrzati proces razvijanja i održavanja, dok će na drugim projektima možda više vremena oduzeti postavljanje date arhitekture i praćenje logike iste prilikom razvoja nego izrada celog projekta.

Razlaganjem projekta na module ne znači nužno pisanje manje programskog koda nego kada se sve nalazi na jednom mestu. Naprotiv, deljenjem problema na manje celine često sa sobom povlači upravo veći broj linija koda koji kao manje celine mogu da funkcionišu i da se primenjuju na više različitih mesta. Treba imati u vidu da je često mnogo lakše snaći se u kodu koji je podeljen na više manjih delova, nego u programskom kodu gde je sve međusobno isprepleteno. Iako je možda u početku vremenski zahtevnije napisati više manjih komponenti nego svu funkcionalnost ubaciti u jednu komponentu, treba uzeti u obzir da je mnogo lakše razumeti i održavati sistem koji je jasno podeljen na date celine.

Iz tog razloga jako je bitno uraditi dobru analizu pre početka rada na projektu. Za analizu možete uzeti u obzir i stavke koje su primenjivane na ovom istraživanju pre nego što je odlučeno za primenu modularnog pristupa. Pored prikupljanja neophodnih informacija o projektu od klijenata, potrebno je pokušati i predvideti u kom će se pravcu dati projekat razvijati. Šta je to što klijent možda nije uspeo da definiše kroz specifikaciju ili šta je to što klijent podrazumeva da će dati projekat da sadrži. Sve ovo obuhvata posao dobrog programera i vođe tima jednog projekta. Na osnovu svih tih informacija, vrši se procena vremena i radi se analiza troškova. Kao što smo videli u ovom radu, primenom modularnog pristupa, procena za izradu mobilne aplikacije koja je najpre bila procenjena na nekoliko meseci, dovela je do toga da je aplikacija svoju prvu verziju dočekala već za manje od mesec dana.

Po ugledu na koncept mikroservisa, i ovaj pristup potencijalno može da preraste u neki vid dobre prakse prilikom razvoja frontend aplikacija. Kroz ovo istraživanje je pokazana njegova primena u bankarskoj industriji, jer za banke važi opšti trend da koriste velike monolitne sisteme koji su davno napravljeni i imaju potrebu za njihovim unapređenjem. Osim te činjenice, poznata je izreka u svetu finansija „Vreme je novac.“ koja ukazuje upravo na važnost da se u što kraćem roku izađe na tržište i u njemu uspešno posluje. Svaki minut potrošen na rešavanju problema je potencijalni novčani trošak. S toga, modularni pristup upravo zastupa efikasnost u tom smislu, jer je jako lakše rešiti problem koji je nastao u okviru projekta podeljenog na nezavisne celine.

Na kraju, ništa nije dobar pokazatelj koliko je neki pristup kvalitetan i pouzdan dok se na njemu ne izgradi veći broj projekata i dok ne prođe određeni vremenski period. Sa ove tačke gledišta, na osnovu nekoliko projekata koji su bazirani na modularnom pristupu, može se reći da su utisci poprilično pozitivni. Bitno je da se i drugi programeri osećaju lagodno prilikom korišćenja ovog pristupa i da se još nije ukazalo na neki nerešiv problem. Na osnovu svega toga, problemi na koje se naišlo prilikom izgradnje ovog sistema su zanemarljivi. Jedino usko grlo prilikom primene ovog pristupa je bio upravo vremenski period potreban za njegovo osmišljavanje i njegovu realizaciju. S druge strane, taj problem jednom rešen više se ne dovodi u pitanje, jer svaki sledeći projekat može da koristi već postojeću strukturu i da funkcioniše bez problema.

7. REFERENCE

[1] https://dev.to/mmcshinsky/why-frontend-architecture-matters-1ldj
[2] https://medium.com/containerum/10-tech-challenges-that-are-solved-by-microservices-d91adeecb2e7
[3] https://dzone.com/articles/what-problems-do-microservices-solve
[4] https://www.robinwieruch.de/react-micro-frontend/
[5] https://medium.com/@alexmngn/from-reactjs-to-react-native-what-are-the-main-differences-between-both-d6e8e88ebf24
[6] https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet
[7] https://www.semrush.com/blog/fintech-user-behavior-trends-mobile-desktop/
[8] https://www.slideshare.net/t.bak/design-patterns-in-react-152397149
[9] https://reactpatterns.com/
[10] https://dev.to/jack/organizing-your-react-app-into-modules-d6n
[11] https://www.typescriptlang.org/docs/handbook/react.html

Dario Jahić
Kontakt: dariojahic95@gmail.com
Oblast interesovanja: IT, razvoj web aplikacija, arhitektura softvera