SVEUČILIŠTE U SPLITU

**PRIRODOSLOVNO MATEMATIČKI FAKULTET**

DIPLOMSKI RAD

**USPOREDBA IMPERATIVNE I DEKLARATIVNE PARADIGME U PROGRAMIRANJU KORISNICKIH SUCELJA**

Marino Dražić

Split, rujan 2022.**Temeljna dokumentacijska kartica**

Diplomski rad

Sveučilište u Splitu

Prirodoslovno-matematički fakultet

Odjel za informatiku

Ruđera Boškovića 33, 21000 Split, Hrvatska

**USPOREDBA IMPERATIVNE I DEKLARATIVNE PARADIGME U PROGRAMIRANJU KORISNICKIH SUCELJA**

Marino Dražić

**SAŽETAK**

Imperativna i deklarativna paradigma su dva glavna pristupa u programiranju. Dok imperativno programiranje se fokusira vise na kako izvršiti pojedini program korak po korak, deklarativno programiranje se vise fokusira na opisivanje sto točno želimo u kodu ali i ne nužno kako doci to tog rezultata. Ova dva programska principa će biti primijenjena na raznim alatima za razvoj korisničkih sučelja. Iako većina od ovih alata za razvoj i jezika koji podržavaju se polako prebacuju u funkcionalni pristup programiranju, ovaj rad će usporediti prednosti i ograničenja između ova dva pristupa, te kako se oni s novim programskim paradigmama.

**Ključne riječi**: Imperativna paradigma. Deklarativna paradigma, funkcionalna paradigma, Flutter, React, React-Native

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

**Rad sadrži:** \_\_ stranicu, \_\_ grafičkih prikaza, \_\_ tablice i \_\_ literaturnih navoda. Izvornik je na hrvatskom jeziku.

**Mentor:** **dr. sc. Saša Mladenović,** *izvanredni profesor Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

**Ocjenjivači:** **dr. sc. Saša Mladenović,** *izvanredni profesor Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

**dr. sc. Divna Krpan,** predavač *Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

**dr. sc. Goran Zaharija,** *poslijedoktorand Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Rad prihvaćen: rujan 2021.

**Basic documentation card**

Thesis

University of Split

Faculty of Science

Department of informatics

Ruđera Boškovića 33, 21000 Split, Croatia

**COMPARISON OF IMPERATIVE AND DECLARATIVE PARADIGM IN DEVELOPMENT OF USER INTERFACES**

Marino Dražić

**ABSTRACT**

Imperative and declarative paradigms are the two main approaches to programming. While imperative programming focuses more on how to execute the program step by step, declarative programming focuses on writing the code that describes what you want but not necessarily how to get it. These two programming principles will be applied to various frameworks that enable the development of user interfaces. While most of these frameworks and languages that they support are slowly transitioning to a functional approach to programming, this paper will compare the benefits and limitations between the two approaches and how they compare to the up-and-coming new programming paradigms.

**Key words**: Imperative paradigm, declarative paradigm, functional paradigm, Flutter, React, React-Native

Thesis deposited in library of Faculty of science, University of Split

**Thesis consists of:** \_\_ pages, \_\_ figures, \_ tables and \_\_ references

Original language: Croatian

**Mentor:** **Saša Mladenović, Ph.D.** *Associate Professor of Faculty of Science, University of Split*

**Reviewers:** **Saša Mladenović, Ph.D.** *Associate Professor of Faculty of Science, University of Split*

**Divna Krpan, Ph.D.** *Lecturer of Faculty of Science, University of Split*

**Goran Zaharija, Ph.D.** *Postdoctoral of Faculty of Science, University of Split*

Thesis accepted: September, 2021.

**Sadržaj**

[Uvod 1](#_Toc114689170)

[1. Okviri za testiranje 2](#_Toc114689171)

[1.1. Odabir alata 2](#_Toc114689172)

[1.1.1. Jest 3](#_Toc114689173)

[1.1.2. Mocha 3](#_Toc114689174)

[1.1.3. Jasmine 5](#_Toc114689175)

[1.2. Odabir tipa testa 6](#_Toc114689176)

[2. AGS 9](#_Toc114689177)

[2.1. Sadržaj AGS sustava 9](#_Toc114689178)

[2.1.1. Stvaranje testova 10](#_Toc114689179)

[2.1.2. Pokretanje testova 10](#_Toc114689180)

[2.1.3. Prikaz rezultata testa 12](#_Toc114689181)

[2.1.4. Izvršavanje testova 13](#_Toc114689182)

[2.1.5. Inteligentni pomoćnici 15](#_Toc114689183)

[2.1.6. Nextjs 16](#_Toc114689184)

[2.1.7. React 17](#_Toc114689185)

[Literatura 21](#_Toc114689186)

[POPIS SLIKA I TABLICA 22](#_Toc114689187)

# Uvod

U ovom diplomskom radu obraditi će se usporedba imperativne paradigme s deklarativnom paradigmom unutar konteksta izrade korisničkih sučelja. Osim usporedbe ovih dvaju primarnih programskih paradigmi, one će se usporediti s novim tipovima programiranja kao programiranje vođeno događajima i reakcijsko programiranje koji se mogu implementirati koristeći obije paradigme.

Fokus ovoga rada će biti same usporedbe unutar alata Flutter, te usporedba Fluttera kao alata s drugim popularnim alatima kao React i Angular. Ciljevi usporedbe alata su njihova adaptabilnost različitim programskim paradigmama i ograničenja pojedinog alata kao i programskih jezika koje oni koriste.

Rad je koncipiran u dvije cjeline.

U prvoj cjelini obraduje se tema paradigmi programiranja, njihovi početci, razvoj i trenutne primjene. Usporedbe između njih će biti teoretski prezentirane kao i njihovi nedostatci koji popunjavaju druge paradigme sto dovodi do razvoja multiparadigmicnih jezika.

U drugoj cjelini obraduje se tema paradigmi programiranja primijenjena na samim programskim alatima, multiparadigmički pristupi jezicima koje koriste alati za izradu korisničkih sučelja, te usporedba samih alata unutar konteksta primjene programskih paradigmi.

# Okviri za testiranje

Pri odabiru okvira za izradu projekta kljucnu ulogu također igraju alati za testiranje. Alat Nextjs koristi JavaScript jezik kao svoju podlogu te nas to ograničava na alate za testiranje u JavaScriptu. Sto bi do nedavno bio problem jer su alati za testiranje na webu bili teški za implementaciju, spori te nisu imali široku dokumentaciju kao današnji okviri. U zadnjih nekoliko godina okviri za testiranje na webu su se znatno stabilizirali sto je potaklo testiranje na webu kao jedan od integralnih dijelova izrade web projekata. O zadovoljstvu sa novim okvirima za testiranje nam govori anketa „The state of JavaScript“ koja prikazuje da 96% korisnika Jest okvira je jako zadovoljno. (stateofjs, 2021)

## Odabir alata

U svijetu JavaScripta izbora ima previše. Svima je poznato da novi JavaScript alat izlazi svaki dan te obećava biti bolji nego svi prijašnji. U ovome radu će se obraditi samo najpopularniji i najstabilniji okviri za testiranje.

Svaki okvir ima svoje ugrađene mogućnosti koje se mogu nadodati sa raznim proširenjima.

Table

Description automatically generated

Slika 1 Značajke popularnih alata

### Jest

Jest je okvir za testiranje koji je razvio Facebook. Nagli porast u popularnosti je doživio u 2017 te od tada drži prvo mjesto. U početku je bio temeljen na okviru Jasmine te je s vremenom Facebook implementirao svoje funkcionalnosti te dodao puno značajki povrh toga. Kljucne značajke Jesta su:

* **Performanse** Jest se smatra bržim za velike projekte s mnogo testnih datoteka zbog svog mehanizma pametnog paralelnog testiranja.
* **Korisničko sučelje** – Jednostavno i korisno, sve sto je potrebno
* **Odmah spremno** – Kao sto je vidljivo iz gornje slike okvir jest dolazi sa svim značajkama koje su potrebne za većinu projekata. Naravno Jest je moguće proširiti sa dodatnim značajkama vrlo lako.
* **Globali** – Kao u Jasmineu, Jest stvara globalne podatke te nema potrebe za dodatnim potraživanjem. Ovaj postupak se može smatrati lošim jer ograničava fleksibilnost i ograničava kontrolu nad testovima ali u isto vrijeme značajno olakšava živote običnih programera.

### Mocha

Mocha je najkorišteniji alat za testiranje. Za razliku od Jesta i Jasmine, Mocha sadrži samo okruženje za pokretanje testova te sve ostale značajke se moraju ugraditi koristeći dodatne ekstenzije. Najčešće ekstenzije su Sinon i Chai. Mocha je malo teza za postaviti u početku ali zato je nenadmašena u svojoj fleksibilnosti. Kljucne značajke Moche su:

* Zajednica – Mocha ima mnogo ekstenzija i proširenja za testiranje jedinstvenih scenarija kao i vrlo učestalih scenarija
* Proširivost – Mocha je lako proširiva do te mjere da su proširenja i ekstenzije posebno dizajnirane kako bi radile samo na Mocha okviru.
* Globali – Kreiranje globalne strukture koje ne ograničava fleksibilnost, no nažalost pruža dosta manje funkcionalnosti od alata Jest

#### Chai

Chai je BDD („Behavioral-Driven Development“) / TDD („Test-Driven Development“) okvir koji sluzi za potvrđivanje vrijednosti pojedinih varijabli. Glavna primjena je u okruženjima za Node i Web preglednike, te se može koristiti sa svakim JavaScript okvirom za testiranje. Chai ima nekoliko sučelja koji omogućuju korisniku da odabere onaj s kojim se osjeća najudobnije.

Text, letter

Description automatically generatedText, letter

Description automatically generatedText, letter

Description automatically generated

Slika 2 Različiti tipovi sintakse

Iako bi se moglo puno reći o tome koji je od ovih stilova poželjniji, trenutno vlada „Expect“ sintaksa. „Expect“ je funkcija koja uzima jedan argument, vrijednost koja se testira ili nadređenu vrijednost koja se testira, ovisno o testu.

Chai.js uključuje kozmetička svojstva koja nemaju utjecaja na ponašanje, ali umjesto toga dodaje prirodni jezik kako bi bio jasniji ljudima koji čitaju kod.

Figura 1 Kozmetička svojstva „to“, „be“

Proširujući prethodni primjer, dodavanje „to“ i „be“ ne mijenja ponašanje. Dodavanje kozmetičkih svojstava „to“ i „be“ jasno pokazuje što se očekuje od rijeci "foo".

Chai trenutno ima 15 kozmetičkih svojstava: to, be, been, is, that, which, and, has, have, with, at, of, same, but, does.

Chai također uključuje svojstva markiranja sa takozvanim zastavicama. Zastavica ne predstavlja tvrdnju. Umjesto toga ona postavlja zastavicu na lanac očekivanja koju druge tvrdnje u lancu mogu pročitati. Postojanje zastave samo po sebi ne mijenja ništa. Na pojedinačnim tvrdnjama je da odluče kako će protumačiti zastavu. Takva tumačenja mogu uključivati ​​negacije ili više ili manje stroge tvrdnje.

Figura 2 Zastavica "not"

Chai trenutno ima 7 zastavica: not, deep, nested, own, ordered, any, all

### Jasmine

Jasmine je testni okvir na kojem je baziran Jest. Jasmine ima ogromnu prednost u tome sto je na tržištu već dugo vremena te posjeduje mnoštvo članaka, alata i odgovorenih pitanja na raznim forumima koje su kreirane od strane zajednice. Angular tim službeno koristi Jasmine kao svoj okvir za testiranje. Kljucne značajke Jasnima su:

* Odmah spremano - Jasmine dolazi spreman sa svime sto je potrebno za započeti testirati na webu
* Zajednica – Jasmine je na tržištu još od 2009 te je skupio nenadmašne količine sadržaja. Nažalost u JavaScript svijetu stvari postanu beskorisne jako brze te većina starog sadržaja je u potpunosti neiskoristiva
* Angular – Podržava sve verzije alata Angular, te je službeno preporučen kao okvir za testiranje Angular aplikacija.

## Odabir tipa testa

Načini testiranja se razlikuju od jezika do jezika i od okvira do okvira, no tipovi testova su isti za sva okruženja. Kako je projekt baziran na testiranju studentovog koda potrebno je odlučiti kakav tip testa želimo izvršiti na njegovom kodu.

Imamo tri tipa testa:

* **Jedinici testovi** – testiranje pojedinih dijelova koda kao funkcije ili klase pružajući im ulaz te očekujući određeni izlaz.
* **Integracijski test** – testni proces koristeći vise jediničnih testova kako bi postigli cilj.
* **Funkcionalni testovi** – Testiranje scenarije u kojem Internet preglednik simulira akcije korisnika programatski. Ovi testovi inače ignoriraju način izrade aplikacije te je promatraju kao crnu kutiju koja mora na određenu akciju vratiti zahtijevani odgovor.

Sam primjer koda kojeg će se testirati nam otkriva koji test ćemo koristiti.



Figura 3 Primjer koda za testiranje

Kao sto vidimo kod je poprilično jednostavan te sadrži samo jednu klasu sa nekoliko svojstava. Za ovaj slučaj je najbolje koristiti jedinične testove kako bi smo mogli provjeriti zahtijevane elemente neovisno o sintaksnoj točnosti koda.

Zahtijevani testovi na ovom kodu mogu biti:

* Provjera sintakse
* Provjera postojanja konstruktora
* Provjera nasljeđivanja
* Zahtijevana svojstva
* Provjera postojanja funkcije
* Provjera nadopunjene funkcije

Svaki od ovih dijelova se može neovisno provjeriti te aplikacija vraća rezultat o točnosti svakog pojedinačnog dijela.

A screenshot of a computer

Description automatically generated with medium confidence

Slika 3 Zeljeni prikaz rezultata testa na ucenickom kodu

BLA BLA…

# AGS

AGS (Automated grading system) ili system za automatsko ocjenjivanje je projekt sa ciljem ubrzavanja i automatizacije jednostavnih testova u području objektno orijentiranog programiranja. Sustav se sastoji od 4 dijela.

## Sadržaj AGS sustava

AGS sustav je od početka konstruiran da bude modularan i proširiv. Prednosti ovakvog pristupa omogućavaju proširivanje na druga područja kao programiranje mrežnih aplikacija ili cak i predmete koji ne koriste JavaScript kao kljucni jezik. Sustav je razdvojen u vise dijela koji koriste minimalnu interakciju između dijelova

Diagram

Description automatically generated

Slika 4 Arhitektura AGS sustava

### Stvaranje testova

Kako bi smo mogli pokrenuti test na učeničkome kodu prvo je potrebno definirati sto točno nas test treba provjeriti. Iz tog razloga je izrađena stranica za stvaranje i izmjenjivanje testova. Dodavanje novih polja u test je lagano i intuitivno, no proširivanje same stranice kako bi se dodale dodatne mogućnosti ovisi o znanju Reacta i Nextjs okvira.

Diagram

Description automatically generated

Slika 5 Arhitektura sustava za stvaranje testova

Arhitektura sustava za stvaranje testova se sastoji od 2 dijela, stranice Create i Firestore baze podataka. Stranice create sadrži pod komponente FileForm i ClassForm koji omogućavaju dodavanje neograničenog broja datoteka i klasa koje treba provjeriti za vrijeme pokretanja testa.

### Pokretanje testova

Kako bi smo pokrenuli test prvo moramo odabrati koji test želimo pokrenuti. Jednom napisanu testovi su dostupni svim korisnicima bez obzira na lokaciju. Ovaj pristup je odabran kako bi olakšao dijeljenje testova između korisnika te kako bi omogućio korištenje sustava sa vise lokacija bez poteškoća. Pokretanje testa se sastoji od četiri faze.

Prvi korak je dohvaćanje testova sa Firestore servisa. Svi testovi se moraju držati istog formata podataka kako ne bi došlo do kasnijih greška. Testovi su zatim spremljeni u JSON formatu te imaju sljedeću strukturu:

Figura 4 Struktura testa

Iz strukture testa važno je istaknuti nekoliko polja. Polje „testParams“ je ulazna točka Mocha test pokretača, te trenutno podržava samo jedan test koji se može pokrenuti. Ovaj tip podataka je niz sto omogućava lagano proširenje u budućnosti. Polje „File“ sadrži lokaciju datoteke na kojoj je potrebno izvršiti test. Lokacija datoteke se nalazi u odnosu na svaku pojedinu datoteku unutar „Targets“ datoteke.

Drugi korak je odabir testa iz liste svih dohvaćenih testova. Nakon odabira testa on se prebacuje u drugi stupac koji ukazuje na trenutno odabrani test. Sa desne strane se također nalazi i treći stupac koji nam ispisuje sve pronađene direktorije u direktoriju „Targets“. Važno je naglasiti da sustav isključivo gleda direktorije te ostali tipovi nisu podržani.

Nakon odabira teste slijedi treći i zadnji korak koji se sastoji od pokretanja testa. Pritiskom na tipku Run šalje se POST zahtjev serveru na adresu "/api/runtest" sa svim podatcima trenutno odabranog testa. Nakon sto zahtjev stigne na server izvršava se zapis testnih podataka u JSON datoteku pod nazivom „testconfig.json“, te se zatim pokreće Mocha testni pokretač.

Figura 5 Pokretač testa na serveru

Iz gornje figure možemo vidjeti da pokretanje testova je u potpunosti neovisno o poslanim podatcima i odgovoru servera. Što znaci da nismo u mogućnosti znati trenutni status teste te njegovu uspješnost izvršavanja. Povezivanje servera i Mocha testnog pokretača nije moguće jer testni pokretač koristi odvojenu paralelnu nit kako bi izvršio testove, te sami testovi se izvršavaju u paraleli. Zbog ovoga ograničenja rezultati testa će se uvijek prikazati pet sekundi nakon uspješnog odgovora sa servera. Spajanjem sva tri koraka dobivamo finalnu arhitekturu pokretanja testa.

Diagram

Description automatically generated

Slika 6Arhitektura za pokretanje testova

### Prikaz rezultata testa

Nakon uspješnog pokretanja testa sa strane servera potrebno je prikazati rezultate testa. Mocha testni pokretač ima ugrađenu funkcionalnost izrade elegantne web stranice sa rezultatima testa. Za izradu rezultata se koristi proširenje zvano „Mochawesome“ koji generira „mochawesome-report“ unutar javnog direktorija. Generiranje rezultata unutar javnog direktorija omogućava jednostavan pristup rezultatima otvarajući novu stranicu u pregledniku sa lokacijom javnog direktorija.

Graphical user interface, text, application

Description automatically generated

Slika 7 Vizualni prikaz rezultata testa

### Izvršavanje testova

Nakon pokretanja testova koristeći dodijeljenu rutu na serveru, Mocha test pokretač je zaslužan za pronalazak svih datoteka unutar projekta koje se odnose na testiranje. Ove datoteke cesto imaju nastavak „.spec.js“. AGS projekt ima samo jednu datoteku za izvršavanje testova koja se zove „ags.spec.js“. Unutar te datoteke se nalazi sva potrebna logika za izvršavanje šest mogućih tipova testova.

Prije izvršavanja samog testa potrebno je pronaći sve datoteke unutar „Targets“ direktorija. Koristeći ugrađene mogućnosti Nextjs okvira možemo koristiti sve dostupne mogućnosti Node servera. Te mogućnosti uključuju skeniranje direktorija.

Nakon učitavanja svih direktorija vrijeme je za pokrenuti test na svim valjanim rezultatima.

Način izvršavanja testova je predstavljao jedan od temeljnih problema pri izradi projekta. Moguća su bila dva pristupa:

1. Prvi pristup se odnosio na učitavanje JavaScript koda unutar testnog okruženja te zatim izvršavanje zadanih testova. Ovaj pristup ima velike prednosti kao fleksibilnost testiranja svih mogućih zahtijeva i veću brzinu izvršavanja. Nažalost ovaj pristup također ima i veliku manu a to je nemogućnost testiranja sintaktički netočnog koda. Testno okruženje ne može prihvatiti i pokrenuti kod koji nije ispravan. Ako promotrimo ciljanu publiku na koja će pisati kod koji ćemo pokusati pokrenuti pronalazimo podosta sintaktičkih greška upravo mekog pristupa greškama unutar JavaScripta. Iz ovoga razloga način testiranja je morao biti prebačen na neki drugi način.
2. Drugi pristup se odnosi na učitavanje cijelog koda unutar JavaScript datoteke kao jedan dugi niz. Ovaj pristup uvelike ograničava fleksibilnost testiranja te stvari kao polimorfizam. Ovaj pristup je također bio preporučen od strane inteligentnog pomoćnika Copilot, te je odabran kao način izvršavanja testa

Nakon prihvaćanja drugog pristupa pri izvršavanju testova potrebno je napisati same testove. Ovdje također dolazi u pomoć pomoćnik Copilot koji je sam predložio i napisao sve testove unutar ovoga projekta.

Figura 6 Primjer testova korištenih unutar aplikacije

U gornjem prikazu imamo dva tipa testova koji prikazuju zašto smo morali odabrati drugi pristup pri izvršavanju testova. Gornji test provjerava jeli zadana klasa nasljeduje drugu klasu. Ovaj test je relativno lagan za izvršiti s obzirom na zadana okruženja rada sa jednim jako dugim nizom. Donji test provjerava valjanost koda tj. provjerava sintaktičku točnost primljene datoteke. U koliko datoteka nije ispravno napisana to ne predstavlja nikakav problem u izvršavanju drugih testova i provjera valjanosti ostalih zahtijeva.

### Inteligentni pomoćnici

Izrada AGS sustava je u velikom dijelu potpomognuta novim tehnologijama u umjetnoj inteligenciji. Istaknuti ću dva takva sustava koji stoje na raspolaganju svim programerima, te uvelike mogu promijeniti način izrade projekta

#### DALL-E 2

DALL-E 2 je novi system umjetne inteligencije baziran na kreiranju realističnih slika i umjetnosti iz zadanog teksta. DALL-E 2 kombinira koncepte, atribute i stilove različitih slika kako bi kreirao unikatna umjetničkih dijela. O uspjehu alata za kreiranje umjetnosti nam najviše govori nedavni uspjeh Midjourney sustava koji je osvojio nagrade na natjecanju u Coloradu. Iako DALL-E 2 nije osobito dobar u kreiranju teksta odabran je za izradu logotipa AGS sustava. Koristeći kratak opis projekta DALL-E 2 je uspješno generirao osam logo tipova za AGS sustav od kojih je odabran jedan.

Text

Description automatically generated with medium confidence

Slika 8 Odabrani logo

#### Github Copilot

Github Copilot je jedan od najvećih iskoraka u programiranju ikada. Github Copilot je programerski pomoćnik koju pruža niz mogućnosti kao automatsko dopunjavanje koda, generiranje čitavih funkcija i transformiranje komentara u kod. Važno je naglasiti da trenutno Copilot ne razmišlja kao programer i ne programira kao programer. Njegova posebnost dolazi iz ogromnog seta podataka koji se sastoji od većine koda ikada napisanog na Githubu. Sa tolikom bazom podataka Copilot je sposoban predlagati rješenja u svim programerskim jezicima i okvirima. Sa konstantnim korištenjem Copilot sve vise i vise razumije kod koji korisnik koristi te predlaže sve točnije i točnije rezultate. Prva verzija Copilota nije bila sposobna pogledati vise od trenutne datoteke te dati generalni prijedlog, dok trenutna verzija je sposobna pregledati čitav kod, te dati prijedloge u istom stilu kao i dosada napisani kod. Osim predlaganja koda Copilot također razumije kontekst podataka te uspješno razumije tip podatka te kakvu obradu je potrebno izvršiti kako bi se dobio željeni rezultat. Kao sto je očito Copilot je jedan od najboljih alata za pisanje testove, zbog potrebe da se svaki test opise detaljno prije pisanja Copilot u potpunosti razumije ulazne podatke te rezultat koji treba postići. U ovome radu 100% testova je automatski predloženo od stanje Copilota. Čvrsto preporučam ovaj alat svima koji čitaju ovaj rad, te se nadam da će Microsoft nastaviti sa uspješnim razvijanjem ovoga alata.

### Nextjs

Nextjs je JavaScript okvir inspiriran PHP-om te koristi JavaScript module, koji nam omogućavaju izvoz komponenti unutar aplikacije, što također omogućuje izvođenje pojedinačnih testova za svaku komponentu, kao i preuzimanje tisuća komponenti ili modula s NPM-a

Kada govorimo o aplikacijama u Next.js, moramo govoriti o CSS sustavu koji se zove „styled-jsx“ ili stilizirani JavaScript, ovaj sustav je posebno kreiran za rad s Next.js, te nam omogućuje da radimo sa svom snagom CSS-a izravno u JS datoteci.

Styled-jsx nam daje određene prednosti, na primjer, kada predstavljamo komponente samo generiramo CSS koji se koristi i, kada se komponenta više ne koristi, automatski uklanja CSS, što znači da nikada nećemo imati nepotreban CSS. Ovaj pristup smanjuje veličinu aplikacije te pruža brze vrijeme odaziva.

Fleksibilnost koju Nextjs nudi programerima i dizajnerima nema premca. Njegove značajke su brzina, responzivnost i prilagođenost SEO optimizira web stranicu za bilo koji uređaj ili razlučivost zaslona. Njegova analitika olakšava praćenje stvarnog učinka korisnika i automatski optimizira slike. Nextjs omogućava izradu potpuno prilagođenih stranica s jednostavnošću.

Next.JS je izvrstan izbor za projekte web razvoja, ali bitno je razumjeti osnove Reacta prije njegove upotrebe. Next.JS koristi usmjerivač koji se temelji na datotekama i podržava određeno dinamičko usmjeravanje. Osim ovih značajki, Nextjs također podržava TypeScript. Nextjs automatski konfigurira i kompilira TypeScript datoteke, osiguravajući sve prednosti TypeScripta bez problema sa postavljanjem i prevođenjem nazad u JavaScript.

Budući da je Nextjs okvir za izradu React aplikacija koje poslužujemo sa servera ključno je poznavanje Reacta

### React

React.js je JavaScript okvir koji je brz, siguran i skalabilan. Pruža fantastično iskustvo za korisnike i programere. Štoviše, njegova popularnost raste jer ga podržavaju Facebook i šira zajednica programera. React.js je dominantna JavaScript tehnologija za izradu korisničkih sučelja i postaje sve popularnija.

React je alat za izgradnju komponenti korisničkog sučelja i cijelih korisničkih sučelja – sve što se tiče sastavljanja vizualnih elemenata, povezivanja podataka s tim elementima i određivanja logike koja njime upravlja.

React.js se može koristiti za stvaranje korisničkih sučelja u JavaScriptu za različite platforme. ReactDOM se koristi za web aplikacije, React Native za razvoj mobilnih aplikacija (dijeleći većinu koda između Androida i iOS-a) i višeplatformske hibridne stolne aplikacije s Electronom. Nedavno je Microsoft također izdao React Native za Windows.

Dva su moguća pristupa pri korištenju modernih JavaScript okvira – prikazivanje na strani klijenta, gdje preglednik preuzima kod i prikazuje korisničko sučelje, ili prikazivanje na strani poslužitelja, pri čemu se korisničko sučelje prikazuje na pozadini.

Glavna značajka React.js-a koja ga razlikuje od ostalih popularnih JavaScript okvira je fleksibilnost. Možete zgrabiti biblioteku i koristiti je za prikaz jednostavne stranice ili pogleda, ali također možete kombinirati React.js s drugim alatima i koristiti ga kao okvir koji će postaviti temelje za složenu aplikaciju.

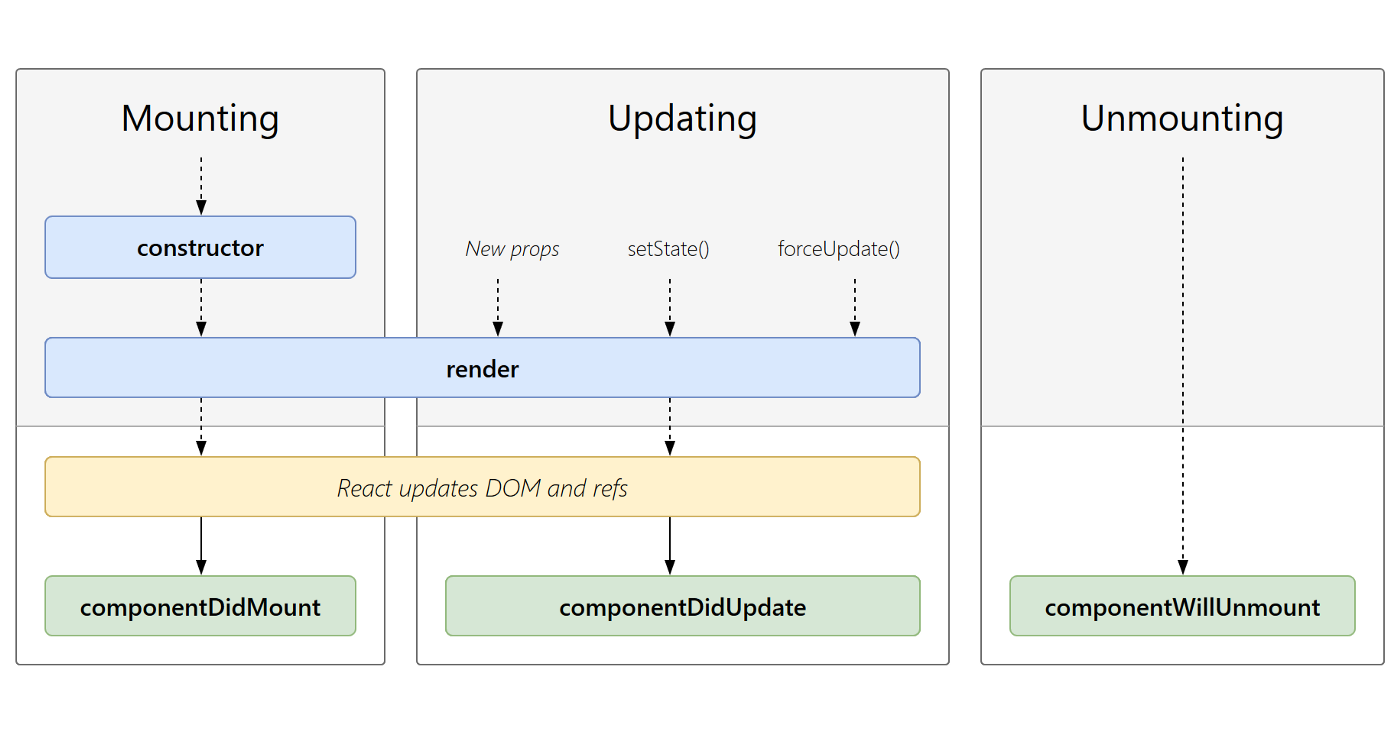
#### Komponente

Komponente su jedan od temeljnih blokova Reacta. Drugim riječima, možemo reći da će svaka aplikacija koju ćete razviti u Reactu biti sastavljena od dijelova koji se nazivaju komponente. Komponente uvelike olakšavaju zadatak izgradnje korisničkih sučelja, te inkapsuliraju pojedinačnu funkcionalnost. Možete vidjeti korisničko sučelje razbijeno na više pojedinačnih dijelova koji se nazivaju komponente i raditi na njima neovisno te ih sve spojiti u nadređenu komponentu koja će biti vaše konačno korisničko sučelje. One se mogu podijeliti na klasne i na funkcijske komponente.

##### Klasne komponente

Funkcionalne komponente lakše je konstruirati od React JS klasnih komponenti, koje su ES6 klase. Konstruktori, metode životnog ciklusa, funkcije renderiranja, kao i upravljanje stanjem ili podacima, uključeni su u svaku klasnu komponentu. Klasne komponente moraju proširiti React komponentu kako bi se mogle koristiti. Samo će React tada moći prepoznati da je ova specifična komponenta klasa i iscrtati ih ili vratiti potrebni React element.

Objekti klasne komponente imaju stanje, što znači da objekt može sadržavati informacije koje se mogu mijenjati tijekom životnog ciklusa objekta. Klasne komponente također mogu imati ulazna svojstva (poznata i kao „props“) koja im se prosljeđuju. Ulazna svojstva se prosljeđuju kao argumenti konstruktoru i trebaju se proslijediti klasi nadređene komponente pozivom „super(props)“. Podatci su dostupni tijekom cijelog vijeka trajanja objekta.



Slika 9 Životni ciklus React komponente

U Reactjs-u svaki proces stvaranja komponente uključuje različite metode životnog ciklusa. Ove metode nazivaju se životnim ciklusom komponente. Metode životnog ciklusa nisu jako komplicirane i pozivaju se u različitim trenucima tijekom vijeka komponente. Životni ciklus komponente podijeljen je u tri faze. One su:

* Montiranje (eng. *Mounting*)
* Ažuriranje (eng. *Updating*)
* Demontiranje (eng. *Unmounting*)

Montiranje je prva faza u životnom ciklusu komponente. To znači da je komponenta postavljena na DOM stablo. U ovome prvom koraku se definiraju njeni ulazni parametri, zatim nakon postavljanja prvog stanja komponente poziva se metoda „render()“ koja prikazuje ili iscrtava zadanu komponentu.

Nakon montiranja komponente ona sluša na bilo kakve promjene u njenom stanju i proslijeđenim parametrima. Prilikom promjene stanje pokreće se proces ažuriranja koji ponovo iscrtava komponentu sa novim stanjem.

Demontiranje je proces gdje se komponenta uništava svaki put kada napusti korisnikov prikaz tj. ekran.

##### Funkcijske komponente

Funkcijske komponente su samo normalne JavaScript funkcije. U današnje vrijeme se koriste isključivo funkcijske komponente, sto znaci da se sve metode opisane u klasnim komponentama vise ne koriste. Kao zamjenu za te metode React je uveo kuke ili „hooks“ koje postižu isti ili cak bolji rezultat nego njegovi klasni predci. Kuke nam služe kako bi smo upravljali životnim ciklusom komponente u odnosu na promjene u stanju i ulaznim podatcima.

Funkcijske komponente su znatno poboljšanje u odnosu na klasne komponente te pokušavaju riješiti ceste probleme kod klasnih komponenti kao sto su:

* Mijenjanje stanja komponenti kada je više ugniježđenih komponenti
* Kompleksne komponente postanu teške za čitati i razumjeti
* Klase su zbunjujuće u svojoj primjeni

Prednost funkcijskih komponenti je upravo to sto je jednostavna JavaScript funkcija. Kao takva uvelike je laska za pročitati i razumjeti. Samim time sto nisu klase nemaju potrebu za oslanjanje na kljucnu riječ „this“, što uvelike pomaže početnicima snalaženje unutar komponente. Također nemaju konstruktor sto znaci da stanje se mora spremiti na dugi način, a taj način je korištenjem „useState“ kuke koja daje lako vidljivu vrijednost varijable i način za promijeniti vrijednost varijable na pravilan način koji će uzrokovati pravilno mijenjanje stanja i ponovo iscrtavanje komponente.

# Literatura

BOHM, C., & JACOPINI, G. (1966). *Flow Diagrams, Turing Machines aAd Languages With Only Two.* International Computation Centre and Istituto Nazionale.

Brilliant, S. S., & Wiseman, T. R. (1996). *The first programing paradigm and language dilema.* SIGCSE '96.

Bulent Tugrul, Gunduc, S., & Eryigit, R. (2017). *An Analysis of Imperative Programming Languages from Data and.* Ankara, Turkey: ICTACSE, 2017 .

Hinsen, K. (2009). *The Promises of Functional Programming.* S cie n t i f ic P rogr a m m i n g.

Lloyd, J. (1994). *Practical Advantages of Declarative Programming.*

PALUMBO, D. (2021). *The Flutter Framework: Analysis in a Mobile Enterprise Environment.*

Pierce, B. C. (2002). *Types and Programming Languages.* The MIT Press.

stateofjs. (2021). *stateofjs.* stateofjs.

Torgersson, O. (1996). *A Note on Declarative Programming Paradigms and the Future of Definitional Programming.* Chalmers University of Technology and Göteborg University.

Waterlow, S. (1982). *The Third Man's Contribution to Plato's Paradigmatism.*

# POPIS SLIKA I TABLICA

Popis slika: