UNIVERSITETI “ISA BOLETINI” MITROVICË

FAKULTETI I INXHINIERISË MEKANIKE DHE KOMPJUTERIKE

DEPARTAMENTI: INFORMATIKË INXHINIERIKE

PUNIM DIPLOME

Mentori: Kandidatja:

Phd. Berat Ujkani Erëza Begu

Mitrovicë, Dhjetor, 2022

UNIVERSITETI “ISA BOLETINI” MITROVICË

FAKULTETI I INXHINIERISË MEKANIKE DHE KOMPJUTERIKE

DEPARTAMENTI: INFORMATIKË INXHINIERIKE

PUNIM DIPLOME

RESTful API në React

Mentori: Kandidatja:

Phd. Berat Ujkani Erëza Begu

Mitrovicë, Dhjetor, 2022

Mitrovicë, Dhjetor, 2022

Përmbajtja

1. Hyrje

Procesi i digjitalizimit paraqet një rrjedhë të shpejtë dhe gjithpërfshirëse në të gjitha sferat e jetës. Zhillimi i softurerëve po shkakton gjithnjë e më tepër rritje të bizneseve dhe rritje ekonomike kudo ne botë. Kërkesat e klientëve po bejnë që të kemi një ambient konkurues për zhvillim sa më konform nevojave të tyre. Për të zhvilluar softuerë modern dhe me një ndërfaqe të përshtatshme për përdoruesin e poashtu me një përmbajtje të formatuar mirë e me shpejtësi të navigimit, si temë diskutimi do të trajtojmë React JS tekonologjinë. React JS është librari e javascriptit nevoja e të cilit u ndje qysh para se të lansohej për herë të parë në vitin 2013. Paternat e fuqishme dhe mundesitë për zhillim që ofron React-i kanë bërë që zhvilluesit të fokusohen thellë në pjesën e strukutimit dhe mirëmbajtjes së kodit të tyre e që kjo pason me implementimin sa më korrekt të kërkesave në aplikacion. Ky punim përshkruan saktë dhe drejtë mënyren e implementimit të React JS, përdorimin e restful API për renderimin e aplikacionit dhe paternave të tij duke u përcjellur me shembuj konkret. Ndër të tjera, gjatë shtjellimit të temave iu referohemi një aplikacioni të zhvilluar në React dhe Nest JS duke përdorur rest api, i cili sqaron më tepër implementimin e React-it në zhvillimin e interfejsave modern dhe me kualitet të lartë.

Objekt studimi ne këtë punim është studimi i detajuar i perdorimit të restful api në React në mënyrë që lexuesit ti krijohet një pasqyrë e qartë mbi përshkimin e React JS si librari dhe praktikave më të mira të tij.

2. Hyrje në React

2.1 Zhvillimi në Web

Zhvillimi në Web i referohet krijimit dhe mirëmbajtjes së web aplikacioneve dhe websajteve e që përfshinë aspekte të ndryshme si dizajni i Webi-t, programimi në Web si dhe menaxhimin e databazës [1]. Zhvillimi në Web kategorizohet në këto ndarje:

- Front end development - ndryshe njihet edhe si "cilent side programming" i cili bën ekzekutimin e skriptave, e më së shpeshti ato të Javascriptit në browser. Paraqet kombinimin e HTML, CSS dhe Javascript për krijimin e një websajti me të cilin përdoruesi do të ndërveprojë direkt [1].
- Back end development - që zë vend në "server side programming" e që fokusohet në pjesën e pa dukshme të website-it nga ana e klientit dhe është përgjegjës për organizimin dhe ruajtjen e të dhënave si dhe funksionimin dhe qasjen e tyre nga pjesa e front-it [1].

Zhvillimi në Web përfshinë teknologji të ndryshme të cilat duhet të jenë në gjendje të krijojnë aplikacione të qëndrueshme dhe të përdorshme nga klienti, për këtë arsyje një numër i madh i teknologjive po marin eufori zhvillimi e një ndër to është edhe React-JS-i. Përgjatë këtij punimi ju do të njoftoheni me librarinë e javascriptit - React JS, përdorimin dhe praktikat e tij [1].

2.2 Çka është React JS?

React JS është një librari e Javascriptit e cila shërben për të ndërtuar UI(user interface) të fuqishëm, stabil dhe kompleks. React-i u bë publik për herë të parë në Maj të vitit 2013 dhe sot përben një ndër libraritë më të përdorura të front endi-t. Reacti është librari me kod të hapur, vjen falas dhe shërben për të ndërtuar aplikacione dinamike siç janë ato e-commerce apo edhe më të mëdha. Është librari e cila në vete përmban një komunitet të madh të programuesve dhe mirëmbahet nga Meta(Facebook) [2]. Shpesh ndodh ngatërresa në mes asaj se a është React-i një librari apo një framework? Për të kuptuar më saktë sygjerojmë se një librari paraqet një ndërtesë e cila ndërtohet nga fillimi ku strukturimin e objekteve brenda saj e bejmë sipas dëshirës, ndërsa një framework është një ndërtesë e ndërtuar për të cilën ne nuk brengosemi për ndërtimin e saj mirëpo nuk kemi qasje në strukturimin e objekteve brenda saj. Nga ana teknike React-i na ofron mundesinë e stilizimit të komponentave ashtu si na nevoiten për

zhvillim dhe për këtë arsyje React JS cilësohet si librari. React është një mjet i fuqishëm për ndërtimin e të ashtëquajturave SPA (single page applications) që paraqesin një teknologji të re në ndërtimin e aplikacioneve me përparesi të medha sepse webfaqja ngarkohet vetëm një herë dhe pas çdo përditesimi të të dhënave nuk ka nevoje të ngarkohet e terë webfaqja por vetëm kërkesa specifike që i dërgohet serverit. Ndërtimi i SPA me React JS bën që webfaqet të jenë një hap më përpara me shpejtesinë në kohë se sa websajtet tradicionale MPA (Multi page applications) [3].

2.3 Pse të zgjedhim React JS?

React-i është një librari e ndërtuar nga komponentat, secila komponentë paraqet një bllok ndërtues në aplikacionin tonë. Komponentat janë dinamike dhe të ri përdorueshme në tërë aplikacionin. Një komponentë mund të jetë p.sh një button, input, ikonë etj. Fuqia e përdorimit dhe kombinimit të komponentëve na ofron fleksibilitet dhe jo vetëm. Më poshtë gjeni aryjet kryesore pse të përdoret React JS:

- Krijimi i lehtë i aplikacioneve dinamike: React-i e bën më të lehtë krijimin e aplikacioneve dinamike sepse kërkon më pak kodim dhe ofron më shumë funksionalitet për dallim nga Javascripti, kompleksiteti i të cilit mund të rritet bukur shumë [2].
- Përformaca më e mirë: React-i përdor Virtual DOM-in i cili krahason gjendjen e më parshme të komponentëve dhe përditëson vetëm gjendjet që kan ndryshuar në DOM në vend se të përditesojë tërë komponentët [2].
- Komponentët e ripërdorshëm: Zakonisht një aplikacion i ndëruar në React përbehet nga disa komponentë, këta komponentë kanë logjikën dhe kontrollet e tyre dhe mund të ripërdoren gjatë tërë aplikacionit, kështu redukton kohën e zhvillimit të aplikacionit dhe shton fleksibilitetin e tij [2].
- Të dhënat një direksionale. React-i ndjek rrjedhjen e të dhënave në një drejtim me të cilin nënkuptohet se vendosja e komponentëve bëhet nga jashtë në brenda ose nga prindi në femijë dhe jo në të kundërtën. I tërë aplikacioni renderohet në një Id që emërtohet root dhe mbi të krijohen komponentët të cilët në vete përmbajnë vetëm dhe vetëm një prind, i cili mund të përmbajë një ose më shumë komponenta femijë.
- Sintaksa: React përdor një sintaksë të veçantë të quajtur JSX - javascript XML, e cila lejon shkrimin e Javascript-it në HTML.

- Tools për debugim: React-i ofron mjete të dedikuara për debugim më të lehtë, një ndër to është React components developer tool me te cilën bëhet më i lehtë procesi i testimit të komponentëve.
- React-i mund të përdoret për zhvillimin e aplikacioneve në web dhe atyre mobile: Eksizton një kornizë e quajtur React Native që është derivat i React JS dhe që mund të krijojë aplikacione mobile [2].

Gjitha keto që u thanë më sipër bëjnë që Reacti të krijojë një ekosistem stabil, kursim të kohës, rritje të produktivitetit dhe qëndrueshmërisë së aplikacioneve [1].

2.4 Karaktesistikat

Më poshtë gjeni të listuara disa nga karakteristikat kryesore të React JS:

- Virtual DOM
- States dhe Props
- Global state management
- Sintaksa e JSX

Virtual DOM.

Dom-i (Document Object Model) virtual në React paraqet një koncept fundamental. Kurdo që një dokument siç është HTML, ngarkohet në browser, një reprezentim i objekteve të atij dokumenti krijon një strukturë në formë peme. Kjo strukturë në vete paraqet Document object Modeli-n të cilin e njohim si DOM. Në React ndodh procesi i ri renderimit, proces ky i cili krijon një kopje të DOM-it që rrin gjithmonë i sinkronizuar me DOM-in real dhe quhet DOM-i virtual. Sa herë ndryshon një komponentë, ndodh procesi i ri renderimit dhe bën që të bëhet krahasimi i nyjeve të domit dhe virtual domit dhe për çdo nyje që ndryshon në virtual dom bën që të rirenderohet domi dhe të dhënat të përditesohen. Kjo paraqet bazën mbi të cilën është krijuar React-i e që bën që aplikacioni të jetë më i shpejtë për arsyje të renderimit të nyjeve të ndryshuara dhe jo të tërë DOM-it si në figurën 1. Në aplikacionet MAP (multi page aplications) kjo qasje nuk ka qenë fare e involvuar sepse për çdo ndryshim që ka

ndodhur në DOM, është dashur që tërë webfaqja të ri renderohet dhe kjo ka shkaktuar një proces mjaft të gjatë dhe të stërngarkuar të ngarkimit të një webfaqeje [4].

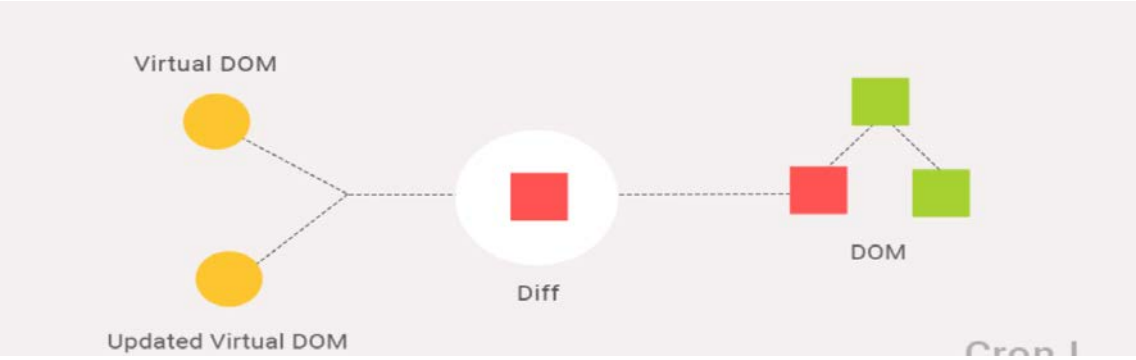


Figura 1 Përditësimi i DOM-it [4]

States dhe Props

Ashtu siç u përmend më sipër, komponentët janë blloqe ndërtuese dinamike e që dinamizmi i tyre bëhet më anë të props-ave. Komponentët marrin të dhëna nga prindi i tyre duke përdorur props të tipeve të ndryshme si string, number, react node dhe i renderojnë ato, ndërsa logjika e renderimit dhe menaxhimit të tyre bëhet duke përdorur states. Props janë të dhëna të lexueshme dhe jo të modifikueshme nga komponenta ndërsa states janë gjendje të brendshme të modifikueshme por që nuk mund të qasen nga jashtë përveq se me anë të menaxhimit global të gjendjeve siç është konteksti ose redux-i. Props mund të përcillen vetëm nga prindi tek fëmiu dhe jo anasjelltas. Me anë të states caktohet rrjedha e komponentëve dhe komponentët caktojnë rrjedhën e aplikacionit. [2]

Global state management

States në React janë logjikë brenda komponentës dhe nuk mund të nxirren jashtë saj. Nëse një state duhet të qaset në nyjet më të thella të DOM-it atëherë duhet të kalohet me anë të props nga komponenta në komponentë deri sa të arrijë tek nyja e dëshiruar. Kalimi i props nga komponentat në komponentë krijon efektin e quajtur “props drilling”, efekt ky që shkakton humbje të madhe të përformancës sepse krijon ri renderimin e secilës komponentë. Për arsyje të lejimit të qasjes së një state nga cilado komponentë, pavarësisht vendosjes dhe thellësisë së saj në pemën e objekteve u krijua menaxhimi global i states e më i njohuri është Redux-i. Redux krijon një “bazen” të quajtur store në të cilën vendosen states dhe behën të qasshme nga i tërë apikacioni

pavarësisht se në cilën radhë hierarkike është vendosur komponenta në DOM. Redux store përbehet nga states dhe reducers, reducers janë funksione që mundesojnë modifikimin e states dhe poashtu bëhen të qasshme për të gjitha komponentat ashtu siç shihet ne figurën 2. Reducers mund të ndryshojnë cilëndo gjendje nga cilado komponentë dhe ndryshimi do të afektojë të gjitha komponentët në të cilën është përdorur state-i në fjalë. Komponenta renderohet vetëm atëherë kur ndryshon një key në store dhe key i ndryshuar gjendet në komponentë. Redux ka përmirësuar rrjedhën e të dhënave në React në menyrë drastike dhe po bëhet përpjekje edhe për përsosjen e superseteve të tij siç janë Redux saga dhe Redux toolkit [5].

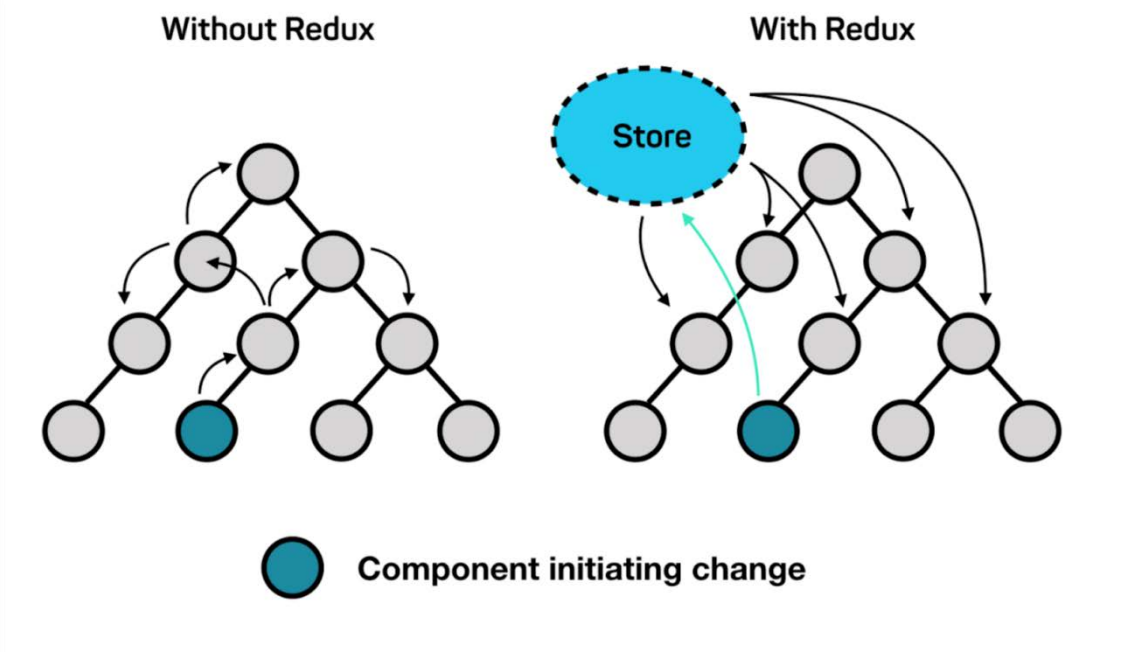


Figure 2 Redux global state management [5]

Sintaksa e JSX

Javascript XML njihet si shkurtesë për JSX. Është një sintaksë shtesë e Javascriptit. Është XML ose HTML sintaksë që përdoret brenda React JS. Kjo sintaksë konvertohet nga Reacti në Javacript. Versioni i fundit i javascriptit i njohur si ecmascript 6 e lejon që brenda HTML-së të shkruhet kod në javascript. JSX lexohet nga Babel compaileri dhe ai e përkthen dhe e konverton këtë gjuhë në Javascript dhe kështu ndodh ekzekutimi i skriptave. Për çdo element që krijohet në DOM, çfarëdo elementi qoftë ai, në prapaskenë ekzekutohet funksioni create element() ku secilin element e përkthen në objekt dhe e implementon në DOM. Nëse ne krijojmë një button p.sh

```
<MyButton>Click Me</MyButton>
```

Ateherë në prapaskenë Babel-i do ta implementojë në DOM duke përdorur këtë sintaksë:

```
React.createElement(
  MyButton,
   'Click Me'
).
```

JSX në vete përfshinë një set të rregullave shumë më të gjera se ato që u thanë më sipër [2].

3. Restful API në React

3.1 Komunikimi në mes frontend dhe backend

Back-endi dhe Font-endi punojnë së bashku për të ofruar një aplikacion me të cilin përdoruesi është i gatshëm të ndërveprojë, por si komunikojnë mes vete këto dy pjesë të rëndësishme të aplikacionit. Në detaje ky ndërveprim do të duket si më poshtë:

- Përdoruesi iu qaset një url-je me anë të një browserit:
    - Kjo bën që browseri të dërgojë një kërkese në server
    - Browseri pret derisa serveri të kthejë një përgjigjje
    - Përgjigjja që kthehet përdoret për të renderuar aplikacionin
- Përdoruesi pret derisa browseri të renderojë aplikacionin
- Përdoruesi sheh një ndërfaqe të pasur para vetës
- Përdoruesi ndërvepron me ndërfaqen
    - Dërgohen më shumë kërkesa në backend dhe me shumë të dhëna shfaqen në aplikacion…

Në spjegimin e mësiperm u permend se si pjesa e frontend-it dërgon kërkesa në pjesën e backendit, mirëpo për të realizuar këtë operacion më poshtë do të diskutojmë se çka janë API dhe si përmes tyre mundësohet komunimimi backend dhe frontend [6].

3.2 Si komunikon browseri me backend-in?

Browseri paraqet një aplikacion që funksionon në paisjen e perdoruesit. Përmes tij dërgohen kerkesat HTTP, merret pergjigjja dhe perpunohen të dhënat që merren. I gjithë komunikimi nga ana e përdoruesit ndodh duke perdorur browserin e tyre. Por qka paraqet HTTP kërkesa?

HTTP (Hypertext transfer protocol) është një protokol për transmetimin e dokumenteve siç është HTML, i dizajnuar për komunikimin në mes të një web browseri dhe një web serveri.

Një HTTP kërkesë përbehet zakonisht nga:

- Një HTTP metodë, zakonisht një metodë GET, POST, PUT, PATCH, DELETE. Këto metoda tregojë mënyrën se si një klient do të ndërveprojë me serverin. Zakonisht kur një përdorues dëshiron të marrë një pergjigjje përdor metodën GET.
- Një path të resurseve që do të kërkojë.

- Verisonin e HTTP protokolit.
- Ndonjë informatë shtesë për ti dërguar serverit përmes headers.
- Resurset që dëshiron ti dergojë në server permes body të mesazhit.

Dhe zakonisht pergjjigjja që kthehet përbehet nga :

- Versioni i HTTTP.
- Një status kod, qe tregon nëse kërkesa është realizuar me sukses apo jo.
- Një status mesazh, një përshkrim i shkurtër i status kodit.
- HTTP headers.
- HTTP body që përmbanë resurset e kërkuara.

Dërgimi i një kërkese inicohet nga një klikim që inicohet nga kodi i shkruajtur në javascript.

Në figurën e mëposhtme tregohet se si një klient dërgon një kërkese, serveri e lexon kërkesën që vjen dhe shikon nëse ka ndonjë resurs si një JS file apo ndonjë CSS file dhe pastaj e ekzekuton metodën që i dërgohet si në figurën e mëposhtme [7]. Pastaj serveri kthen përgjigjjen në formatin JSON dhe në bazë të përgjigjjes renderohet pamja e aplikacionit .

.

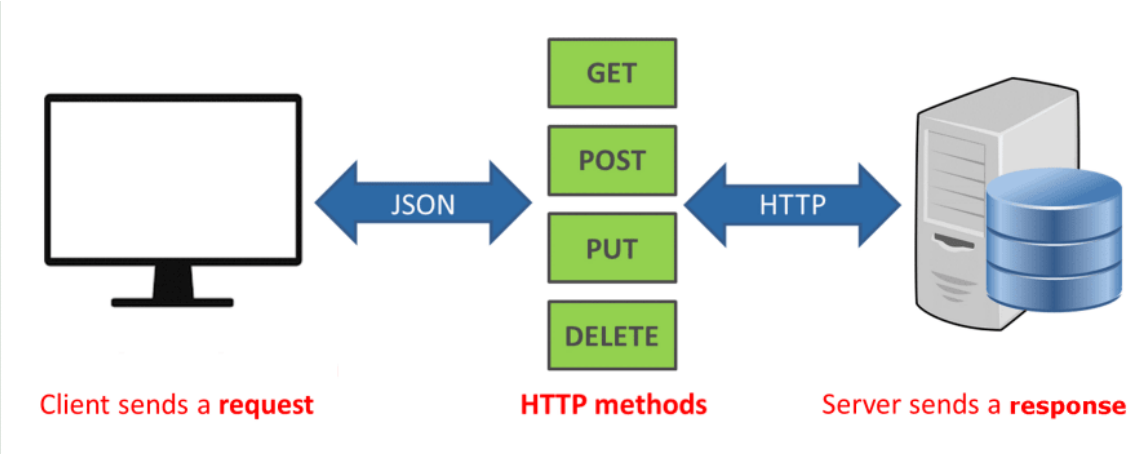


Figura 3 HTTP metodat [8].

3.3 Çka është një API?

API ose Application Programming Interface paraqesin një grup procedurash dhe funksionesh që ndihmojnë në krijimin e një aplikacionit. Përmes API bëhet qasja e të dhënave në servise, aplikacione apo edhe sisteme operative. Në esencë API paraqesin një ndërmjetësues për platforma të ndryshme softuerësh dhe lejojnë dy aplikacione të palidhura në mes vete të "komunikojnë" me njëra tjetrën. API është një rregullator që ndihmon në rregullimin e infrastrukturës së shërbimit të të dhënave. API-të poashtu përdoren në pothuajse të gjitha rastet në internet. P.sh nëse ekziston një objekt në google maps, ai objekt është aty për shkakt se google përdor një api për të shfaqur objektet në atë faqe të internetit. Nëse ky API nuk do të ekzistonte, zhvilluesit do të duhej të ndërtonin hartat e tyre dhe të ofronin të dhënat e tyre vetëm për të vendosur një objekt në hartë. Ekzistojnë disa lloje të API-s. Më poshtë gjeni të listuara llojet e API:
Web API: janë API që mund të qasen duke përdorur HTTP protokollin. API i definon pikat fundore për qasje, kërkesat valide dhe formatin e përgjigjjeve. Disa web API-s krijojnë një API të përberë e që njihet me emrin API service.

API e hapur: ndryshe njihen edhe si API publik ose të jashtëm, janë të qasshëm nga zhvilluesit por kanë disa kufizime në qasshmeri. Për përdorimin e tyre zakonisht kërkohet që perdoruesi të regjistrohet për të marur një API key të cilin e përdor për tu identifikuar dhe për të përdorur të dhënat që ai api kërkon.

API e mbyllur: ndryshe njihen edhe si API internal që lejojnë qasje vetëm në grupe të caktuara dhe janë të fshehur nga përdoruesit e jashtëm. Ky lloj API është shumë i frekuentuar sepse ruan integritetin e të dhënave dhe është shumë i avansuar në sistemin e sigurisë.

API i përbërë: i lejojnë përdoruesve qe me një kerkesë t'iu qasen shumë pikave fundore. Kjo mund të cilësohet një ndër API më modern sepse e redukton ngarkesën e serverit duke mundësuar kështu që me një kërkesë të ofrohen shumë shërbime [9].

3.4 Restful API

REST është një sinonim për REpresentational State Transfer. Arkitektura REST u përcaktua së pari nga Roy Fielding në vitin 2000. Që një API të jetë Rest ai duhet ti plotësoj kërkesat e mëposhtme te cilat janë:

- Përdorimi i një Uniform interface (UI): Resurset duhet të jenë unike ne një URL të vetme dhe vetëm me HTTP metodat e referuara më sipër do të jetë e mundshme të manipulohet me resurse.
- Client Server Based: Duhet te ekzisojë një ndarje e qartë në mes të pjesës së serverit dhe asaj të klientit dhe përgjegjësitë mes tyre duhet të jenë poashtu të ndara.
- Stateless: Kuptimi stateless bie në atë se nëse përfundohet një kërkesë dhe përgjigjja kthehet te klienti, serveri nuk do të jetë në gjendje që ta ruajë përgjithëmonë këtë kërkesë sepse ai do të mbyllë sesionin e kërkesës dhe do të procesojë kërkesën e radhës.
- Ruajtja në kesh: Ruajtja në kesh nënkupton se një pergjigje nga serveri mund të ruhet apo jo në kesh. Nëse ruhet ne kesh mund te bëhet e qasshme për një kohë më te gjatë të përdorimit dhe nuk shkakton një kërkesë të re përderisa të invalidohet nga keshi.

Në projektin e mëposhtëm përdoret REST arkitektura kudo. Me anë të rest arkitektures krijohen serviset dhe më pas bëhen të qasshme nga pjesa e serverit. Për të krijuar një servis kemi përdorur një paketë ndihmëse e quajtur Axios e cila rregullon mënyrën e parashtrimit të kërkesës edhe pse në React ekzison një build in funksion i cili quhet fetch mirëpo në krahasim me fetchin axiosi del të jetë një alternativë për projekte më të mëdha dhe të jetë më i shpejtë sa i përket performancës. Më poshtë do të demostrohet qartë edhe me pjesë te kodit se si axios instanca ndikon në krijimin e serviseve që do të përdorin RESTful API. Më tutje do të njoftoheni me paternat e dizajnit që paraqesin ecurinë e punës pasi të pranohet pergjjigjja nga serveri. Paternat luajnë një rol te veçantë në renderimin e të dhënave [9].

4. Çka janë paternat e dizajnit.

Në inxhinierinë softuerike, një paternë e dizajnit është një zgjidhje e përgjithshme dhe e përseritshme për një problem që shfaqet gjatë ndërtimit të softuerit. Është një përshkrim ose shabllon se si të zgjidhjet një problem. Dizajni efektiv i softuerit kërkon shqyrtim të problemeve dhe me qëllim të zgjedhjes së problemeve dhe lehtësimit të lexueshmërisë së kodit ka lindur nevoja e krijimit të paternave të dizajnit. Krahas zhvillimit dhe avansimit të React-it po zhvillohen edhe paternat për dizajnimin e tij. Më poshtë gjeni disa nga paternat kryesorë të dizajnit:

- The higher order component pattern
- Provider pattern
- React conditional rendering patterns
- React hooks pattern
- Compound components pattern
- Kompozicioni
- Container/Presentational Pattern [10].

4.1 The higher-order component pattern

High order components paraqet një teknikë të avansuar të ripërdorimit të komponenteve të React-it. Është një paternë e krijuar nga kompozicioni. Kompozicioni paraqet një teknikë në React që mundëson pranimin e komponentave si parametra të një komponenteje tjetër. HOC u frymëzua nga hoc funksionet në Javascript. Por qka janë HOC funksionet? Hoc funksionet për dallim nga funksionet primitive të javascriptit, janë funksione të javascriptit të cilat pranojnë si parametër një funksion dhe kthejnë një funksion në return. Ne mund të shkruajmë funksione të shkurtëra që mund të përmbajnë një pjesë të vogël të logjikës mirëpo këto mund të kompleksohen duke i përdorur sikur parametra në funksione të tjera. Disa nga HOC funksionet të ndërtuara në javascript janë: filter, map, forEach, reduce etj. Këto funksione mund të krijojnë një rezultat të njëjtë edhe nëse përdoren funksionet primitive të javascripti-ti mirëpo esenca e tyre është të thjështësohet mënyra e të shkruarit kod dhe të krijohen algoritme më pak komplekse. Në bazë të kësaj arrijmë të ndërtojmë një logjikë të HOC komponentëve në React e që në vete kanë këtë strukturë

- HOC është një komponentë

- HOC pranon një komponentë tjetër si argument
- HOC kthen një komponentë të re
- Komponenta qe kthehet mund të jetë komponenta e pranuar si argument pa dhe me modifikime [11].

Ne figurën 4 tregohet se si një komponentë (Wrapped component) dërgohet si prop në një komponentë tjeter HOC (High Order Component) dhe më pastaj bëhet kthimi i saj pa modifikuar komponentën përkatëse. Rasti në fjalë paraqet formën më të thjeshtë të paternit.

```
import React from 'react';
const higherOrderComponent = (WrappedComponent) => {
    return <WrappedComponent />;
    }
};
```

Figure 4 High Order Component

Rastet më komplekse të paternit, përveq se komponenta pranon si prop një komponentë tjetër, pranon edhe props të vet komponentes që e pranon e poashtu edhe props të vetëvetes. Në rastin e mëposhtëm në figurën 5 do të trajtojmë një HOC të quajtur WithColor që pret një komponentë dhe një ngjyrë, pos kësaj pret edhe të gjithë props që i kalohen komponetës femijë. Komponenta WithColor është komponentë gjenerike e cila pranon çfarëdo prop që i kalohet pavarësisht tipit që ka. Prop-i i cili kalohet mund të jetë i tipit string, react.ComponentType, ReactNode, number, Array apo çkado tjetër. Në rastin tonë kjo është e nevojshme sepse nuk dihet saktë se çfarë props do ti ketë komponenta fëmijë e cila është në gjendje që ti trajtojë vetë propërtitë e veta.

```
import React from 'react';

interface Props {
    color: string;
}

export default function WithColor<P>(WrappedComponent: React.ComponentType<P>) {

    return (props: P & Props) => {
        return <div style={{color: props.color}}>
            <WrappedComponent {...props} />
        </div>
    };
}
```

Figura 5 WithColor HOC komponenta

Por si do të jetë e shfrytëzueshme HOC komponenta WithColor në komponentet e tjera? Marrim shembull një komponentë - Component A sikur në figurën 6. Komponenta A paraqitet e eksportuar brenda komponentës WithColor. Në raste të rëndomta një komponentë eksportohet duke mos qenë e varur nga një komponentë tjetër si në këtë rast. Arsyeja kryesore pse komponenta eksportohet brenda WithColor është të tregohet se WithColor është HOC e Komponentës A.

```
import React from 'react';
import WithColor from "../HOC/HOC";

interface Props {
    fontSize?: number;
    children: React.ReactNode

}

const ComponentA = ({fontSize, children}: Props) => {
    return <h1 style={{fontSize: `${fontSize}px`}}>{children}</h1>
}

export default WithColor(ComponentA)
```

Figura 6 Ndërtimi i komponentës A

Nëse simulojmë rezultatin në editor pas importimit të komponentës A sikur në figurën e mëposhtme ne fitojmë këtë rezutat:

```
const Home = () => {
    return <>
        <ComponentA color={'red'} fontSize={30}>Component A</ComponentA>
        <ComponentA color={'pink'} fontSize={30}>Component A</ComponentA>
        <ComponentA color={'blue'} fontSize={30}>Component A</ComponentA>
    </>
}
```

Component A

Component A

Component A

Figura 7 Rezultati i fituar pas implementimit të komponentës A

HOC paraqet një ndër paternat më modern në React dhe shumë të përdorshëm edhe tek teknologjia e mikrofrontendëve. Mikrofrontend teknologjia është mjaft e përdorur sepse shumë ekipe mund të punojnë së bashku në një projekt dhe të ndertojnë komponenta të

veçantë pa afektuar komponentet tjtera. Në këtë rast shumë komponenta mund të eksportohen nga moduli në modul dhe mund të jenë të mbeshtjellura brenda paternit HOC si më sipër [11].

4.2 Provider Pattern

Provider pattern është një paternë e cila shërben për të shpërndarë të dhënat në mes komponentëve të ndryshëm të Reactit pa marrë parasysh vendosjen e komponentës në DOM. Provider Pattern përfshinë një komponentë kryesore të quajtur Provider. Komponenta Provider në aplikacion qëndron në instancën më të lartë të komponentëve në radhën hierarkike. Kjo komponentë përmbanë states që bëhen të qasshme nga tërë komponentat tjera në aspektin e leximit dhe modifikimit të tyre. Në React-in klasik, për të dërguar një prop nga një komponentë A tek një tjetër Z e vendosur në shtresat e fundme të DOM-it, ne duhet të përcjellim atë nga Komponenta A në komponentën B më pastaj në C e kështu me radhë deri të arrijë tek komponenta e fundit Z. Sa herë ndodhë ndonjë ndyshim në komponentën A do të renderohen tërë komponentat me rradhë deri te komponenta Z. Kjo shkakon një efekt jo të dëshiruar në React që njihet me termin props drilling. Props drilling paraqet një algoritëm të stërngarkuar dhe për të eliminuar atë është ndjerë nevoja e krijimit të një paterni i cili do të trajtojë numrin e renderimeve në DOM. Një build in provider pattern është React Context. React Context është global state management që krijon mundësi leximi dhe modifikimi të states brenda çdo komponente. Përpos React Context-it, i ndërtuar në logjiken e Provideri-t është edhe React-Redux-i i cili luan rolin e njejtë si të kontekstit por është një librari eksternale. Ne figurën 8 është paraqitur React Reduxi si Provider.

```
const Main = () => {
    return <>
        <Provider store={store}>
            <BrowserRouter>
                <Routing/>
            </BrowserRouter>
        </Provider>
    </>
}
```

Figura 8 React Redux Provider

Në rastin e mesipërm Redux store bëhet i qasshëm nga cilado komponentë e aplikacionit. Një aplikacion mund të përmbajë disa providerë, secili nga ta duhet të vendoset mbi komponentën kryesore e cila në rastin e mesipërm është komponenta <Routing/>. Më sipër paraqitet edhe një Provider tjetër i cili është <BrowserRouter> e librarisë React Router Dom e cila mundëson rrugëzimin e aplikacionit. Rrugëzimi i aplikcionit mundëson krijimin e routave nëpër të cilat do të navigojë përdoruesi. Renditja e Providerëve nuk ka rëndësi përderisa përmabajnë në vete funksione të ndryshme dhe qellime te ndryshme [12].

4.3 Paterna e renderimit të kushtëzuar

Renderimi i elmenteve në bazë të një kushti të caktuar është një paternë e zaknonshme që hasim gjatë zhvillimit në React. Në React ekzistojnë disa paterna të renderimit të listuara si më poshtë:

- If / else
- Ternary operator
- If me operatorin logik &&
- Switch operatori
- Renderimi i kushtëzuar duke përfshirë enums
- Kompozicioni [13].

Metoda if / else

If / else në React paraqet një kushtëzim sikur në Javacriptct por që përdoret për renderime të DOM -it. Në figurën e meposhtme paraqitet një metodë e renderimit nëse state-i isLoggedIn është i saktë renderohet <h1>Is logged In </h1> përndryshe do të renderohet Is logged out ashtu si në figuren 9.

```
const IfElseComponent = (isLoggedIn: boolean) => {
    if (isLoggedIn) {
        return <h1>Is logged In</h1>
    } else {
        return <h1>Is logged out</h1>
    }
}
```

Figura 9 Renderimi if/else

Ternary operator

Ternary operator është shkuresa e if else e cila pranon tri operande. Sintaksa e saj është si më poshtë:

Kushti ? “E sakta” : “E pa sakta”

Kur kushti është i saktë ekzekutohet “E sakta” kur kushti është i pasaktë ekzekutohet e pa sakta [13].

Në shembullin figurës 10 mund të implementohet si vijon:

```
return <div>
    IsLoggedIn ? <h1> Is logged In </h1> : <h1>Is logged out </h1>
</div>
```

Figura 10 Ternary operator

If me operatorin logjik &&

&& është një operatorë bulean i cili në esencë nënkupton “dhë”-në. Që kushti të rezultojë i saktë duhet që ana e djathtë të jetë e saktë. Shembulli i mësipërm mund të implementohet duke përdorur këtë paternë si vijon:

```
return <div>
    IsLoggedIn && <h1> Is logged In </h1>
</div>
```

Figura 11 Renderimi me operatorin logjik &&

Nëse IsLoggedIn është e pa saktë do të kthehet vlera undefined, nëse IsLoggedIn është e saktë do të kthehet vlera <h1>Is logged in </h1>[13].

Switch operatori

Switch operatori në React është i njëjtë me operatorin switch në Javascript. Metoda switch pranon një variabël në bazë te së cilës ndodh renderimi. Variabla në fjalë mund të ketë disa cases dhe cila nga to e plotëson kushtin kthen vlerën që gjendet në return. Në figurën 12 paraqitet rasti i figurës 9 por duke përdorur operation switch.

```
const RenderSwitch = (isLoggedIn: boolean) => {
    switch (isLoggedIn) {
        case true:
            return <h1>Is logged In</h1>
        case false:
            return <h1>Is logged out</h1>
    }
}
```

Figura 12 Switch operatori

Nëse asnjë nga variablat nuk është e saktë por që në rastin tonë nuk është e mundur pasi që vlera hyrëse është buleane, atëherë shtohet edhe kushti default i cili ekzekutohet nëse asnjë nga rastet nuk është i saktë [13].

Renderimi i kushtëzuar duke përfshirë objektet

Një objekt paraqet një koleksion të propertive ku një properti përbehet nga një emër i njohur si key dhe nga një vlerë. Në shembullin e mëposhtem do të trajtohet një rast me njoftime ku varësisht se çfarë key kalohet në komponentë do të renderohet vlera e atij key. Në figurën 13 paraqitet objekti i emërtuar Notifications.

```
const NOTIFICATIONS = {
    info: <Info/>,
    warning: <Warning/>,
    error: <Error/>,
}
```

Figura 13 Objekti NOTIFICATIONS

Objekti notifications përmbanë tri vlera: info që përmbanë komponentën Info, warning që përmbanë komponentën Warning dhe error që përmanë komponentën Error. Përmbajtja e komponentave në fjalë paraqitet në figurën 14.

```
const Info = () => {
    return <h1>Info component</h1>;
}
const Warning = () => {
    return <h1>Warning component</h1>;
}
const Error = () => {
    return <h1>Error component</h1>
}
```

Figura 14 Përmbajtja e komponentëve Info, Warning, Error

Në vazhdim, në figurën 15 do të implementojmë komponentën Notification e cila në vartësi të stringut që i kalohet si prop do të renderojë komponentën që është vlerë e atij stringu në enumin NOTIFICATIONS

Figura 15 Komponenta Notification

Implementimi i kësaj komponente bëhet si më poshtë:

<Notification state=”info”/> në menyrë që të renderohet komponenta <Info/>

<Notification state=”warning në menyrë që të renderohet komponenta <Warning/>

<Notification state=”error”/> në menyrë që të renderohet komponenta <Error/>

Shprehia e përdorimit të objekteve paraqet një strukturë majft të qartë të kodit [14].

4.4 React Hooks

Një tjetër paternë e dizajnit është edhe Hooks pattern e cila u implementua në versionin 16.8 të Reactit dhe ka revulucionizuar mënyrën se si ne ndërtojmë komponentat në React. Para se të prezentohej versioni në fjalë i Reactit, mënyra e të shkruarit dhe logjikës së tij ishte e bazuar në class Componenta, këto komponenta kishin disa metoda të tyre që shënonin se kur duhet të renderohet, përditesohet dhe fshihet komponenta. Këto metoda ishin: componentDidMount(), componentDidUpdate() dhe componentDidUnmount(). Me implementimin e FC (functional components) , class komponentat u eliminuan nga përdorimi dhe filloj të implementohej logjjika e hooks. Hooks në realitet janë funksione të Javascriptit që shërbejnë për të izoluar pjesën e ripërdorshme nga komponenta[15]. Në vartësi të përdorimit, hooks mund të kenë natyra dhe funksione të ndryshme. Disa prej tyre shërbejnë për të memorizuar komponentet, për të memorizuar funksionet, për të bërë përditesimin e state-s, për të detektuar rirenderimet, për të larguar elementet nga DOM-i etj. Më poshtë gjeni disa prej hooks më të përdorshëm në React:

- useState: një gjendje në komponentë deklarohet si state. State mund të përditesohet duke përdorur hook-un useState dhe funksionin përkatës useState të cilit i kalohet si parametër një vlerë apo një funksion.
- useEffect: ekzekutohet në tri raste: kur DOM-i renderohet në herën e parë, sa herë që ndonjë state pëson ndryshime dhe sa herë që deshirojmë ta largojmë një element nga DOM-i. UseEffect përmbanë në vete një varg të gjendjeve të njohura ndryshe edhe si depedencies të cilat bejnë që useEffect të ekzekutohet sa herë që ndryshon njëra nga to. Use Effect përmban edhe funksionin fshirës i njohur si cleanUp që ekzekuton kodin e implementuar në atë funksion së herë që zhduket komponenta.
- useRef: Në javascript për t'iu qasur një elementi në duhet të targetojmë id-në, klasën apo edhe vetë elementin. Në React për të targetuar ndonjë DOM element në krijomë referencë ndaj atij elementi dhe sa herë që na nevoitet ai element për të krijuar ndonjë fokus event apo për çfarëdo qëllimi tjetër ne e përdorim referencën e tij. UseRef përdoret më së shpeshti me propertinë current.
- useMemo: shërben për të memorizuar të dhënat e një funksioni në mes të dy rirenderimeve. Nëse rezultati i funksionit është i njëjti si ne renderimin e kaluar ateherë komponenta nuk renderohet dhe kështu bën që të kursehet koha e renderimit në website. Një derivat i useMemo është edhe hook-u Memo i cili nuk memorizon rezultatet e funksioneve por memorizon tërë komponentën dhe props të saj, në menyrë analoge sikur me argumentet e funksionit veprohet edhe me props të komponentës.
- useReducer: është hook shumë i ngjajshëm me useState. Në vend se të bëhet përditesimi i menjëhershëm, brenda useReducer pranojmë actions të cilat janë funksione që tregojnë se si duhet të bëhet update state-i. Kur përdoret useReducer paraprakisht duhet të caktohet emri i action në menyrë që të ndodh dispatch i atij action.
- UseLayoutEffect: Është hook i ngjajsëm me useEffect, të dy e kan qëllimin e njëjtë mirëpo dallon koha e ekzekutimit mes tyre. UseLayoutEffect ekzekutohet pasi që të bëhet renderimimi mirëpo para se të bëhet përditesimi i Domi-t, për dallim nga useEffect që ekzekutohet pasi të renderohet DOM dhe pasi të përditesohet [15].

Hook-s që u prezentuan më sipër janë hooks të implementuar nga vet React-i, mirëpo shpesh ndjehet nevoja e përdorimit të hooks për të ndërtuar një logjike të kostumizuar në komponentet tona. Prej paraqitjes së hooks në React po bëhet edhe adaptimi i të ashtëquajturave Custom hooks që janë të ngjajshme për nga sintaksa me hook-s të Reactit. Custom Hook-s fillojnë me fjalën use e që paraqet një fjalë të rezervuar në React [16]. Një shembull i përdorimit të një hook-u të veçantë është shembulli i implementimit të kërkesave në backend. Një hook i tillë do të pranonte një url në të cilën do të procesohej kërkesa. Të dhënat që mund të dergohen në body të kërkeses apo edhe të dhënat që mund të dergohen në parametra si query parametra. Me rastin e pranimit të këtyre parametrave hook-u do të jetë në gjendje të detektojë përgjigjjen e kërkesës dhe statusin e saj. Nëse statusi do të ishte 200 ateherë do të ktheheshin të dhënat, nëse statusi do të ishte 400 do të kthehej një mesazh refuzimi e kështu me radhë. Në figurën 16 paraqitet ndërtimi i një custom hook që u spjegua më sipër [16]:

```
import { useState, useEffect } from "react";

const useFetch = (url) => {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetch(url)
      .then((res) => res.json())
      .then((data) => setData(data));
  }, [url]);

  return [data];
};

export default useFetch;
```

Figura 16 Custom Hook në React

Logjika brenda custom hooks mund të përdoret kudo në aplikacion [16].

4.5 Komponentat e përbëra

Komponentat e përbera janë një set i komponentave që i përkasin njëra tjetres dhe funksionojnë shumë mirë së bashku. Komponentat e përbera nuk janë të njohura vetëm në React, në realitet ato ekzistojnë edhe në HTML p.sh html tagu <select> i cili në vete përmbanë si komponentë femijë tagun <option>[17]. Në shembullin më poshtë do të paraqitet ndërtimi i Tabs i cili përfshinë Tab Items. Në figurë është paraqitur

komponenta Prind e cila në vete përmbanë si femijë Tab Items. Tabs pranon një interface i cili përmbanë color dhe children por përpos kësaj pranon edhe Items. Items në këtë rast paraqesin një komponentë të përberë e cila vjen si femije i Tabs dhe kjo deklarohet në rreshtin Tabs.Items = Items

```
interface Props {
    no usages
    color: string;
    no usages
    children: ReactNode;
}

new *
const Tabs = ({color, children}: Props) => {
    return <>
        <ul style={{color: color}}>{children}</ul>
    </>
}
new *
const Items = ({children}: { children: ReactNode }) => {
    return <li>{children}</li>
}

Tabs.Items = Items

export default Tabs;
```

Figure 17 Komponentet e përbëra

Implementimi i komponentës së mësipërme paraqitet në figurën 17. Për çdo iterim në map paraqitet një compound component që quhet <Tabs.Items/>. Përvec Tabs.Item mund të kalohet çdo properti tjetër si në rastin e mëposhtëm që është butoni dhe ngjyra.

```
const tabs = ['tab1', 'tab2', 'tab3'];
return <Tabs color={'red'}>
    <button>Click me!</button>
    {tabs.map((tab :string ,key :number ) => <Tabs.Items key={key}>{tab}</Tabs.Items>)}
</Tabs>
```

Figure 18 Implementimi i komponentës së përbërë

Kjo patternë është mjaftë e rëndësishme sepse krijon kategorizimin e nënkomponentave. Shumë UI paketa e përdorin për ndërtimin e menuve, tabs, cards etj[17]

4.6 Kompozicioni

Kompozicioni është paterna që përdoret më së shpeshti në React. Ne shpesh krijojmë komponenta që pranojnë si prop një femijë që nuk modifikohet dhe shfaqet ashtu siç vjen[18]. Shembull konkret kemi buttonin i cili pranon një text si më poshtë

<Button>{text}</Button>.

Në këtë rast text mund të zëvendesohet me një fjalë të rezervuar që njihet me emrin {children}. Nësë React-i përdoret me typescript që është një superset i javacript-it e që mundëson që gjatë krijimit të komponenteve të caktohen tipet e props, do të shohim se nëse ne e deklarojmë një komponentë si FC ose functional component, {children} do të jetë automatikisht e njohur. Nëse në vend të text përdorim children ateherë komponenta do të duket keshtu:

<Button>{children}</Button>

Kompozicioni shërben kryesisht për ndërtimin e strukturës së faqeve[18].

4.7 Container/Presentational Pattern

Në React, një menyrë për të zbatuar ndarjen e problemeve është përdorimi i Container/Presentational Paternit. Me këtë model ne mund të ndajmë pamjen nga logjika e aplikacionit. Le të themi se dëshirojmë të krijojmë një aplikacion që do të bëjë një kërkesë në back-end për të shfaqur 6 lule në ekran. Për të aplikuar këtë patern ne duhet të ndajmë këtë proces në dy pjesë:

- Ne pjesën e prezentimit: Komponentet që kujdesen se si të dhënat shfaqen tek përdoruesi, e që përfshinë listën me imazhe të luleve
- Ne pjesën e konteinerit: Komponentet që kujdesen së qfarë të dhëna do të shfaqen tek përdoruesi që në rastin tonë përfshinë cilat imazhe do të shfaqen[19].

Komponenta prezentuese

Një komponentë prezentuese pranon të dhënat përmes Props. Funksioni primar i saj është thjështë të shfaqë të dhënat që i pranon në menyrën e duhur duke përfshire vetëm stilizimin dhe jo modifikimin e të dhënave. Komponentat prezentuese zakonisht janë

stateless: ato nuk përmbajnë states të Reactit dhe të dhënat që i pranojnë nuk ndryshohen nga brenda komponentes [19].

Komponenta përmbajtese - konteineri

Roli kryesor i komponentës përmbajtëse është që ti kalojë të dhënat tek komponenta prezentuese. Komponenta përmbajtëse nuk renderon ndonjë komponentë tjetër përpos komponentave prezentuese. Për këtë aryje nuk përfshinë ndonjë stilizim internal të tij [19].

Një shembull konkret që përfshihet tek projekti Rate Teacher të cilit do t'iu referohemi në fund të faqës është struktura organizuese e atomikut. Në figurat e mëposhtme gjeni Templates që në këtë rast luajnë rolin e Komponentes prezentuese, dhe komponentes përmbajtëse e që në këtë rast janë Pages të cilët krijojnë kërkesa në backend dhe marrin të dhënat. Në rastin tonë në page përfshihen vetëm fajllat me tsx dhe ts sepse stili i lihet komponentës përmabjtëse si në figurën 19.

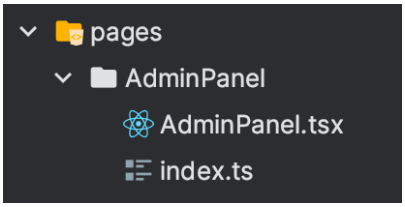


Figura 19 Komponenta konteiner

Ndersa Ne figurën 20 paraqiten Templates që përfshijnë edhe pjesën e stilizimit në supersetin e CSS të njohur si SCSS.

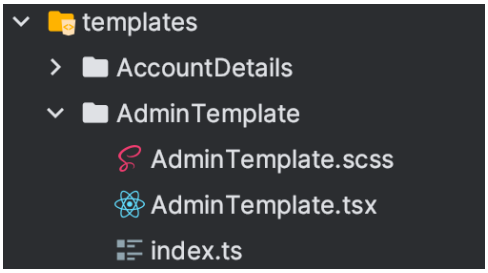


Figura 20 Komponenta prezentuese

5. Krahasimi i paternave të dizajnit

Paternat e dizajnit përdoren për të thjeshtësuar aplikacionin por jo vetëm. Paternat ndikojnë në zhvillim produktiv të tërë aplikacionit dhe ekipës së zhvillimit sepse arrihet mirëmbajtja e kodit në mënyrë sa më të strukturuar të mundshme. Në bazë të analizave të komunitetit të React-it nuk është ndonjë paternë që vlen më shumë se tjetra. Paternat marin fuçi krahasuese në momentin që ato fillojnë të implementohen. Nuk mund të thuhet se një paternë krijon një logjikë më të mirë se tjetra sepse përdorimi i tyre duhet të jetë i drejtë për secilin rast të mundshëm. Varësisht nga problemi që parashtrohet bëhet përafrimi i problemit më atë patern. E rëndësishme është që paterni të jetë sa më i kuptueshëm dhe të jetë sa më gjeneral. Siç e dimë arsyeja primare e krijimit të paternave është krijimi i kodit të ripërdorshëm, nësë paterni krijon kod të ripërdorshëm atëherë mund të themi se i plotëson nevojat e komponentës dhe duhet të implementohet. Por nëse paterni krijon një kaos e që shkaktohet si pasojë e mos implementimit të drejtë të tij ateherë duhet të eliminiohet për atë rast. Pra, më një fjalë, çdo paternë është e rëndësishme dhe asnjëra nga to nuk duhet të neglizhohet [20].

6. Krijimi i një aplikacionit duke përdorur restful api në React JS

Aplikacioni i realizuar në vijim i titullurar Rate Teacher është një aplikacion që përfshinë React js si teknologji në frontend dhe nest js në backend dhe për ruajten e të dhënave përdoret Mongo DB. Aplikacioni mundëson vlerësimin e performancës së profesorëve duke u bazuar në disa pyetje. Përgjigjjet e këtye pyetjeve pasqyrojnë se sa është i votuar një profesor në të gjitha shkallët e votimit (pa mjaftueshëm, mjaftueshëm, mirë, shumë mirë, shkëlqyeshëm) dhe përcaktojnë kualitetin e mësimdhënjes së tij. Aplikacioni përfshinë dy module kryesore të cilat janë:

- Moduli i përdoruesit: përdoruesi logohet në aplikacion dhe pas logimit kalon tek faqja kryesore, përmes faqes kryesore përdoruesi mund të kërkoj për ndonjë profesor dhe t‘iu qaset të dhënave të tij.
- Moduli i administratorit: përfshinë një apo disa administatorë të cilët kanë qasje në databazën e aplikacionit përmes një paneli të kontrollit të zhvilluar brenda aplikacionit, përmes këtij paneli mundësohet editimi, shtimi dhe fshierja e të gjitha entiteteve të aplikacionit.

6.1 Databaza

Databaza e aplikacionit është e ndërtuar me Mongo DB. Mondo DB është një databazë jo relacionale e cila ofron fleksibilitet në qasje të të dhënave. Ndryshe nga databazat relacionale të ndërtuara me tabela, mongo DB ndërtohet mbi formatin e të dhënave JSON me të cilin krijohen skemat [20]. Në aplikacion gjenden pëse skema të paraqitura edhe në figurën 21

- Departments: shërben për ruajtjen e departamenteve ose lëndeve mësimore
- Professor: shërben për ruajtjen e profesorëve
- Schools: shërben për ruajtjen e institucioneve shkollore
- User: shërben për ruajtjen e përdoruesve
- UserRoles: shërben për ruajtjen e roleve të përdoruesit (administratorë ose përdorues i thjështë)

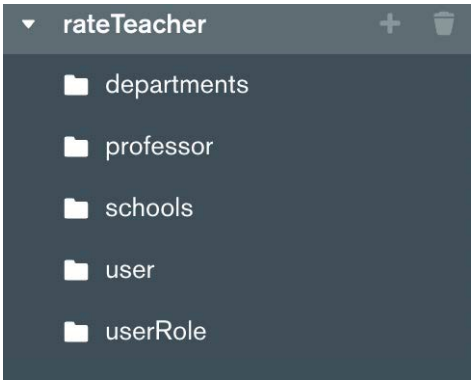


Figura 21 Skemat në aplikacion

Në figurën e mëposhtme shihet një shembull se qfarë përmbanë skema profesorët dhe merret shembull një profesor i cili përmbanë një Id të gjeneruar nga mongoDb, emrin, shkollën, Id-në e shkollës, departamenetet dhe vlerësimet.

Figura 22 Instanca e një profesori në databazë

Të gjitha instancat e lartë cekura kanë nga një skemë përkatese.

6.2 Organizimi i strukturës së aplikacionit në front end dhe back end

Strukturimi i fajllave ndikon në përmirësimin e rrjëdhes së aplikacionit dhe logjikës. Në boten e progamimit, zhvilluesit iu përshtaten paternave të ndryshëm të organizimit. Në platformën e mësipërme sa i përket pjesës së front-end-it është përdorur atomic structuring patten. Në figurën 23 paraqitet struktura pesë shtresore e cila përfshinë atomet si njësi më të vogla të ripërdorimit e që në rastin konkret mund të jetë një buton, një input, një imazh etj. Tërësia e dytë përfshinë molekulat të cilat janë kombinimi i dy apo më tepër atomeve. Tërësia e tretë paraqet organizmin i cili përfshinë fillimin e logjikës së aplikacionit siç jane iterimet dhe kushtet. Tërësia e katërt paraqet templates e që në vete përbejnë një ndër pjesët më të rëndësishme sepse aty caktohet pamja apo

vendosja e elementeve në faqe dhe struktura e pestë paraqet faqet. Në faqe krijohen kerkesat për back-end dhe pas kthimit te përgjigjjeve mbushen templates me të dhëna nga back-end-i. Atomic struktura po gjen zbatim të madh për arsyje të zbërthimit të detajshëm dhe kompleksitetit të ulët.

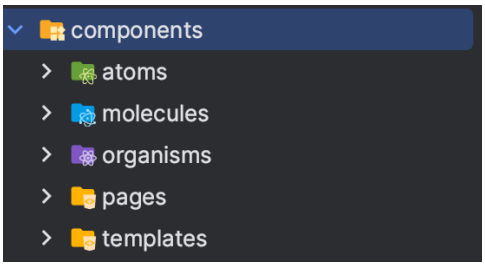


Figure 23 Atomic pattern

Përpos strukturës së mësipërme, përfshihen edhe enums, funksionet, rutimi, stilizimi i komponentave që bëjnë pjesë në organizim të fajllave.

Sa i përket pjesës së backendit, gjenerimi i resurseve në nest js është bërë me komandën nest-g-resources ku është krijuar një follder për secilin entitet. Në figurën 24 paraqitet follderi i shkollave i cili në vete përfshinë modelin kontrollerin dhe serviset. Në kontroller tregohet HTTP metoda në të cilën duhet të dërgohet requesti e që mund të jetë GET, POST, DELETE, PUT, PATCH , si dhe prarametrat që mund të dërgohen në body të kërkesës apo edhe si query string. Ndërsa pjesa e servisit paraqet implementimin dhe zgjedhjen e problemit në bazë të inputit nga kontrolleri.

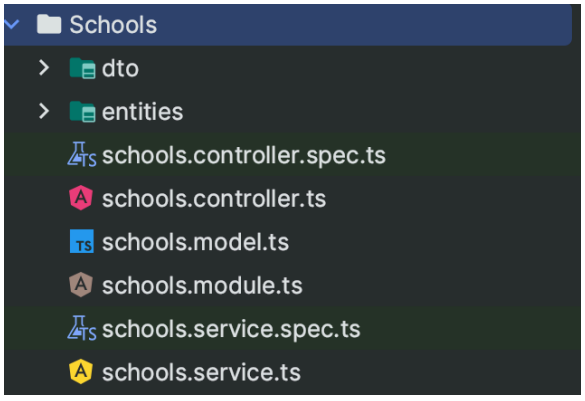


Figure 24 Organizimi i struktures në backend

Krijimi i front end dhe back end është bërë në folldera të ndryshem në mënyre që menaxhimi i punës të jetë më i lehtë.

6.3 Përdorimi i RESTful api

Përdorimi i restful api në aplikacion është kudo. Nëse e analizojmë më poshtë pjesën e kodit të shënuar shohim një servis asinkron të krijuar duke përdorur formën try catch. Në javascript serviset janë kryesisht asinkrone sepse ekzekutohen atëherë kur ktheht përgjigjja nga serveri, mirëpo aplikacioni nuk pret për ekzekutimin e tyre sepse vazhdon rrjedhën e të ashtëquajturës "event loop" dhe në momentin që kërkesa e backendit ekzekutohet ndodh edhe vendosja e rezultateve në DOM. Më poshtë shohim se funksioni request në vete përmbanë dy parametra:

- HTTP metodën e cila në këtë rast është GET
- URL e resurseve që në rastin tonë është "professors"

Rasti i mëposhtëm është rasti më i thjeshtë i zbatimit të rest arkitekturës sepse nuk ka as parametra ne body as si query string.

```
export const getProfessors = async () => {
    try {
        return await request(RequestMethods.GET, url: 'professors')
    } catch (e) {
        console.log(e)
    }
}
```

Figure 25 Servisi getProffesors

Nëse e marim një shembull më kompleks të një servisi më të komplikuar si në figurën më poshtë shohim se si në servisin modifyProfessors kërkohet që në backend të dergohen edhe parametra duke përdorur string interpolation dhe të dërgohen te dhëna si data në body.

```
export const modifyProfessors = async (params: any, data: any) => {
    console.log('hi')
    try {
        return await request(RequestMethods.PUT, url: `professors/${params}`, data);
    } catch (e) {
        console.log(e)
    }
```

Figura 26 Servisi modifyProfessors

Më poshtë gjeni listën e të gjitha serviseve që zbatojnë rest arkitekturën në platformë.

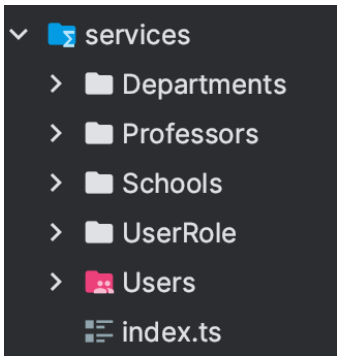


Figure 27 Serviset ne aplikacion

6.4 Përshkrimi i aplikacionit

Me startimin e aplikacionit përdoruesi njoftohet me kryefaqen ku vëmendjen e mer pjesa e shiritit të kërkimit. Në shiritin e kërkimit duhet të shkruhet emri i profesorit ashtu sikur në figurën 28 p.sh Filan Fisteku. Gjatë kërkimit hapet menu-ja rënëse me profesorët përkates ku me klikimin mbi emrin ose shkollën aplikacioni ridirektohet tek të dhënat e profesorit.

Ndër të tjera në kryefaqe përfshihet edhe pjesa informuese me titullin “Rreth platformës Rate Teacher” që tregon detaje rreth platformës dhe pjesa e statistikave që përfshinë numrin e përdoruesve, shkollave dhe profesorëve aktiv në platformë si ne figurën 29.

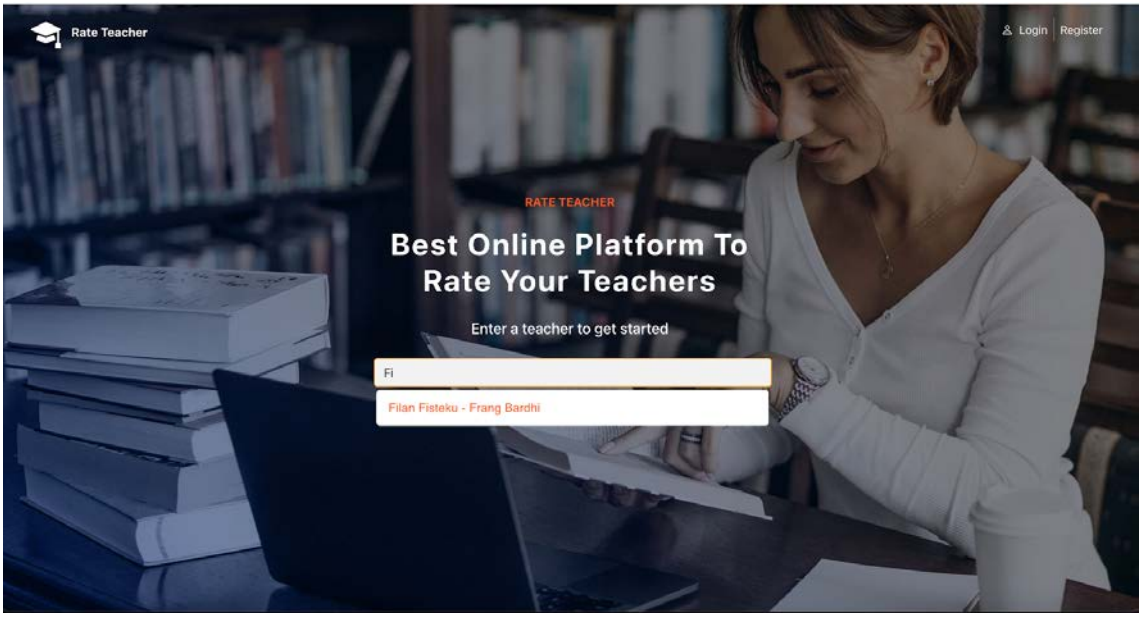


Figura 28 Kryefaqja e aplikacionit

Figura 29 Statistikat online

Pas klikimit në profesorin përkatës hapet një dritare tjetër si në figurën 30. Në anën e djathtë të faqes paraqiten pesë shirita te progesit, shkallet e tyre janë si më poshtë: terrible, bad, good, great, awesome dhe në anën e djathë të tyre janë numri i votave që rritet proporcionalisht me ngjyrën portokalli. Në anën e majtë me anë të algoritmit të mesatares aritmetike është treguar kualiteti në bazë të votave e që në rastin e mëposhtëm është 3.33/5 ndërsa në pyetjen se a do të dëshironin ta kenë prap si ligjerues përgjigjja është 41.47% për po. Poshtë statistikave paraqiten edhe top komentet të shkruajtura nga studentët dhe numri i tyre limitohet në 10 për faqe në menyre që të jenë më lehtë të menaxhueshëm.

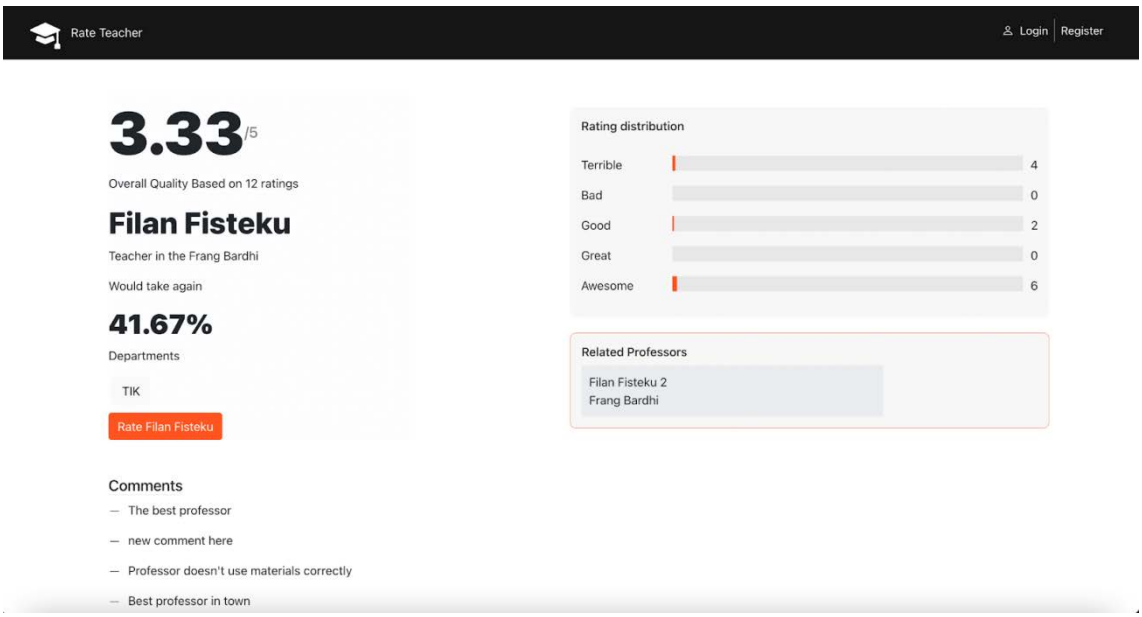


Figura 30 Faqja e detajeve të profesorit

Në anën e poshtme përkatesisht tek related professors paraqiten profesorët të cilët janë të punësuar në shkollën e njëjtë. Kardat që paraqiten janë të gjitha të klikueshme dhe nëse klikohet në ndonjëren prej tyre, profesori aktual do të zëvendësohet me të klikuarin dhe të dhënat e tij do të bëhen aktive. Poashtu nëse klikohet tek butoni Rate atëherë bëhet navigimi te pyetësori mirëpo janë dy raste të mundshme të navigimit:

1. Kur useri është i loguar
2. Kur useri nuk është i loguar

Në rastin e parë aplikacioni të ridirekton në faqen e login-it në menyre që të shkruani kredencialet e nevojshme si më poshtë

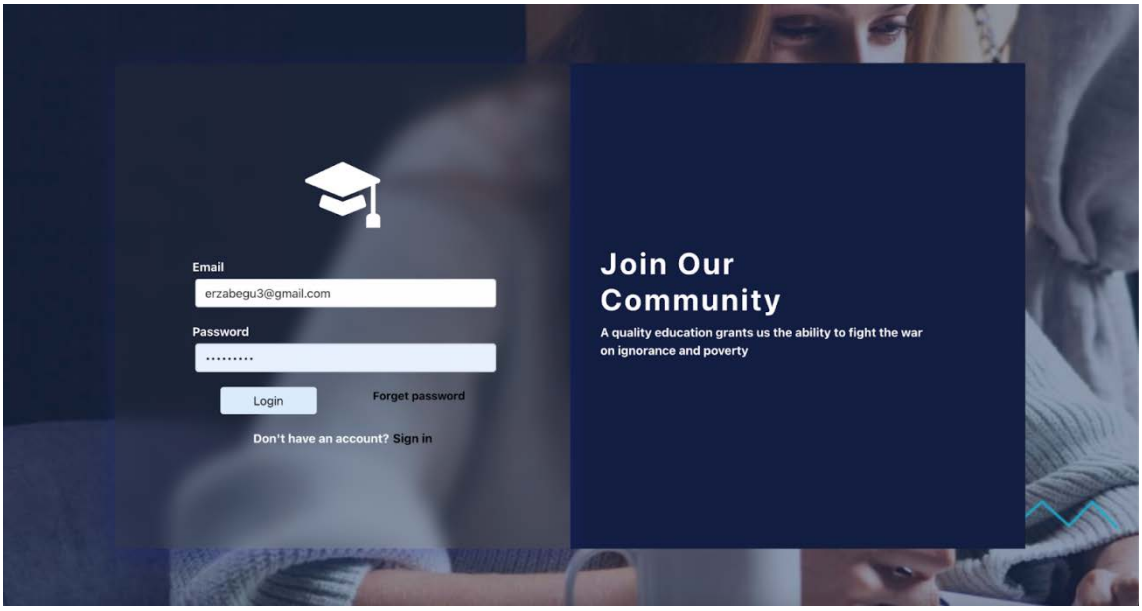
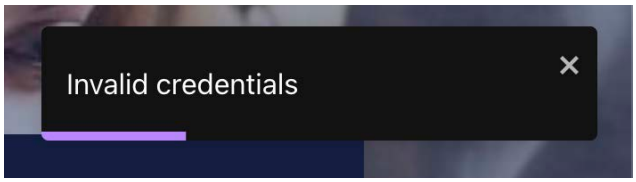


Figura 31 Faqja e loginit

Forma e loginit përmbanë dy inpute: emailin dhe passwordin. Ekzistojnë validime për disa raste si më poshtë:

- Kur emaili nuk është valid
- Kur passwordi është më i shkurtër se 6 karaktere dhe nuk përmbanë shkronjë të madhe e karakter special
- Kur ndonjëra prej fushave është e zbrazët

Në rastet e mësipërme përdoruesit i shfaqen mesazhe paralajmëruese si në figurën 32



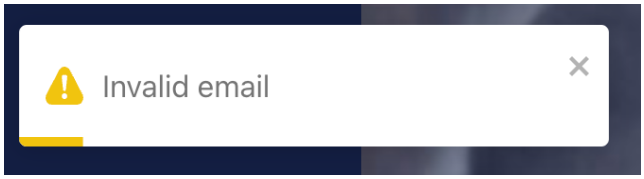


Figura 32 Mesazhet paralajmëruese

Sistemi i loginit përfshinë authentikimin e përdoruesit duke përdorur JWT - Json web tokens që janë një standard i veqantë i authentifikimit dhe për çdo user të loguar shoqërohet nga një token i cili nuk skadon deri në momentin kur përdoruesi vendos të shkyqet nga aplikacioni. Vlen të ceket se nëse përdoruesi ka harruar passwordin, mund të navigoje tek faqja për risetimin e passwordit ku përdoruesi do të pranojë një email i cili mundëson risetimin e passwordit.

Nëse përdoruesi nuk ka llogari, ateherë me anë të hederit të faqes mund të navigojë tek faqja për regjistrim që duket si ne figurën 33.

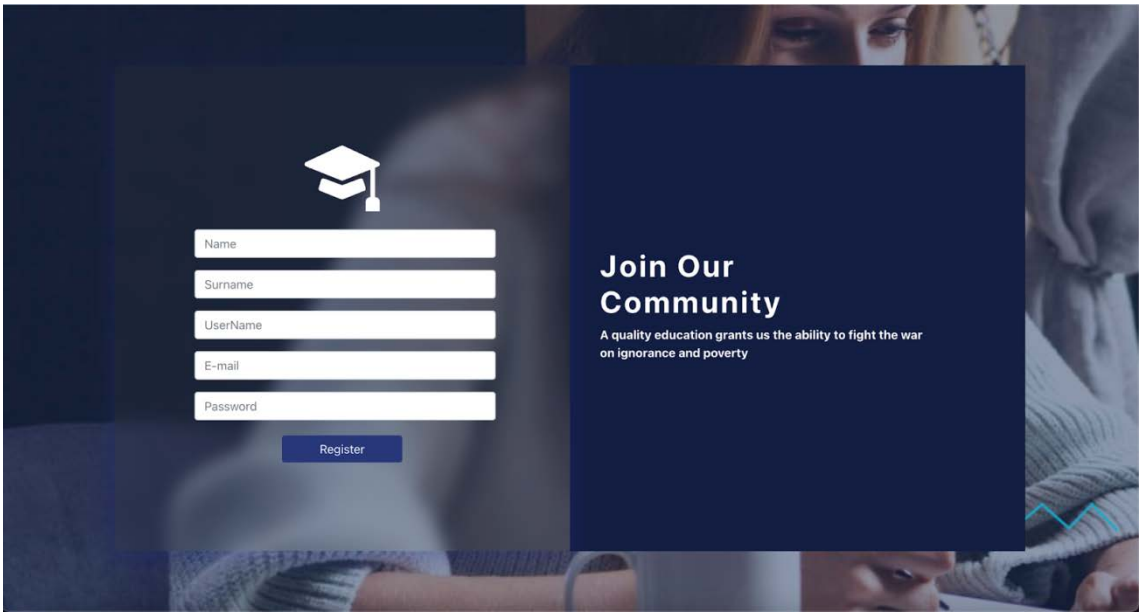


Figura 33 Faqja e regjistrimit të përdoruesve

Përdoruesi duhet të mbushë të gjitha fushat e lartë kërkuara: emri, mbiemri, username-i, emaili, fjalëkalimi ku prej tyre emaili dhe username-i janë fusha unike. Nëse ekziston username-i do të kthehet mesazhi se përdoruesi me username të tillë eksziston, dhe poashtu nëse njëra nga fushat nuk plotesohet do të paraqitet mesazhi paralajmërues për plotësimin e të gjitha fushave.

Rasti i dytë është kur përdoruesi është i loguar paraprakisht dhe klikon në butonin i cili e dergon tek pamja si në figurën 34. Në pamjen që shihet në figurë është një pyetësor që përmbanë 5 pyetje. Dy nga to janë në formë të rankimit me yje, dy të tjera përmbajnë

inpute të tipit select dhe në fund është një fushe e tipit text area. Të gjitha fushat e mëposhtme janë të detyrueshme për plotësim dhe nëse njëra nga to nuk plotesohet, përdoruesi ndalohet nga mesazhet paralajmëruese të dorezojë pyetësorin. Me plotësimin e të gjitha fushave përdoruesi krijon një rankim për profesorin në fjalë dhe procedura do të përseritet nga fillimi.

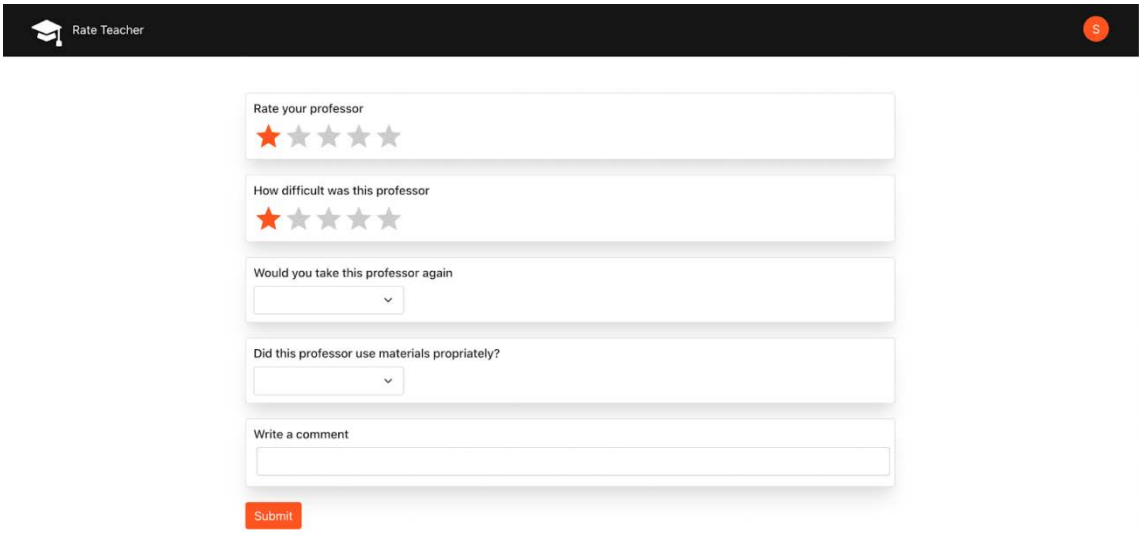


Figure 34 Faqja e rankimit

7. Përfundim

Në temat e trajtuara më sipër ju patët mundësi të njiheni më shumë me komunikimin e pjesës së serverit me ate te klientit apo edhe backendit dhe frontendit. Me shfaqjen e Restful API në botën e internetit po krijohet një metodë më e lehtë për qasjen e të dhënave qoftë në menyrë publike apo private dhe po krijohet edhe rritja e sigurisë dhe performancës. Nëse gjatë kodimit ju përdorni drejtë restful api ju do të keni një kod më të pastër dhe më fleksibil dhe nje aplikacion me te shpejte. Në anën tjetër paternat në React mundësojnë që zhvilluesit të krijojnë një ambient pune dhe një infrastrukturë të qartë në atë se qfarë ata implementojnë. Aplikacioni i zhvilluar në bënë të kuptojmë se si ndodh zhvillimi i një aplikacioni duke integruar Front end dhe Back end teknologjitë. Front end teknologjia e përdorur me sipër len të kuptohet se React do të mbetet libraria e së ardhmes edhe për një kohë shumë të gjatë. Projekti në fjalë paraqet një pasqyrë të qartë të asaj se si implementohen paternat e dizajnit në kod dhe si rritet përfomanca e aplikacionit në përgjithësi. Paternat e dizajnit mbeten të zhvillohen edhe më tutje sepse e ardhmja e reactit bazohet në fuçinë ripërdorimit të komponentëve të tij.

Literatura

[1]. “What Is Web Development (2021 Guide) | BrainStation®,” *BrainStation*. https://brainstation.io/career-guides/what-is-web-development [qasur më 16.09.2022].

[2]. “What is ReactJS: Introduction To React and Its Features,” *Simplilearn.com*. https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs [qasur më 20.09.2022]

[3].“Framework vs Library: Full Comparison,” *InterviewBit*, Oct. 23, 2021. https://www.interviewbit.com/blog/framework-vs-library/ [qasur më 20.09.2022]

[4]. https://mkmahfuz.medium.com/react-introduction-to-react-45b89fda9c07 [qasur më 01.10.2022]

[5]. “Redux Fundamentals: Building Blocks & Data Flow,” celestialblog. https://celestialsys.com/blog/redux-fundamentals-building-blocks-and-data-flow/ [qasur më 01.10.2022]

[6]. Red Hat, “What is an API?,” Redhat.com, 2019. https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces [qasur më 01.12.2022]

[7].mozilla, “An overview of HTTP,” MDN Web Docs, Feb. 19, 2019. https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview [qasur më 05.12.2022]

[8]. “HTTP Methods For Restful Services,” DEV Community. https://dev.to/pramudaliyanage/http-methods-for-restful-services-1515 [qasur më 07.10.2022]

[9]. “What is REST API (RESTful API)?,” SearchAppArchitecture. https://www.techtarget.com/searchapparchitecture/definition/RESTful-API [qasur më 10.12.2022]

[10] L. Eagles, “React component design patterns for 2022,” *LogRocket Blog*, Mar. 02, 2022. https://blog.logrocket.com/react-component-design-patterns-2022/#:~:text=Design%20patterns%20are%20solution%20templates [qasur më 05.10.2022]

[11]. “Higher-Order Components – React,” *reactjs.org*. https://reactjs.org/docs/higher-order-components.html#:~:text=A%20higher%2Dorder%20component%20 [qasur më 07.10.2022]

[12]. “Getting Started | React Redux,” *react-redux.js.org*. https://react-redux.js.org/introduction/getting-started [qasur më 08.12.2022]

[13]. N. Sebhastian, “React conditional rendering with switch statement,” *Nathan Sebhastian*. https://sebhastian.com/react-switch/#:~:text=The%20switch%2Fcase%20statement%20is . [qasur më 08.10.2022]

[14].“Conditional Rendering with Enum | reactpatterns,” *reactpatterns.js.org*. https://reactpatterns.js.org/docs/conditional-rendering-with-enum/ .[qasur më 12.10.2022]

[15].“Introducing Hooks – React,” *Reactjs.org*, 2018. https://reactjs.org/docs/hooks-intro.html [qasur më 15.10.2022]

[16]. R. Wieruch, “React: How to create a Custom Hook,” *www.robinwieruch.de*. https://www.robinwieruch.de/react-custom-hook. [qasur më 15.10.2022]

[17]. “Compound Components In React,” *Smashing Magazine*, Aug. 27, 2021. https://www.smashingmagazine.com/2021/08/compound-components-react/ [qasur më 20.10.2022]

[18].“React Component Composition Explained | Felix Gerschau,” *felixgerschau.com*. https://felixgerschau.com/react-component-composition/ [qasur më 21.10.2022]

[19]. “Container/Presentational Pattern,” *patterns.dev*. https://www.patterns.dev/posts/presentational-container-pattern/ [qasur më 25.10.2022]

[20].UXPin, “The Best React Design Patterns You Should Know About,” *Studio by UXPin*, Dec. 09, 2020. https://www.uxpin.com/studio/blog/react-design-patterns/ [qasur më 26.10.2022]

[21]. MongoDB, “The most popular database for modern apps,” *MongoDB*, 2019. https://www.mongodb.com/ [qasur më 01.11.2022]

Tabela e figurave