Upotreba React-a za razvoj web sajtova uz korišćenje naprednijih kocepata i .NET-a

Student: Nikola Davinić 17588
Profesor: Aleksandar Milosavljević

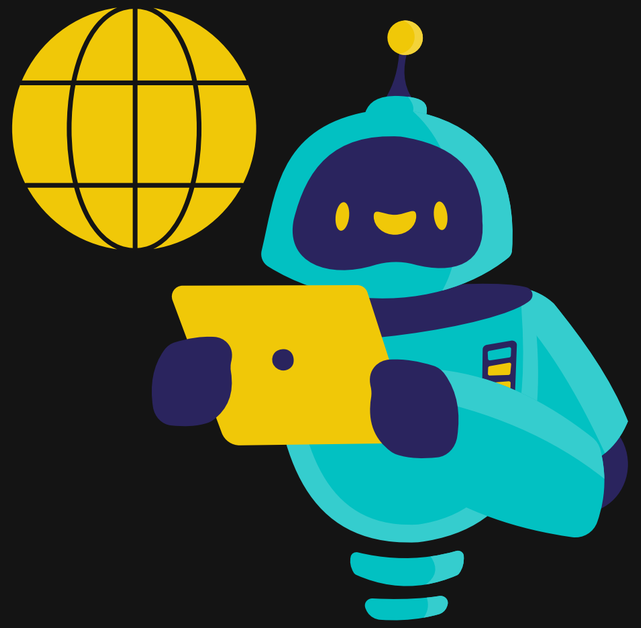
Uvod

React je JavaScript biblioteka i koristi se za izgradnju korisničkih interfesja.

Razvijen je od strane Facebook-a.

Postao je dosta popularan zbog svoje efikasnosti, modulanosti i fleksibilnosti.

Glavni cilj ove prezentacije biće uvod u osnovne koncepte ove tehnologije, kao i nekih malo naprednijih stvari.



Deklarativno programiranje

Deklarativno programiranje predstavlja način gde se opisuje željeni rezultat, a ne korake koje treba preduzeti da bi se to postiglo.

To znači da je fokus na “šta želiš postići”, a ne “kako”.

Kod React-a je ključna stvar kako se upravlja stanjem.

Razlozi zašto se koristi ovakva vrsta programiranja su:

- Jednostavnost razumevanja koda
- Lakše održavanje
- Reaktivnost

Deklarativno vs proceduralno

U poređenju sa sa deklarativnim programiranjem proceduralno programiranje se fokusira na korake koje računar treba da odradi kako bi se došlo do nekog cilja.

Programer eksplicitno definiše korake.

Program se sastoji od funkcija.

Prednosti i mane deklarativnog programiranja

Prednosti:

- Lako razumevanje koda
- Veća apstrakcija
- Ponovna upotreba koda
- Lako održavanje
- Automatska optimizacija

Mane:

- Gubitak kontrole nad implementacijom
- Nedostatak fleksibilnosti
- Teže rešavanje nekih problema
- Veći zahtevi za resursima

Osnovne karakteristike React-a

- Komponente
- JSX
- Jednosmerni tok podataka
- Virtualni DOM
- Stanje
- Reaktivni pristup
- Props
- Lifecycle metode
- React router

Instalacija React-a

Kako bi moglo da se radi sa React-om neophodno je da se najpre instraliraju Node.js i npm (Node Package Manager) sa zvaničnog sajta za Node.js.

Instaliranjem ovog neophodnog alata dobijamo i Vite koji predstavlja brz i miniamalan alat za razvoj React projekta.

Komanda kojom se kreira React projekat uz pomoć Vite je:

```
npm create vite@latest
```

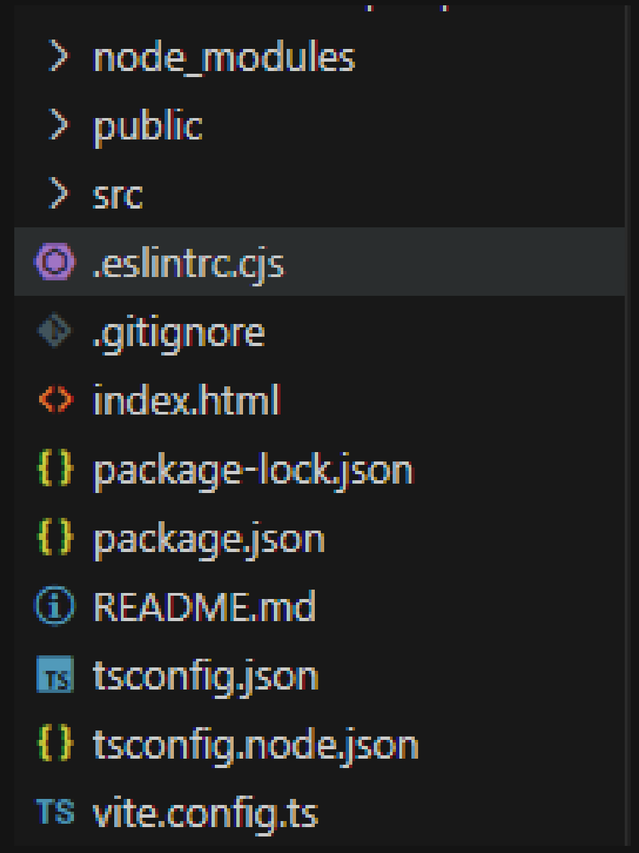
```
√ Project name: ... vite-project
? Select a framework: » - Use arrow-keys. Return to submit.
    Vanilla
    Vue
>   React
    Preact
    Lit
    Svelte
    Solid
    Qwik
    Others
```

```
vite-project>npm run dev
```

Struktura projekta

Svaki React projekat sadrži sledeće početne fajlove i foldere:

- node_modules folder: instalirane zavisnosti
- public folder: javno dostupni fajlovi
- src folder: izvorni kod aplikacije
- .gitignore fajl: šta git treba da ignoriše
- index.html: osnovni HTML dokument za aplikaciju
- package.json: definisane zavisnosti, skripte i konfiguracije
- tsconfig.json: konfiguracija za TypeScript
- vite.config.ts: konfiguracija za Vite
- README.md: dokumentacija



Instalacija biblioteka

U React-u se biblioteke i zavisnosti instaliraju preko npm-a ili yarn-a.

Instalacija se radi sledećo

npm install --ime_biblioteke--

Na sledećem primeru je prikazano kako može da se instalira MaterialUI biblioteka uz pomoć npm-a i yarn-a:

```
npm install @mui/material @emotion/react @emotion/styled
```

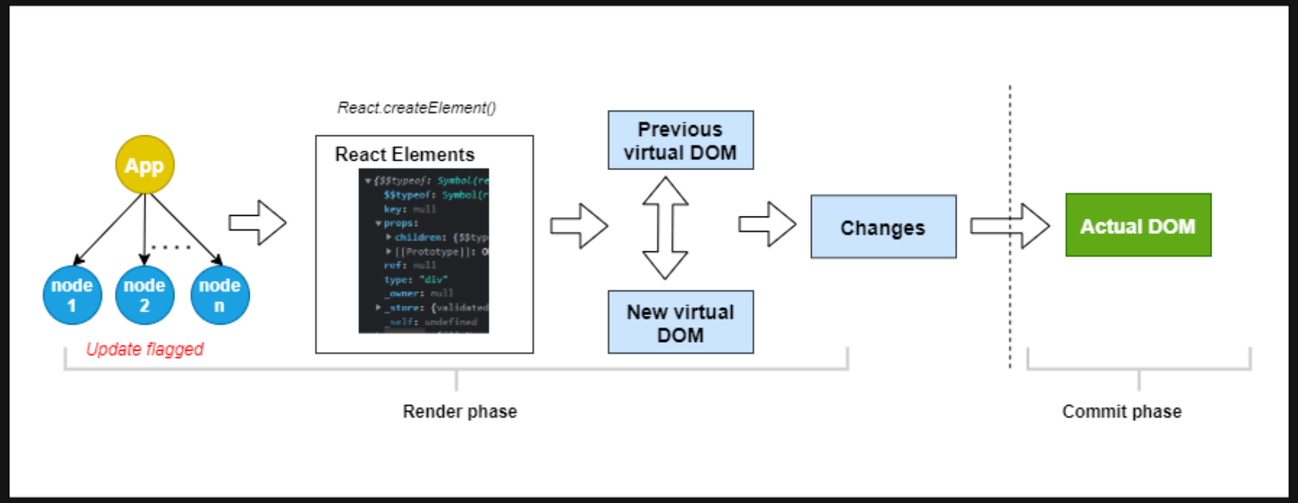
```
yarn add @mui/material @emotion/react @emotion/styled
```

Kako React funkcioniše

React funkcioniše na osnovu virtualnog DOM-a (Document Object Model) i uz pomoć koncepata reaktivnog programiranja.

Neki od osnovih koraka su sledeći:

- Inicijalizacija komponenti
- Pisanje JSX-a
- Renderovanje u virtualnom DOM-u
- Virtualni DOM i stvarni DOM
- Iscrtavanje promenjenih delova
- Reaktivno programiranje
- Hooks



Komponente

Glavni gradivni blokove za izgradnju korisničkog interfejsa.

Omogućavaju lakšu organizaciju i ponovno korišćenje koda.

Razolozi zašto se one koriste su:

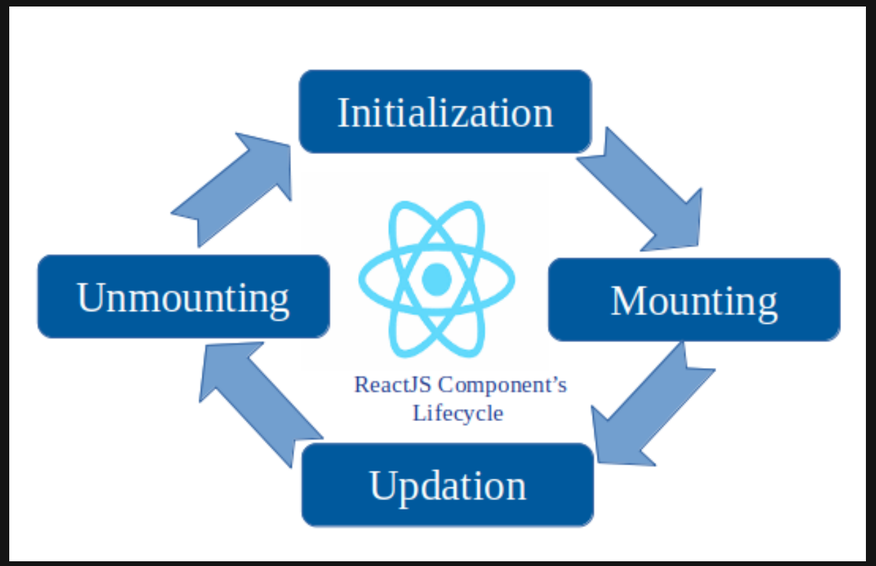
- Ponovna upotreba koda
- Lakše održavanje
- Modularnost
- Poboljšavaju čitljivost koda

Lifecycle komponente

Dakle svaka React komponenta prolazi kroz životni ciklus od trenutka kada su kreirane do trenutka kada su uništene. Životni ciklus se sastoji od nekoliko faza, a u pozadini se pozivaju i druge metode.

Faze životnog ciklusa su:

- Initialization
- Mounting (componentDidMount())
- Updating (componentDidUpdate())
- Unmounting (componentWillUnmount())



Funkcionalna komponenta

Funkcionalna komponenta je vrsta React komponente koja se definiše kao JavaScript funckija.

Danas se sve češće preferiraju zbog svoje jednostavnosti i čitljivosti.

Uz korišćenje ovog tipa komponenti koriste se hook-ovi.

```tsx
interface ComponentProps {
    name: string;
}
const BasicFunctionalComponent = (props: ComponentProps) => {
    return(
        <div>
            <p>This is basic functional component</p>
            <p>Name of this component is {props.name}</p>
        </div>
    )
}
export default BasicFunctionalComponent;
```

Klasna komponenta

Klasna komponenta je tip React komponente koji se definiše kao klasa i danas se sve manje i manje koriste.

Kod ove vrste komponenti se stanje inicijalizuje unutar konstruktora uz pomoć this.state, a zatim može da se ažurira uz pomoć metode this.setState.

```
import React, { Component } from 'react';

class Counter extends Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0,
    };
  }

  incrementCount = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.incrementCount}>Increment</button>
      </div>
    );
  }
}

export default Counter;
```

Reuse komponenta

Reuse komponenti odnosi se na praksu kreiranja komponenti na način koji će da omogući da se lako i efikasno komponente koriste na razilčitim mestima unutar iste ili druge aplikacije.

```
import BasicFunctionalComponent from "../BasicFunctionComponent/BasicFunctionalComponent";

const BasicReuseComponent = () => {
    return (
        <div>
            <BasicFunctionalComponent name="component1" />
            <BasicFunctionalComponent name="component2" />
            <BasicFunctionalComponent name="component3" />
        </div>
    )
}
export default BasicReuseComponent;
```

Funkcionalna vs Klasna

Funkcionalna komponenta:

- Jednostavnost i čitljivost
- Pregledan API
- Ponovna upotreba koda
- Lako testiranje
- Fleksibilnost

Klasna komponenta:

- Životni ciklus metode
- Legacy podrška
- Ref-ovi
- Više mogućnosti za optimizaciju

Virtual DOM

Predstavlja koncept koji se koristi za poboljšanje efikasnosti i performansi ažuriranja korisničkog interfejsa.

Ovaj proces pomaže u minimizovanju stvarnih manipulacija DOM-a i optimizuje proces osvežavanja stranice.

Način funkcionisanja:

1. Promena stanja ili svojstva
2. Kreiranje Virtualnog DOM-a koji je kopija stvarnog, ali sa ažuriranjima
3. Upoređivanje starog Virtualnog DOM-a sa novim kako bi se našle razlike
4. Generisanje minimalnog skupa promena
5. Primena ovih promena na stvarni DOM

Dakle menjaju se samo oni delovi koji su neophodni za ažuriranje, dok drugi ostaju tu gde jesu i time se postiže veća efikasnost.

MaterialUI library

Za lepši i lakši razvoj komponenti u React-u može da se koristi biblioteka MaterialUI, koja je jedna od mnogih koje postoje. Ona već poseduje neke komponente koje su gotove i koje se mogu upotrebiti bilo gde u kodu.

Dobra stvar kod ovih komponenti jeste to što su reuse, a uz pomoć njih takođe može da se napravi druga reuse komponenta.

```
import { Box, Typography } from "@mui/material";

const MaterialUIComponent = () => {
    return (
        <Box>
            <Typography>
                Basic example with component from Material UI library!
            </Typography>
        </Box>
    )
}
export default MaterialUIComponent;
```

State

Stanje (state) predstavlja objekat koji sadrži informacije o trenutnom stanju komponente.

Svaka komponenta može da ima svoje stanje i ono može da se menja tokom vremena u odgovorim na korisničke akcije.

Stanje se često koristi za dinamičko renderovanje i praćenje promena na UI-u.

Stanje se inicijalizuje uz pomoć Hook-a **useState** za funkcionalne komponente.

```
const [typeOfClients, setTypeOfClients] = useState<string>("all");
const [filter, setFilter] = useState("");
const [nameForNewClient, setNameForNewClient] = useState<string>("");
const [addClient, setAddClient] = useState<boolean>(false);
```

```
<Select
  value={typeOfClients}
  label="Age"
  onChange={(e) => handleChangeType(e.target.value)}
>
  <MenuItem value={"active"}>Show active</MenuItem>
  <MenuItem value={"all"}>Show all</MenuItem>
  <MenuItem value={"archived"}>Show archived</MenuItem>
</Select>
```

Managing state

Ovaj pojam se ondosi na način kako će da se prati i upravlja stanjem neke komponente.

Omogućava bolje organizovanje koda, reaktivnost na korisničke akcije i olakšava održavanje.

Promenu i praćenje stanja možemo da izvršimo na više načina, a to su:

- Poziv funkcije koja se dobija prilikom poziva useState hook-a
- Prosleđivanje stanja deci kao props i promena unutar deteta
- Kontekst za deljenje stanja između više komponenti
- Redux za kompleksnije upravljanje stanjem u aplikaciji

Data props

Kako bi deca mogla da znaju za stanje roditelja i na neki način uticali uvedeno je svojstvo (prop). Uz pomoć njij se roditeljski podaci prenose deci.

```
<ClientTable
  typeOfClient={typeOfClients}
  newClientName={nameForNewClient}
  addClient={addClient}
  setAddClient={setAddClient} filter={filter}
  />
```

```
useEffect(() => {
  if (addClient) {
    const newClient: addClientDTO = {
      name: newClientName,
    };

    api
      .post("/clients", newClient)
      .then((response) => {
        setClientsForTable([...clientsForTable, response.data]);
      })
      .catch((err) => console.error(err));

    setAddClient(false);
  }
  // eslint-disable-next-line react-hooks/exhaustive-deps
}, [addClient]);
```

Function Props

Ovo je drugi tip svojstava koje mogu da se proslede od roditelja ka deci. Njih predstavljaju funkcije.

Ove funkcije mogu da budu funkcije koje su se dobile pozivom hook-a useState i služe za menjanje stanja roditelja.

```
<ClientTable
  typeOfClient={typeOfClients}
  newClientName={nameForNewClient}
  addClient={addClient}
  setAddClient={setAddClient} filter={filter}
  />
```

```
useEffect(() => {
  if (addClient) {
    const newClient: addClientDTO = {
      name: newClientName,
    };

    api
      .post("/clients", newClient)
      .then((response) => {
        setClientsForTable([...clientsForTable, response.data]);
      })
      .catch((err) => console.error(err));

    setAddClient(false);
  }
  // eslint-disable-next-line react-hooks/exhaustive-deps
}, [addClient]);
```

State vs Prop

State prednosti :

- Dinamičko ažuriranje
- Lokalno stanje
- Održavanje podataka

Prop prednosti :

- Prenos podataka između komponenti
- Reuse komponente
- Lakše praćenje toka podatka

State mane:

- Nije deljen između komponenti
- Kompleksno upravljanje stanjem

Prop mane:

- Neefikasno u dubokom ugnježdenju
- Nisu promenljiva (immutable)

Event handling

Uz pomoć event handling-a možemo da reagujemo na korisničke interakcije, kao klik na neko dugme, unos teksta ili promene u komponenti.

Kako bi se obavio event handling neophodno je da postoji funkcija koja će da se izvriši kada dođe do tog događaja. Ime ove funkcije počinje obično sa handle.

```
<Dialog open={editOpen} onClose={handleEditClose}>
```

```
<Button onClick={handleEditClose}>Cancel</Button>
<Button onClick={handleEditSave} color="primary">
  Save
</Button>
```

```
const handleEditClose = () => {
  setEditOpen(false);
};

const handleDeleteOpen = (clientId: number) => {
  setSelectedClient(clientId);
  console.log(clientId);
  setDeleteOpen(true);
};

const handleDeleteClose = () => {
  setDeleteOpen(false);
};
```

Hooks

Hook-ovi predstavljaju glavni mehanizam za upravljanje stanjem i efektima kod funkcionalnih komponenti.

Kako bi se hook-ovi koristili u aplikaciji, oni najpre moraju da se uvezu iz React-a, a nakon toga se one pozivaju direktno unutar funkcionalne komponente.

Prednosti:

- Jednostavnost pisanja
- Povećana fleksibilnost
- Deljenje logike
- Poboljšani reuse

Mane:

- Promena paradigme
- Učenje novog API-ja

Postojeći hook-ovi:

- useState
- useEffect
- useContext
- useCallback i useMemo
- useRef
- useNavigate

useState

useState hook već postoji u React-u i on se koristi za dodavanje stanja komponentama.

On prima jedan argument i to je inicijalno stanje, a kao povratnu vrednost vraća niz sa dve vrenosti: trenutno stanje i funkciju za ažuriranje stanja.

```
const [typeOfClients, setTypeOfClients] = useState<string>("all");
const [filter, setFilter] = useState("");
const [nameForNewClient, setNameForNewClient] = useState<string>("");
const [addClient, setAddClient] = useState<boolean>(false);
```

useEffect

useEffect hook takođe postoji već unutar React-a samo je neophodno da se uveze. On služi za upravljanje side efektima, a to su stvari koje uključuju stvari poput asinhronih zahteva za podacima, promene ili manipulacija DOM-a.

Prima dva argumenta: prvi argument je funkcija koja sadrži kod side efekta, a drugi je niz zavisnosti. Zavisnosti su niz vrednosti (props ili state) koje se prate, i ako se bilo koja od njih promeni, useEffect će ponovo da izvrši svoj kod.

```
useEffect(() => {
  api
    .get("/clients")
    .then((response) => setClientsForTable(response.data))
    .catch((err) => console.error(err));

  // eslint-disable-next-line react-hooks/exhaustive-deps
}, []);
```

useNavigate

useNavigate hook dolazi iz biblioteke react-router-dom i on pruža mogućnosti za navigaciju između različitih ruta unutar aplikacije. U suštini on radi slično kao i <Link> komponenta React Router-a.

Za korišćenje je neophodno da se pozove unutar funkcionalne komponente i smesti u nekoj promenljivoj. Zatim se preko te promenljive radi navigacija na druge rute aplikacije.

```
import { useNavigate } from "react-router-dom";
 const navigate = useNavigate();
```

```
case 0:
  navigate("/home");
  break;
case 1:
  navigate("/dashboard");
  break;
case 2:
  navigate("/projects");
  break;
```

useCallback

useCallback je hook koji se koristi za memoizaciju callback funkcija. Memoizacija označava da će callback funkcija da se ponovo kreira ako se neka od zavisnosti promeni. Ovo je korisno kada je potrebno da se izbegne nepotrebno ponovno renderovanje.

Prima dva parametra: prvi parametar je callback funkcija koja će da se memoizuje, a drugi parametar je niz zavisnosti.

```
function MyComponent() {
  const [count, setCount] = useState(0);

  // Korišćenje useCallback za memoizaciju callback funkcije
  const handleClick = useCallback(() => {
    setCount(count + 1);
  }, [count]);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={handleClick}>Increment</button>
    </div>
  );
}
```

Context

U React-u predstavlja način kako za deljenje podataka između komponenti koje nisu direktno povezane. Korisno je da se upotrebi za teme, autentifikacju ili bilo koje druge globalne podatke.

Sastoji se iz dva dela:

- **Provider**: komponenta koja sadrži podatke koji će biti dostupni komponentama koje su obavijene ovom komponentom, odnsonso nalaze se unutar njene hijerarhije
- **Consumer**: komponenta koja koristi te podakte dostupne u context-u

Prednosti:

- Jednostavno deljenje podataka
- Smanjuje prop drilling

Mane:

- Teže praćenje toka podataka
- Performanski problemi
- Nepraktičan za male aplikacije

useContext

```tsx
interface AuthStateProviderProps {
  children: ReactNode;
}

const AuthStateContext = createContext<AuthStateContextI>(
  {} as AuthStateContextI
);
const context = useContext(AuthStateContext);
```

```tsx
const ProtectedRoutes = ({
    redirectPath ="/signin",
}: ProtectedRoutesProps) => {
    const { isAuthentificated } = useAuthContext();
    console.log("rg");
    return isAuthentificated() ? <Outlet /> : <Navigate to={redirectPath} />;
}
```

Custom hook

Custom hook se odnosi na hook koji programer sam razvija radi njegove potrebe zbog neke funkcionalnosti.

Oni omogućavaju da se izdvoji lohika komponenti u funkcije koje mogu da se koriste ponovo bilo gde unutar aplikacije i na veoma lak način.

Razlozi zašto je dobro da se kreiraju custom hook-ovi su:

- Izdvajanje i deljenje logike između komponenti
- Organizovaniji kod
- Lakše testiranje
- Lakše upravljanje stanjem
- Smanjuje se kod koji se ponavlja
- Čitljiviji koda

useSignalR

Ovo je primer custom hook-a koji treba da obavlja pribavljanje podataka u realnom vremenu korišćenjem signalR-a.

SignalR je biblioteka koja omogućava dvosmernu komunikaciju između klijenata i serveera u realnom vremenu, a uz pomoć nje se kreiraju aplikacije koje zahtevaju brzu i efikasnu komunikaciju.

```
const UseSignalR = (
  hubUrl: string,
  method: string,
  onReceiveUser: (newUser: TimeRecord) => void
) => {
  const [connection, setConnection] = useState<HubConnection | null>(null);

  useEffect(() => {
    const newConnection = new HubConnectionBuilder()
      .withUrl(hubUrl, {
        skipNegotiation: true,
        transport: HttpTransportType.WebSockets,
      })
      .build();

    newConnection.on(method, onReceiveUser);

    setConnection(newConnection);

    return () => {
      newConnection.stop();
    };
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, []);

  return { connection };
};
```

Fetching data

Kako bi postojali podaci koji će da se prikazuju unutar aplikacije, neophodno je da se izvrši data fetch. To se postiže korišćenjem fetch API-ja ili pomoću neke third-party library kao što je Axios.

```
useEffect(() => {
  const fetchData = async () => {
    try {
      const response = await fetch('https://api.example.com/data');
      const result = await response.json();
      setData(result);
    } catch (error) {
      console.error('Error fetching data:', error);
    } finally {
      setLoading(false);
    }
  };

  fetchData();
}, []);
```

```
useEffect(() => {
  const fetchData = async () => {
    try {
      const response = await axios.get('https://api.example.com/data')
      setData(response.data);
    } catch (error) {
      console.error('Error fetching data:', error);
    } finally {
      setLoading(false);
    }
  };

  fetchData();
}, []);
```

React router

React Router je biblioteka za upravljanje rutama u React aplikacijama, a ona omogućava i organizaciju i manipulaciju URL-om i time se omogućava dinamičko prikazivanje različtih delova korisničkog interfejsa

Funkcionalnost:

- Deklarativno definisanje ruta
- Upravljanje istorijom pregledača
- Ugnježdene rute
- Prenos parametara ili podataka između delova apliakcije

Unutar <Router> komponente treba da se navedu rute koje postoje i koje se komponente na tim rutama renderuju.

```
<Routes>
  <Route element={<ProtectedRoutes />}>
    <Route element={<Navigation />}>
      <Route path="" element={<Navigate to="/home"></Navigate>}></Route>
      <Route path="/home" element={<Home />}></Route>
      <Route path="/dashboard" element={<Dashboard />}></Route>
      <Route path="/projects" element={<Projects />}></Route>
      <Route
        path="/clients"
        element={<Clients />}
      ></Route>
      <Route path="/tags" element={<Tags />}></Route>
      <Route path="/calendar" element={<Calendar />}></Route>
    </Route>
  </Route>
  <Route path="/signin" element={<SingIn />}></Route>
  <Route path="/signup" element={<SignUp />}></Route>
  <Route path="*" element={<NotFound />}></Route>
</Routes>
```

Redux

Redux je JavaScript biblioteka koja se koristi unutar React aplikacijama kako bi se upravljalo stanjem aplikacije. Glavni cilj jeste da se olakša upravljanje stanjem, posebno kada je aplikacija kompleksna.

Glavni koncepti kod Redux-a su:

- **Store**: svi podaci koji se koriste u aplikaciji se ovde smeštaju
- **Actions**: akcije koje su objekti i opisuju događaje ili promene u aplikaciji
- **Reducers**: funkcije koje opisuju kako se stanje aplikacije menja u odgovoru na akcije
- **Dispatch**: predstavlja proces slanja akcija reducer-ima. Kada se dogodi neki događaj u aplikaciji, stvara se odgovarajuća akcija i to se putem dispečovanja šalje Redux store-u
- **Subsribe**: komponente koje žele promene u stanju se pretplaćuju na Redux store

Prednosti:

- Centralizovano upravljanje
- Predvidivo ponašanje
- Lako praćenje akcija i promena

Mane:

- Složenost za manje apliikacije
- Veći boilerplate kod
- Dodatna apstrakcija

Zakljucak

Dakle React je pogodan za razvoj efikasnih i održivih korisničkih interfejsa.

Obradili smo osnovne i napredne koncepte, uključujući Hooks i Context API, a integracija Material-UI biblioteke pruža nam gotove komponente i estetski dobar dizajn.

Takođe prikazan je način kako se podaci mogu prikupiti u realnom vremenu uz pomoć signalR kocepta.

Korišćenje React-a danas je neophodno zbog efikasnog upravljanja stanjem, fleksibilnosti razvoja, aktivnog React ekosistema i investicije u budućnost veb tehnologija.

Hvala na pažnji!

Pitanja?