UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

Limbaje și tehnologii web

Proiect

To-Do List

Student: Laura-Elena POLEUCĂ
Grupa: SDTW1B

Iași, 2024

Cuprins

Scopul lucrării

O aplicație simplă to-do list reprezintă un instrument eficient pentru cei interesați să-și îmbunătățească modul în care își organizează și gestionează activitățile zilnice. Prin intermediul acestei aplicații, utilizatorii au posibilitatea de a înregistra și de a-și planifica sarcinile și obiectivele într-un mod accesibil și ușor de administrat.

Cu funcții precum adăugarea, editarea și ștergerea task-urilor, utilizatorii pot menține o listă de activități clară și actualizată. Capacitatea de a vizualiza separat task-urile active și cele realizate contribuie la concentrarea asupra priorităților curente, în timp ce posibilitatea de a edita task-urile existente aduce flexibilitate în adaptarea listei la schimbări. În ansamblu, aplicația oferă un instrument practic pentru îmbunătățirea organizării personale și creșterea eficienței în gestionarea sarcinilor de zi cu zi.

Scopul acestei lucrări este de a evidenția procesul de dezvoltare al unei aplicații de tipul to-do list și de a ilustra integrarea tehnologiilor React, Redux, Node.js și MongoDB în acest context.

Concepte teoretice

Proiectul este organizat în două module separate pentru frontend și backend.

Modulul de backend reprezintă nucleul funcțional al aplicației, concentrându-se pe implementarea operațiilor din backend, gestionarea comunicării cu baza de date MongoDB precum și furnizarea API-urilor pentru interacțiunea cu modulul de frontend. Acest modul gestionează eficient solicitările HTTP primite de la frontend, dirijându-le către controlerele corespunzătoare, care manipulează și procesează datele, asigurând coerența și integritatea acestora.

Modulul de frontend reprezintă componenta vizibilă și interactivă a aplicației, axându-se pe dezvoltarea și gestionarea interfeței cu utilizatorul. Acesta folosește tehnologii precum React și Redux pentru a construi componente reutilizabile, care oferă o experiență de utilizator dinamică și eficientă. Modulul de frontend realizează interacțiunea directă cu utilizatorul, gestionând evenimente și solicitând informații de la modulul de backend prin intermediul API-urilor. Prin intermediul bibliotecilor precum Redux, se gestionează starea aplicației într-un mod predictibil, facilitând actualizările instantanee ale interfeței la schimbări ale datelor.

React cu Redux

În cadrul aplicației to-do list, s-a optat pentru utilizarea React, o biblioteca JavaScript ce se distinge prin abordarea sa modernă și modulară în dezvoltarea interfețelor de utilizator. Caracterizată prin simplitate, eficiență și reutilizabilitate, React redefinește modul în care sunt construite și gestionate interfețele aplicațiilor web.

Un aspect distinctiv al React constă în arhitectura sa orientată către componente. Interfața de utilizator este împărțită în componente independente, fiecare responsabilă pentru o parte specifică a interfeței.

Ecosistemul extins de pachete pentru React joacă un rol important în extinderea funcționalității aplicației. Aceste pachete suplimentare, dezvoltate de comunitatea React și de terțe părți, adaugă diverse capabilități și utilități pentru a face dezvoltarea mai eficientă și mai flexibilă.

Unul dintre acestea este Axios. Axios este o bibliotecă special concepută pentru gestionarea cererilor HTTP în aplicații web. Una dintre caracteristicile distincte ale Axios constă în abordarea sa bazată pe Promises, care simplifică gestionarea codului asincron.

Pentru gestionarea stării în aplicația construite cu React s-a utilizat Redux. Conceptul din sptele bibliotecii Redux este aceea de a gestiona întreaga stare a aplicației printr-un singur obiect numit "store". Un concept cheie în Redux este "actions". Acțiunile reprezintă evenimente care descriu schimbări în starea aplicației și sunt capturate de "reducers". Reducerii sunt funcții care preiau starea curentă și o modifică în funcție de acțiunile primite, producând astfel o nouă stare. Prin utilizarea unui flux unidirecțional de date, Redux oferă un control precis asupra modului în care starea aplicației este actualizată și utilizată.

NodeJS cu ExpressJS

Node.js este unul dintre cele mai folosite limbaje in prezent pentru implementarea aplicatiilor web, intrucat acesta permite dezvoltarea aplicatiei cu ajutorul unui singur limbaj atat in front-end, cat si back-end. Impreuna cu functionalitatea de procesare async conferita prin intermediul event loop-ului, Node.js devine o alegere potrivita pentru implementarea unui server web, capabil sa raspunda la cat mai multe cereri. Dezvoltarea unui server este de asemenea facilitata de express.js, un framework ce permite gestionarea eficienta a rutelor pentru cererile HTTP.

MongoDB

MongoDB, o bază de date orientată pe documente, se integrează armonios cu aplicațiile Node.js, oferind un model de stocare flexibil și eficient. Datele aplicației sunt memorate în format BSON, similar JSON, facilitând manipularea lor într-un mod ușor de gestionat și adaptat la schimbările frecvente de cerințe.

Exemple practice

Conectarea la baza de date

Se realizeaza prin intermediul “mongoose”, un library bine-cunoscut pentru gestionarea interacțiunii cu baze de date MongoDB în cadrul aplicațiilor Node.js. Aceasta oferă un ORM (Object-Document Mapping) pentru a simplifica manipularea datelor din baza de date MongoDB.

```
import mongoose from "mongoose";
import dotenv from "dotenv";

dotenv.config();

const USERNAME = process.env.DB_USERNAME;
const PASSWORD = process.env.DB_PASSWORD;

const Connection = () => {
  const MONGODB_URI = `mongodb+srv://${USERNAME}:${PASSWORD}@todo-
app.b3rqnzx.mongodb.net/?retryWrites=true&w=majority`;
  mongoose.connect(MONGODB_URI, { useNewUrlParser: true });
  mongoose.connection.on("connected", () => {
    console.log("Database connected");
  });
  mongoose.connection.on("disconnected", () => {
    console.log("Database disconnected");
  });
  mongoose.connection.on("error", () => {
    console.log("Error connecting to database");
  });
};

export default Connection;
```

Definirea structurii datelor

Aplicația utilizează modele pentru a defini modul în care datele sunt structurate și stocate în cadrul colecțiilor din MongoDB. Modelul Todo este definit prin intermediul unei scheme (TodoSchema). Această schemă descrie proprietățile pe care trebuie să le aibă obiectele todo stocate în baza de date, inclusiv conținutul sarcinii (data), starea de finalizare (done), și data de creare (createdAt).

```
import mongoose from "mongoose";
const TodoSchema = new mongoose.Schema({
    data: {
        type: String,
        required: true
    },
    done: {
        type: Boolean,
        default: false
    },
    createdAt: {
        type: Date,
        default: Date.now
    }
})
```

```
const todo = mongoose.model('todo', TodoSchema);

export default todo;
```

Controller

Controlerul utilizează modelul definit anterior pentru a comunica cu baza de date MongoDB și pentru a efectua diverse operații asupra datelor. Acesta exportă un set de funcții care corespund diferitelor acțiuni pe care utilizatorii pot să le întreprindă în legătură cu sarcinile, precum adăugarea unui nou task in baza de date, actualizarea conținutului unui task, ștergerea unui task din baza de date, precum și returnarea tuturor sarcinilor stocate în baza de date. Aceste funcții corespund operațiilor CRUD asociate task-urilor din to-do list și sunt mai departe folosite în rutele aplicației.

```
import Todo from "../models/Todo.js";
export const addTodo = async(request, response) =>{
    try {
        const newTodo = await Todo.create({
            data: request.body.data,
            createdAt: Date.now()
        })

        await newTodo.save();

        return response.status(200).json(newTodo);
    }catch (error) {
        return response.status(500).json(error.message);
    }
}

export const getAllTodos = async (request, response) => {
    try {
        const todos = await Todo.find({}).sort({ 'createdAt': -1 })

        return response.status(200).json(todos);
    } catch (error) {
        return response.status(500).json(error.message);
    }
}
```

Rutele, definite prin intermediul unui sistem de rutare în aplicația ta Node.js, stabilesc cum cererile HTTP primite de server sunt direcționate către diferitele funcții de controler asociate.

```
route.post("/todos", addTodo);
route.get("/todos", getAllTodos);
```

Structura aplicației client

Pe partea de frontend, aplicatia prezinta 3 componente principale:

1. header (care contine titlul aplicatiei);
2. form (pentru introducerea todo-ului dorit) ;
3. componenta *Todos* (care se ocupa de afisarea, filtrarea si interactionarea corespunzatoare a todo-urilor).

Primele doua componente din aceasta lista sunt stateless, avand roluri simple in cadrul aplicatiei web. Complexitatea aplicatiei se observa cel mai bine in cadrul ultimei componente, intrucat aceasta imbina state management si interactiunea cu redux pentru a facilita functionalitatile aplicatiei.

Folosirea React-Redux

```
export const Todos = () => {
  const dispatch = useDispatch();

  const todos = useSelector((state) => state.todos);
  const currentTab = useSelector((state) => state.currentTab);

  useEffect(() => {
    dispatch(getAllTodos());
  }, []);
```

Spre exemplu, aceasta componenta se foloseste de metoda useSelector pentru a putea accesa informatii de stare din store-ul aplicatiei. Astfel, modificarile asupra starii si actualizarile acesteia sunt centralizate prin intermediul redux. Totodata, dispatch-ul este folosit pentru a putea declansa schimbari la nivel de stare a aplicatiei. Mai precis, dispatch este folosit in acest context impreuna cu hook-ul useEffect, pentru a putea declansa operatia de cerere a todo-urilor din server si actualizare a starii in client.

```
export const getAllTodos = () => async (dispatch) => {
    try {
        const res = await axios.get("http://localhost:8081/todos");
        dispatch({ type: GETALL_TODO , payload: res.data });
    } catch (error) {
        console.log('Error while calling getAllTodos API ', error);
    }
}
```

Acesta metoda getAllTodos este una dintre actiunile configurate prin redux. Aceasta se va ocupa de realizarea cererii HTTP catre server, urmand ca datele primite sa fie trimise mai departe pentru procesare (sau lansarea unei erori in cazul unei probleme) printr-o noua actiune. Datele vor fi interceptate de un reducer, al carui scop este strict modificarea starii aplicatiei. Acest reducer va putea realiza modificari corespunzatoare pe baza obiectului trimis, mai precis tipul acestuia, GETALL_TODO.

```
export const todoReducers = (state = [], action) => {
  switch (action.type) {
    case actionTypes.ADDNEW_TODO:
      console.log("Added new todo to state", action)
      return [action.payload, ...state];
    case actionTypes.GETALL_TODO:
      return action.payload;
```

Pentru a permite acces catre store-ul configurat prin acest reducer, acesta trebuie trimis catre o componenta Provider, care sa permita accesul la stare intregii aplicatii. `import store from "./redux/store";`

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Provider store={store}>
    <React.StrictMode>
      <App />
    </React.StrictMode>
  </Provider>
);
```

Astfel, prin acest mecanism s-a respectat principiul separarii responsabilitatilor, intrucat codul aplicatiei permite ca adaugarea sau modificarea unor functionalitati sa se

realizeze in mod centralizat. Aplicatia foloseste acelasi mecanism prin react redux pentru a implementa functionalitatea de vizualizare a diferitelor categorii de todo-uri:

- componenta Todos foloseste metoda getTodos pentru a realiza filtrarea pentru afisare a todo-urilor dorite
- toate todo-urile disponibile sunt incarcate folosind state-ul din redux
- la modificarea unui tab, componenta Tabs se foloseste de dispatch pentru a semnaliza schimbarea tab-ului activ, folosind actiunea toggleTab
- actiunea toggleTab trimite un dispatch catre reducer pentru a modifica starea aplicatiei. Daca ar fi nevoie de a realiza noi actiuni la schimbarea tab-ului activ, metoda toggleTab ar putea sa contina noua logica necesara, fara a afecta operatiile de state management
- reducer-ul primeste noul tab selectat, astfel incat se declanseaza o schimbare la nivel de stare, lucru care determina operatiunea de re-rendering a componentelor stateful

Concluzii

Aplicatia serveste ca un punct potrivit de plecare pentru o aplicatie mai extinsa, care sa contina o serie de functionalitati aditionale, precum implementarea unei modalitati de autentificare, atat prin metode standard (creare de cont), cat si prin metode single sign on (cont google, facebook, etc.) .

Încorporarea react-redux a reprezentat o oportunitate de invatare pentru o metodologie de state management. Aceste cunostinte pot fi aplicate mai departe si in alte proiecte, intrucat redux reprezinta un concept general, ce nu tine de o anumita librarie/framework web.

În cazul unei aplicatii de dimensiuni mai reduse, folosirea react-redux nu este absolut obligatorie, dar faciliteaza respectarea principiilor de dezvoltare de cod precum SOLID sau DRY.

Folosirea unei baze de date reprezinta o imbunatatire considerabila fata de metodele folosite pana in prezent de stocare in memorie a datelor problemei

Bibliografie

- Documentație React: https://react.dev/learn
- React Redux: https://react-redux.js.org/
- Axios: https://axios-http.com/docs/intro
- Font Awesome: https://fontawesome.com/
- NodeJS: https://nodejs.org/en
- ExpressJS: https://expressjs.com/
- MongoDB: https://www.mongodb.com/
- Cursuri Web Dev: https://fullstackopen.com/, https://www.theodinproject.com/paths/full-stack-javascript,
- Aplicatii MERN: https://github.com/search?q=MERN&type=repositories