Görev 10: Firebase ile Depo Yönetim Sistemi

Amaç

Bu ödevde, bir depo yönetim sistemi geliştireceksiniz. Uygulamanız, kullanıcıların depodaki ürünleri yönetmelerini, envanteri kontrol etmelerini ve ürünlerin giriş-çıkış işlemlerini gerçekleştirmelerini sağlayacak. Bu tür bir uygulama, küçük işletmelerin depolarındaki ürünleri düzenli tutmak ve stoklarını yönetmek için kullanılabilir.

Gereksinimler

1. Proje Teknolojileri:
    - React (TypeScript ile)
    - Redux (Global state yönetimi için)
    - Firebase Authentication (Kullanıcı kayıt ve giriş işlemleri için)
    - Firestore (Depo verileri ve envanter yönetimi için)
    - React Router (Sayfa yönlendirmeleri için)
    - Tailwind CSS (Stil yönetimi için)
2. Uygulama İşlevsellikleri:
    - **Kullanıcı Kayıt ve Giriş:** Firebase Authentication kullanarak kullanıcıların sisteme kayıt olup giriş yapmalarını sağlayın.
    - **Depo Yönetimi:** Uygulama içerisinde depoya ürün ekleme, mevcut ürünleri görüntüleme, ürünleri güncelleme ve silme işlemlerini Firestore ile yönetin.
    - **CRUD İşlemleri:** Firestore'u kullanarak depo içerisindeki ürünlerle ilgili CRUD (Create, Read, Update, Delete) işlemlerini gerçekleştirin.
    - **Yönlendirme:** React Router kullanarak uygulama içerisinde giriş, kayıt, envanter listesi, ürün detayları gibi sayfalar arası geçişleri yönetin.
    - **Stil Yönetimi:** Uygulamanın görünümünü düzenlemek için Tailwind CSS kullanın.

Proje İpuçları ve Açıklamalar

- **Başlangıç:** Projenizi oluşturmak için `create-react-app` komutunu TypeScript ve Redux ekleyerek kullanabilirsiniz. Firebase konfigürasyonunu proje başında ayarlayın ve Tailwind CSS'i projenize dahil edin.

Unset

```
npx create-react-app stock-management-app --template typescript
npm install @reduxjs/toolkit react-redux firebase react-router-dom
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

- **Tailwind CSS Entegrasyonu:**
  Tailwind CSS'i projeye dahil ettikten sonra, `tailwind.config.js` dosyasını aşağıdaki gibi düzenleyin ve stil dosyanızda Tailwind direktiflerini ekleyin:

Unset

```
// tailwind.config.js
module.exports = {
  content: ["./src/**/*.{js,jsx,ts,tsx}"],
  theme: {
    extend: {},
  },
  plugins: [],
};
```

Unset

```
/* index.css */
@tailwind base;
@tailwind components;
@tailwind utilities;
```

- **Kullanıcı Kayıt ve Giriş:**
  Firebase Authentication'ı kurmak için Firebase konsolunda yeni bir proje oluşturun ve gerekli API anahtarlarını proje dosyanıza ekleyin. Kullanıcı kayıt ve giriş işlemleri için basit bir form yapısı oluşturun. Burada, kullanıcıların e-posta ve şifre ile giriş yapmasını sağlayan genel bir yapı kurmayı hedefleyin.
- **Redux Kullanımı:**
  Redux kullanarak uygulama genelindeki state yönetimini sağlayın. Aşağıda, örneğin bir alışveriş sepeti uygulamasında, kullanıcıların ürün ekleyip çıkardığı bir yapı düşünün:

Unset

```
// cartSlice.ts
import { createSlice } from "@reduxjs/toolkit";

interface CartState {
  items: { id: string; name: string; quantity: number }[];
}

const initialState: CartState = {
  items: [],
};

const cartSlice = createSlice({
  name: "cart",
  initialState,
  reducers: {
    addToCart: (state, action) => {
      state.items.push(action.payload);
    },
    removeFromCart: (state, action) => {
      state.items = state.items.filter(item => item.id !==
action.payload);
    },
  },
});

export const { addToCart, removeFromCart } = cartSlice.actions;
export default cartSlice.reducer;
```

Yukarıdaki örnek, depo yönetimindeki CRUD işlemlerine benzer bir yapıdadır ancak alışveriş sepeti üzerinden düşünülmüştür. Redux state yönetimini benzer şekilde kullanarak ürünlerinizi yönetin.

- **React Router Kullanımı:**
  Uygulama içi sayfa yönlendirmelerini sağlamak için React Router'ı kullanın. Örneğin, farklı eğitim modüllerine yönlendiren bir eğitim platformu uygulamasını düşünün:

Unset

```
// App.tsx
```

```
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import CourseList from './pages/CourseList';
import CourseDetails from './pages/CourseDetails';
import Login from './pages/Login';

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<CourseList />} />
        <Route path="/course/:id" element={<CourseDetails />} />
        <Route path="/login" element={<Login />} />
      </Routes>
    </Router>
  );
}

export default App;
```

Bu örnek, eğitim modülleri ve detay sayfalarını yöneten bir yapıdadır. Depo yönetim sisteminizde benzer bir yönlendirme yapısını düşünün.

- **CRUD İşlemleri:**
  Firestore ile veri yönetimi işlemlerini gerçekleştirin. Örneğin, bir görev yönetim uygulamasında, görevleri eklemek, güncellemek ve silmek için CRUD işlemleri yapabilirsiniz:

Unset

```
// Task component example
import React, { useState } from 'react';

function AddTask() {
  const [task, setTask] = useState('');

  const handleAddTask = () => {
    // Firestore'a görev ekleme işlemi burada yapılacak
  };

  return (
    <div className="p-4 bg-gray-100 shadow rounded">
```

```
      <input
        className="border p-2 w-full mb-2"
        type="text"
        placeholder="Görev Adı"
        value={task}
        onChange={(e) => setTask(e.target.value)}
      />
      <button
        className="bg-green-500 text-white p-2 rounded hover:bg-green-700"
        onClick={handleAddTask}
      >
        Görev Ekle
      </button>
    </div>
  );
}

export default AddTask;
```

Bu görev yönetim örneği, depodaki ürünlerin CRUD işlemlerine benzerlik göstermektedir. Uygulamanızda depo ürünlerini yönetmek için bu tür yapıların benzerlerini oluşturabilirsiniz.

İpuçları ve Kaynaklar

- **Dokümantasyonlar:** Her teknoloji için resmi dokümantasyonları inceleyin. Özellikle Firebase ve Redux dokümanlarında uygulamaya özel kullanımlar konusunda önemli ipuçları bulabilirsiniz.
- **Örnek Kodlar:** Verilen örnek kodlar, tam çözümler değil, benzer gereksinimler için nasıl kod yazılabileceğine dair fikir vermek içindir.

Değerlendirme Kriterleri

- Uygulamanın belirtilen tüm teknolojileri doğru ve uygun şekilde kullanması.
- Kullanıcı arayüzünün basit ve kullanıcı dostu olması.
- Kodun düzenli, anlaşılır ve yorumlanabilir olması.
- Redux ve Firestore kullanılarak başarılı bir şekilde global state yönetimi ve veri işlemlerinin yapılması.

Teslim

- Görev klasörünü zipleyip classroom’a yükleyiniz.

Notlar

- Ödevi tamamlarken karşılaşabileceğiniz zorluklarla başa çıkmanız için internet kaynaklarından faydalanabilirsiniz. Ancak, tamamen kopyala-yapıştır yapmaktan kaçının; anladığınız kodu yazmaya özen gösterin.

Hepinize başarılar 💙