

“Northwind” ve “ReactProject” Projesi ile ne yaptık ?

React Projesi Nedir?

Bu React projesi, bir **e-ticaret ön yüzü** veya **ürün yönetim paneli** olarak tanımlanabilir. Kullanıcıların ürünleri listeleyip sepete ekleyebildiği, ürün detaylarını görebildiği ve potansiyel olarak yeni ürünler ekleyebildiği bir arayüz sunuyor. Redux ile sepet yönetimi ve Semantic UI ile modern bir tasarım, bunu bir alışveriş sitesinin ön yüzü gibi yapıyor.

Northwind, bu React projesinin **backend veri kaynağı** olarak kullanılmış. Daha spesifik olarak:

1. Veri Sağlayıcısı:

- Northwind, ürün (Product) ve kategori (Category) verilerini PostgreSQL veritabanından çekip React'a sunuyor. Örneğin:
 - `getProducts()` → Tüm ürünleri listeleme (ProductList için).
 - `getByProductName()` → Ürün detayını alma (ProductDetail için).

2.

- ProductService.js'deki API çağrıları, Northwind'in ProductsController'ındaki endpoint'lere istek gönderiyor.

3. REST API Altyapısı:

- Northwind, RESTful bir API sunarak React'ın dinamik veri ihtiyaçlarını karşılıyor:
 - `/api/products/getall`: Ürün listesini döndürür.
 - `/api/products/getByProductName`: Belirli bir ürünün detayını döndürür.

4.

- Bu endpoint'ler, React tarafında tablo doldurma, sepet güncelleme ve detay gösterme gibi işlemlerde kullanılıyor.

5. Dokümantasyon ve Test:

- OpenApiConfig.java ile Northwind, Swagger UI üzerinden API'lerinizi belgeleyip test edilebilir hale getiriyor. Bu, React geliştirme sürecinde hangi endpoint'lerin ne döndüğünü anlamınızı ve test etmenizi kolaylaştırmış.

6. Potansiyel Yönetici Özellikleri:

- UsersController ve ProducttAdd gibi yapılar, Northwind'in sadece veri sağlamakla kalmayıp kullanıcı ekleme (/api/users/add) gibi işlemleri de desteklediğini gösteriyor. React'taki ürün ekleme formu, ileride bu backend'e bağlanabilir.

Northwind'in Rolü

Northwind, bu projede **e-ticaret uygulamasının backend'ini** oluşturuyor. Ürün verilerini saklayıp yönetiyor, REST API üzerinden bu verileri React'a sunuyor ve potansiyel olarak kullanıcı yönetimi gibi ek işlevler sağlıyor. Klasik bir Northwind veritabanı (Microsoft'un örnek veritabanı) baz alınmış gibi görünüyor, ancak özelleştirilmiş bir Spring Boot uygulaması olarak geliştirilmiştir.

Genel Resim

- **React Projesi:** Bir e-ticaret veya ürün yönetim arayüzü (frontend). Kullanıcıların ürünleri görüp sepete ekleyebildiği, detayları inceleyebildiği ve potansiyel olarak ürün ekleyebildiği bir uygulama.
- **Northwind Projesi:** Bu React arayüzünün backend'i. Ürün ve kategori verilerini saklayıp REST API ile sunuyor, Swagger UI ile belgeleniyor ve React'in veri ihtiyaçlarını karşılıyor.

- **Entegrasyon:** React, Northwind'den veri çekiyor (axios ile), Northwind ise bu verileri PostgreSQL'den sağlayıp React'a JSON formatında dönüyor.
-
- "Spring Boot (Northwind) ile bir e-ticaret backend'i geliştirerek REST API'ler sundum ve React frontend ile entegre ettim."
- "Northwind projesini, ürün ve kategori verilerini yönetmek ve React arayüzüne dinamik veri sağlamak için kullandım."
- "React ile bir e-ticaret ön yüzü tasarladım ve Spring Boot backend ile veri entegrasyonu gerçekleştirdim."
-

React projesinin northwind' den nasıl veri aldığıni projemizdeki kodlar ile açıklayalım:

1. API Endpoint'lerinin Uyumu:

- React projesindeki `ProductService.js` dosyasında şu kodlar var:

```
javascript                                         Methi sar   Kopyala

import axios from "axios";

export default class ProductService {
    getProducts() {
        return axios.get("http://localhost:8080/api/products/getall")
    }

    getByProductName(productName) {
        return axios.get("http://localhost:8080/api/products/getByProductName?productName=" + productName)
    }
}
```

- Bu kodlar, Spring Boot backend'deki `ProductsController.java` içindeki endpoint'lerle doğrudan eşleşiyor:

- Bu kodlar, Spring Boot backend'deki `ProductsController.java` içindeki endpoint'lerle doğrudan eşleşiyor:

```
java Methi sar Kopyala

@RestController
@RequestMapping("api/products")
public class ProductsController {
    @GetMapping("/getall")
    public DataResult<List<Product>> getAll() {
        return this.productService.getAll();
    }

    @GetMapping("/getByProductName")
    public DataResult<Product> getByProductName(@RequestParam String productName) {
        return this.productService.getByProductName(productName);
    }
}
```

- Yani React frontend, Northwind backend'den veri çekmek için bu API'leri kullanıyor.

2. CORS Desteği:

- `ProductsController.java`'da `@CrossOrigin` annotationu var:

```
java Methi sar Kopyala

@CrossOrigin
@RestController
@RequestMapping("api/products")
public class ProductsController { ... }
```

- Bu, React uygulamasının (örneğin, `http://localhost:3000` adresinden) backend'e (`http://localhost:8080`) istek göndermesine izin verir. Farklı portlar arasında iletişim için CORS gereklili olduğundan, bu entegrasyonu doğrular.

3. Veri Yapısı Uyumu:

- Backend'den dönen veri (`DataResult<List<Product>>`), React tarafından işleniyor. Örneğin, `ProductList.js`'de:

```
javascript
```

Methi sar

Kopyala

```
useEffect(() => {
  let productService = new ProductService();
  productService.getProducts().then(result => setProducts(result.data.data));
}, []);
```

- `result.data.data`, Northwind'in `SuccessDataResult` yapısından gelen ürün listesini temsil eder (`data` içindeki `List<Product>`).

Bu bulgular, React projesinin Northwind backend'ine bağımlı olduğunu ve birbirine entegre çalıştığını gösteriyor.

Projenin bazı işlevsel özellikleri ve genel açıklamalar :

React Projesi Aslında Ne Projesi?

React projesi, bir e-ticaret veya ürün yönetim arayüzü gibi görünüyor. Bunun nedenlerini ve özelliklerini şöyle açıklayayım:

Özellikler ve Amaç

1. Ürün Listeleme:

- `ProductList.js`, bir ürün tablosu oluşturuyor ve her ürün için ad, fiyat, stok, kategori gibi bilgileri gösteriyor:

```
javascript
```

Methi sar

Kopyala

```
<Table.Cell>{product.productName}</Table.Cell>
<Table.Cell>{product.unitPrice}</Table.Cell>
<Table.Cell>{product.unitsInStock}</Table.Cell>
<Table.Cell>{product.category.categoryName}</Table.Cell>
```

- Bu, bir e-ticaret sitesinde ürün katalogu gibi bir işlevi ima eder.

2. Sepet İşlevselliği:

- `CartSummary.js` ve `CartDetail.js` ile bir sepet sistemi var:

```
javascript
```

Methi sar Kopyala

```
const {cartItems} = useSelector(state => state.cart);
cartItems.map((cartItem) =>
  <Dropdown.Item key={cartItem.product.id}>
    {cartItem.product.productName}
    <Label>{cartItem.quantity}</Label>
  </Dropdown.Item>
);
});
```

- `cartReducer.js`’de `ADD_TO_CART` ve `REMOVE_FROM_CART` eylemleriyle sepet yönetimi sağlanıyor:

```
javascript
```

Methi sar Kopyala

```
case ADD_TO_CART:
  let product = state.cartItems.find((c) => c.product.id === payload.id);
  if (product) {
    product.quantity++;
    return { ...state };
  } else {
    return {
      ...state,
      cartItems: [...state.cartItems, { quantity: 1, product: payload }]
    };
}
```

- Bu, kullanıcıların ürünleri sepete ekleyip çıkarabileceği bir alışveriş sepetini temsil eder.

3. Ürün Ekleme:

- `ProductAdd.jsx`, yeni ürün eklemek için bir form sunuyor:

```
javascript
```

Methi sar Kopyala

```
const initialValues = { productName: "", unitPrice: 10 };
<ErcanTextInput name="productName" placeholder="Ürün Adı"/>
<ErcanTextInput name="unitPrice" placeholder="Ürün Fiyatı"/>
```

- Şu an backend’de veri gönderme implemente edilmemiş (sadece `console.log`), ama bu özellik bir yönetici arayüzü veya satıcı paneli olabileceği gösteriyor.

4. Ürün Detay Sayfası:

- ProductDetail.js, belirli bir ürünün detaylarını gösteriyor:

```
javascript
```

Methi sar Kopyala

```
productService.getByProductName(name).then(result => setProduct(result.data.data));  
<Card.Header>{product.productName}</Card.Header>  
<Card.Meta>{product.category ? product.category.categoryName : 'Kategori yok'}</Card.Meta>
```

- Bu, kullanıcıların bir ürüne tıkladığında detaylı bilgi alabileceği bir sayfayı temsil eder.

5. Kullanıcı Giriş/Cıkış:

- SignedIn.js ve SignedOut.js, basit bir oturum yönetimini içeriyor:

```
javascript
```

Methi sar Kopyala

```
const [isAuthenticated, setIsAuthenticated] = useState(true);  
{isAuthenticated ? <SignedIn signOut={handleSignOut} /> : <SignedOut signIn={handleSignIn} />}
```

- Bu, kullanıcıların oturum açıp çıkabileceği bir özelliği ima eder (henüz backend ile kimlik doğrulama bağlı değil gibi).

Sonuç: React Projesi Nedir?

Bu React projesi, bir e-ticaret ön yüzü veya ürün yönetim paneli olarak tanımlanabilir. Kullanıcıların ürünleri listeleyip sepete ekleyebildiği, ürün detaylarını görebildiği ve potansiyel olarak yeni ürünler ekleyebildiği bir arayüz sunuyor. Redux ile sepet yönetimi ve Semantic UI ile modern bir tasarım, bunu bir alışveriş sitesinin ön yüzü gibi yapıyor.

React Projesin “northwind” backend’imizle çalışma mantığını açıklıyorum :

tam olarak uygulamanızın işleyişini ve <http://localhost:3000/> adresine girdiğinizde gördüğünüz davranışları hangi kodların sağladığını açıklayayım. Ayrıca, backend’den verinin hangi aşamada alındığını da adım adım göstereyim. Sorunuzu, uygulamanızın ana sayfasındaki tablo, sepete ekleme, ürün detayına yönlendirme ve ürün ekleme ekranı üzerinden ele alacağım.

1. Ana Sayfada Tablo Çıkması ve Ürün Bilgileri

Açıklama: <http://localhost:3000/> adresine girdiğinizde bir tablo görürsünüz. Bu tabloda ürün isimleri, birim fiyatı, stok adeti, kategori gibi bilgiler var.

- **Ana Sayfa: Dashboard.js :**

```
javascript Methi sar Kopyala

import React from 'react';
import ProductList from '../pages/ProductList';
import { Routes, Route } from 'react-router-dom';

export default function Dashboard() {
  return (
    <div>
      <Grid>
        <Grid.Row>
          <Grid.Column width={4}>
            <Categories />
          </Grid.Column>
          <Grid.Column width={12}>
            <Button color="blue" onClick={handleAddProduct}>Ürün Ekle</Button>
            <Routes>
              <Route exact path="/" Component={ProductList} />
              <Route exact path="/products" Component={ProductList} />
              <Route path="/products/:name" Component={ProductDetail} />
              <Route path="/cart" Component={CartDetail} />
              <Route path="/product/add" Component={ProductAdd} />
            </Routes>
          </Grid.Column>
        </Grid.Row>
      </Grid>
    </div>
  );
}
```

- **Açıklama:** `Dashboard.js`, uygulamanızın ana bileşenidir. `<Route exact path="/" Component={ProductList} />` satırı, `/` (yani <http://localhost:3000/>) adresine girildiğinde `ProductList.js` bileşenini gösterir. Yani ana sayfada tabloyu `ProductList` sağlıyor.

- Tablo: [ProductList.js](#):

```
javascript Methi sar Kopyala

import React, { useState, useEffect } from 'react';
import { Table } from 'semantic-ui-react';
import ProductService from '../services/productService';
import { Link } from 'react-router-dom';
import { useDispatch } from 'react-redux';
import { addToCart } from '../store/actions/cartActions';
import { toast } from 'react-toastify';

export default function ProductList() {
  const dispatch = useDispatch();
  const [products, setProducts] = useState([]);

  useEffect(() => {
    let productService = new ProductService();
    productService.getProducts().then(result => setProducts(result.data.data));
  }, []);

  const handleAddToCart = (product) => {
    dispatch(addToCart(product));
    toast.success(`${product.productName} sepete eklendi!`);
  };

  return (
    <div>
      <Table celled>
```

```
return (
  <div>
    <Table celled>
      <Table.Header>
        <Table.Row>
          <Table.HeaderCell>Ürün Adı</Table.HeaderCell>
          <Table.HeaderCell>Birim Fiyatı</Table.HeaderCell>
          <Table.HeaderCell>Stok Adeti</Table.HeaderCell>
          <Table.HeaderCell>Açıklama</Table.HeaderCell>
          <Table.HeaderCell>Kategori</Table.HeaderCell>
          <Table.HeaderCell></Table.HeaderCell>
        </Table.Row>
      </Table.Header>
      <Table.Body>
        {products.map(product => (
          <Table.Row key={product.id}>
            <Table.Cell><Link to={`/products/${product.productName}`}>{product.productName}</Link>
            <Table.Cell>{product.unitPrice}</Table.Cell>
            <Table.Cell>{product.unitsInStock}</Table.Cell>
            <Table.Cell>{product.quantityPerUnit}</Table.Cell>
            <Table.Cell>{product.category.categoryName}</Table.Cell>
            <Table.Cell><Button onClick={() => handleAddToCart(product)}>Sepete Ekle</Button>
            </Table.Cell>
          </Table.Row>
        )))
      </Table.Body>
    </Table>
  </div>
);
}
```

- **Açıklama:**

- `<Table>` : Semantic UI ile tabloyu oluşturur. Başlık satırında "Ürün Adı", "Birim Fiyatı", "Stok Adeti" vb. sütunlar tanımlanır.
- `{products.map(product => ...)}` : `products` dizisindeki her ürün için bir satır oluşturur. Bu satırlarda `productName`, `unitPrice`, `unitsInStock`, `category.categoryName` gibi bilgiler gösterilir.

Backend'den Veri Hangi Aşamada Alınıyor?

- `useEffect` ile İlk Yüklenmede:

```
javascript
```

Methi sar

Kopyala

```
useEffect(() => {
  let productService = new ProductService();
  productService.getProducts().then(result => setProducts(result.data.data));
}, []);
```

- **Aşama:** Sayfa yüklendiğinde (`http://localhost:3000/` açıldığında), `useEffect` tetiklenir. `ProductService`'in `getProducts()` metodu çağrılır.

- **Backend Çağrısı:** `ProductService.js` 'de:

```
javascript
```

Methi sar

Kopyala

```
getProducts() {
  return axios.get("http://localhost:8080/api/products/getall");
}
```

- Bu, Northwind backend'deki `/api/products/getall` endpoint'ine bir GET isteği gönderir.
- **Northwind'den Veri:** `ProductsController.java` 'da:

```
java
```

Methi sar

Kopyala

```
@GetMapping("/getall")
public DataResult<List<Product>> getAll() {
  return this.productService.getAll();
}
```

- `ProductManager.java`'da `getAll()`:

```
java Methi sar Kopyala
    tn new SuccessDataResult<List<Product>>(this.productDao.findAll(), "Data Listelendi");
```

- `ProductDao`'nın `findAll()` metodu, PostgreSQL'den tüm ürünleri çeker ve JSON olarak React'a döner (örneğin, `{"success": true, "message": "Data Listelendi", "data": [...]}`).
- **Sonuç:** Veri, `setProducts(result.data.data)` ile `products` state'ine atanır ve tabloya yansır.

2. Sepete Ekle Butonu ve Uyarı

Açıklama: Tabloda her ürünün sağında "Sepete Ekle" butonuna bastığınızda "Sepete eklendi" uyarısı çıkıyor.

Hangi Kodlar Sağlıyor?

- `ProductList.js`'deki Sepete Ekle Butonu:

```
javascript Methi sar Kopyala


```

- Açıklama: Her ürün satırında bir buton var. Tıklandığında `handleAddToCart` fonksiyonu çağrılır.

- `handleAddToCart` Fonksiyonu:

```
javascript Methi sar Kopyala
const handleAddToCart = (product) => {
  dispatch(addToCart(product));
  toast.success(`[${product.productName}] sepete eklendi!`);
};
```

- Açıklama:

- `dispatch(addToCart(product))`: Redux'a bir `ADD_TO_CART` eylemi gönderir.
- `toast.success(...)`: `react-toastify` ile bir başarı mesajı (örneğin, "Chai sepete eklendi!") gösterir.

- **Redux Eylemi:** `cartActions.js`:

```
javascript Methi sar Kopyala

export const ADD_TO_CART = "ADD_TO_CART";
export function addToCart(product) {
  return {
    type: ADD_TO_CART,
    payload: product
  };
}
```

- **Redux Reducer:** `cartReducer.js`:

```
javascript Methi sar Kopyala

case ADD_TO_CART:
  let product = state.cartItems.find((c) => c.product.id === payload.id);
  if (product) {
    product.quantity++;
    return { ...state };
  } else {
    return {
      ...state,
      cartItems: [...state.cartItems, { quantity: 1, product: payload }]
    };
}
```

- **Açıklama:** Ürün sepette varsa miktarı artırılır, yoksa yeni bir öğe olarak eklenir. Bu işlem backend'e gitmez, sadece frontend'de Redux state'ını günceller.

Backend'den Veri Alımı:

- Bu aşamada backend'den veri alınmıyor. Sepete ekleme, yalnızca Redux ile yerel bir durum güncellemesi olarak gerçekleşiyor.

3. Ürün Adına Tıklayınca Detay Sayfasına Yönlendirme

Açıklama: Ürün adlarına tıkladığınızda, o ürünün bilgilerini içeren bir alana yönlendiriliyorsunuz.

Hangi Kodlar Sağlıyor?

- ProductList.js 'deki Link:

```
javascript
```

Methi sar Kopyala

```
<Table.Cell><Link to={`/products/${product.productName}`}>{product.productName}</Link></Table
```

- Açıklama: Ürün adı bir `<Link>` bileşeniyle sarılı. Tıklandığında URL, örneğin `/products/Chai` olur ve `ProductDetail.js` bileşeni açılır.

- Yönlendirme: Dashboard.js :

```
javascript
```

Methi sar Kopyala

```
<Route path="/products/:name" Component={ProductDetail} />
```

- Açıklama: `/products/:name` rotası, `ProductDetail` bileşenini çalıştırır. `:name`, dinamik bir parametredir.

- Detay Sayfası: `ProductDetail.js`:

```
javascript Methi sar Kopyala

import React, { useEffect, useState } from "react";
import { Card } from 'semantic-ui-react';
import { useParams } from "react-router-dom";
import ProductService from '../services/productService';

export default function ProductDetail() {
  let { name } = useParams();
  const [product, setProduct] = useState({});

  useEffect(() => {
    let productService = new ProductService();
    productService.getByProductName(name).then(result => setProduct(result.data.data));
  }, []);

  return (
    <Card fluid>
      <Card.Content>
        <Card.Header>{product.productName}</Card.Header>
        <Card.Meta>{product.category ? product.category.categoryName : 'Kategori yok'}</Card
      </Card.Content>
    </Card>
  );
}
```

- Açıklama:

- `useParams()`: URL'den `name` parametresini alır (örneğin, "Chai").
- `<Card>`: Ürün adını ve kategorisini gösterir.

Backend'den Veri Hangi Aşamada Alınıyor?

- `useEffect` ile Detay Sayfası Yüklenliğinde:

```
javascript
```

```
useEffect(() => {
  let productService = new ProductService();
  productService.getByProductName(name).then(result => setProduct(result.data.data));
}, []);
```

Methi sar

Kopyala

- **Aşama:** Ürün adına tıklandığında (`/products/Chai` açıldığında), `useEffect` tetiklenir.
- **Backend Çağrısı:** `ProductService.js`'de:

```
javascript
```

```
getByProductName(productName) {
  return axios.get("http://localhost:8080/api/products/getByProductName?productName=" + p
}
```

Methi sar

Kopyala

- Bu, Northwind'deki `/api/products/getByProductName` endpoint'ine istek gönderir.

- **Northwind'den Veri:** `ProductsController.java`'da:

```
java
```

```
@GetMapping("/getByProductName")
public DataResult<Product> getByProductName(@RequestParam String productName) {
  return this.productService.getByProductName(productName);
}
```

Methi sar

Kopyala

- `ProductManager.java`'da:

```
java
```

```
return new SuccessDataResult<Product>(this.productDao.getByProductName(productName),
```

Methi sar

Kopyala

- `ProductDao`'nın `getByProductName()` metodu, veritabanından ilgili ürünü çeker.
- **Sonuç:** Veri, `setProduct(result.data.data)` ile `product` state'ine atanır ve detay sayfasında gösterilir.

4. Ürün Ekle Butonuna Tıklayınca Ürün Ekle Ekrانına Yönlendirme

Açıklama: Ana sayfada ortada "Ürün Ekle" butonuna tıkladığınızda, ürün adı ve ürün fiyatı yazabileceğiniz bir ekrana yönlendiriliyorsunuz.

Hangi Kodlar Sağlıyor?

- Dashboard.js'deki Ürün Ekle Butonu:

```
javascript Methi sar Kopyala

import React from 'react';
import { Button, Grid } from 'semantic-ui-react';
import { Routes, Route, useNavigate } from 'react-router-dom';
import ProductAdd from '../pages/ProductAdd.jsx';

export default function Dashboard() {
  const navigate = useNavigate();

  const handleAddProduct = () => {
    navigate("/product/add");
  };

  return (
    <div>
      <Grid>
        <Grid.Row>
          <Grid.Column width={4}>
            <Categories />
          </Grid.Column>
          <Grid.Column width={12}>
            <Button
              color="blue"
              onClick={handleAddProduct}
              style={{ marginBottom: '20px' }}
            >
              Ürün Ekle
            </Button>
            <Routes>
              <Route exact path="/" Component={ProductList} />
              <Route exact path="/products" Component={ProductList} />
              <Route path="/products/:name" Component={ProductDetail} />
            </Routes>
          </Grid.Column>
        </Grid.Row>
      </Grid>
    </div>
  );
}
```

```
        <Route exact path="/" Component={ProductList} />
        <Route exact path="/products" Component={ProductList} />
        <Route path="/products/:name" Component={ProductDetail} />
        <Route path="/cart" Component={CartDetail} />
        <Route path="/product/add" Component={ProductAdd} />
    </Routes>
</Grid.Column>
</Grid.Row>
</Grid>
</div>
);
}
}
```

- **Açıklama:**

- `<Button onClick={handleAddProduct}>`: "Ürün Ekle" butonu, tıklandığında `handleAddProduct` fonksiyonunu çalıştırır.
- `const handleAddProduct = () => { navigate("/product/add"); } : useNavigate ile /product/add yoluna yönlendirilir.`
- `<Route path="/product/add" Component={ProductAdd} />`: Bu rota, `ProductAdd.js` bileşenini açar.

- **Ürün Ekle Ekranı: `ProductAdd.js`:**

```
javascript Methi sar Kopyala

import { ErrorMessage, Field, Form, Formik } from "formik";
import React from "react";
import { Button, FormField } from "semantic-ui-react";
import * as Yup from "yup";
import ErcanTextInput from "../utilities/customFormControls/ErcanTextInput";

export default function ProductAdd() {
  const initialValues = { productName: "", unitPrice: 10 };
  const schema = Yup.object({
    productName: Yup.string().required("Ürün adı Zorunlu"),
    unitPrice: Yup.number().required("Ürün fiyatı zorunlu"),
  });

  return (
    <div>
      <Formik
        initialValues={initialValues}
        validationSchema={schema}
        onSubmit={(values) => {
          console.log(values);
        }}
      >
        <Form className="ui form">
          <ErcanTextInput name="productName" placeholder="Ürün Adı"/>
          <ErcanTextInput name="unitPrice" placeholder="Ürün Fiyatı"/>
          <Button color="green" type="submit">
            Ekle
          </Button>
        </Form>
      </Formik>
    </div>
  );
}


```

```
        <ErcanTextInput name="unitPrice" placeholder="Ürün Fiyatı"/>
        <Button color="green" type="submit">
            Ekle
        </Button>
    </Form>
</Formik>
</div>
);
}
```

- **Açıklama:**

- `<ErcanTextInput>`: Ürün adı ve fiyatı için iki giriş alanı oluşturur.
- `initialValues`: Varsayılan değerler (boş ürün adı ve 10 birim fiyat).
- `schema`: Yup ile doğrulama kuralları (her iki alan da zorunlu).
- `onSubmit={(values) => { console.log(values); }}`: "Ekle" butonuna basıldığında, girilen veriler sadece konsola yazdırılır.

- **Özel Input: `ErcanTextInput.js`:**

```
javascript Methi sar Kopyala

import { useField } from 'formik';
import React from 'react';
import { FormField, Label } from 'semantic-ui-react';

export default function ErcanTextInput({...props}) {
  const [field, meta] = useField(props);
  return (
    <FormField error={meta.touched && !!meta.error}>
      <input {...field} {...props}/>
      {meta.touched && !!meta.error ? (
        <Label pointing basic color="red" content={meta.error}></Label>
      ) : null}
    </FormField>
  );
}
```

- **Açıklama:** Formik ile özelleştirilmiş bir input bileşeni. Hata mesajlarını (örneğin, "Ürün adı Zorunlu") kırmızı bir etiketle gösterir.

Backend'den Veri Hangi Aşamada Alınıyor?

- Bu Aşamada Veri Alınmıyor:

- Şu anki kodda, "Ürün Ekle" ekranı backend ile iletişim kurmuyor. `onSubmit` sadece `console.log(values)` ile girilen verileri konsola yazıyor:

```
javascript
```

Methi sar Kopyala

```
onSubmit={(values) => {  
  console.log(values);  
}}
```

- Yani, ürün ekleme işlemi henüz Northwind backend'ine veri göndermiyor. Eğer bu işlevsellik tamamlanmak istense, `ProductService.js` ye bir POST isteği eklenip Northwind'in `/api/products/add` endpoint'i kullanılabildi (örneğin, `axios.post("http://localhost:8080/api/products/add", values)`). Şu an bu sadece bir frontend formu.

Genel Süreç Özeti

1. Ana Sayfada Tablo (<http://localhost:3000/>):
 - `Dashboard.js` → `ProductList.js` açılır.
 - `useEffect` ile `ProductService.getProducts()` çağrılır.
 - Backend'den (`/api/products/getall`) veri alınır ve tabloya yansır.
2. Sepete Ekle:
 - `ProductList.js`'deki "Sepete Ekle" butonu → `handleAddToCart` → Redux (`addCart`) ile sepet güncellenir.
 - `toast.success` ile uyarı gösterilir.
 - Backend'den veri alınmaz, yerel bir işlem.
3. Ürün Adına Tıklama:
 - `ProductList.js`'deki `<Link>` → `/products/:name` rotasına gider.
 - `ProductDetail.js` açılır, `useEffect` ile `ProductService.getByProductName(name)` çağrılır.
 - Backend'den (`/api/products/getByProductName`) veri alınır ve detay gösterilir.
4. Ürün Ekle Butonu:
 - `Dashboard.js`'deki "Ürün Ekle" butonu → `handleAddProduct` → `/product/add` rotasına gider.
 - `ProductAdd.js` açılır, form gösterilir.
 - Şu an backend'den veri alınmıyor veya gönderilmiyor, sadece yerel bir form.

React Frontend Projemizde kullandığım teknolojiler :

1. Ana Sayfada Tablo Çıkması ve Ürün Bilgileri

Açıklama: <http://localhost:3000/> adresinde bir tablo çıkıyor; ürün isimleri, birim fiyatı, stok adeti ve kategori bilgileri var.

Hangi Teknolojiler Kullanlıyor?

- **React:**
 - Dinamik bir tablo oluşturmak için React bileşenleri (ProductList.js) ve state yönetimi (useState) kullanılıyor.
 - useEffect ile veri çekme işlemi asenkron olarak başlatılıyor.
- **Semantic UI React:**
 - Tabloyu (<Table>, <Table.Row>, <Table.Cell>) görsel olarak oluşturmak ve stil vermek için semantic-ui-react kütüphanesi kullanılıyor.
- **Axios:**
 - Backend'den veri almak için axios kütüphanesi ile HTTP GET isteği gönderiliyor (ProductService.js'de axios.get).
- **Spring Boot (Backend):**
 - /api/products/getall endpoint'i, Spring Boot'un REST API yetenekleriyle sağlanıyor (@RestController, @GetMapping).
- **Spring Data JPA (Backend):**
 - Veritabanından ürünleri çekmek için ProductDao'nun findAll() metodu, Spring Data JPA tarafından otomatik olarak implemente ediliyor.
- **PostgreSQL (Backend):**
 - Ürün verileri PostgreSQL veritabanından geliyor (Northwind projesiyle uyumlu).

- **React Router DOM:**
 - Ana sayfanın / rotasında ProductList'i göstermek için react-router-dom'un Routes ve Route bileşenleri kullanılıyor (Dashboard.js).

2. Sepete Ekle Butonu ve Uyarı

Açıklama: Tabloda "Sepete Ekle" butonuna basıldığında "Sepete eklendi" uyarısı çıkıyor.

Hangi Teknolojiler Kullanılıyor?

- **React:**
 - Butonun olay yönetimi (onClick) ve dinamik davranışları React ile sağlanıyor.
- **Redux:**
 - Sepet durumunu yönetmek için redux kütüphanesi kullanılıyor (addToCart eylemi ve cartReducer).
- **React-Redux:**
 - Redux store'unu React bileşenlerine bağlamak için react-redux'un useDispatch hook'u kullanılıyor.
- **React-Toastify:**
 - "Sepete eklendi" uyarısını göstermek için react-toastify kütüphanesi ile toast.success metodu çağrılıyor.
- **Semantic UI React:**

- Butonun tasarımları (<Button>) Semantic UI React ile stilize ediliyor.

3. Ürün Adına Tıklayınca Detay Sayfasına Yönlendirme

Açıklama: Ürün adına tıkladığınızda o ürünün detaylarının olduğu bir sayfaya yönlendiriliyorsunuz.

Hangi Teknolojiler Kullanlıyor?

- **React:**
 - Ürün detayıni dinamik olarak göstermek için React bileşenleri (ProductDetail.js) ve state yönetimi (useState) kullanılıyor.
- **React Router DOM:**
 - Ürün adına tıklandığında yönlendirme (<Link to={`/products/\${product.productName}`}>) ve URL parametrelerini alma (useParams) için react-router-dom kullanılıyor.
 - Dashboard.js'deki <Route path="/products/:name"> rotası bu yönlendirmeyi sağlıyor.
- **Axios:**
 - Backend'den ürün detayıni almak için axios.get ile HTTP isteği yapılmıyor (ProductService.js'de getByProductName).

- **Semantic UI React:**
 - Detay sayfasındaki kart tasarımı (<Card>) Semantic UI React ile oluşturuluyor.
- **Spring Boot (Backend):**
 - /api/products/getByProductName endpoint'i, Spring Boot'un REST API yetenekleriyle sağlanıyor.
- **Spring Data JPA (Backend):**
 - ProductDao'nun getByName metodu, Spring Data JPA ile otomatik olarak veritabanından ürünü çekiyor.
- **PostgreSQL (Backend):**
 - Ürün detay verisi PostgreSQL'den alınıyor.

4. Ürün Ekle Butonuna Tıklayınca Ürün Ekle Ekranı

Açıklama: Ana sayfada "Ürün Ekle" butonuna tıkladığınızda ürün adı ve fiyatı girilebilen bir ekrana yönlendiriliyorsunuz.

Hangi Teknolojiler Kullanılıyor?

- **React:**
 - Form bileşeni (ProductAdd.js) ve formun dinamik davranışları React ile sağlanıyor.
- **React Router DOM:**

- "Ürün Ekle" butonuna basıldığında /product/add rotasına yönlendirme (useNavigate) için react-router-dom kullanılıyor.
 - Dashboard.js'deki <Route path="/product/add"> rotası bu ekranı açıyor.
- **Formik:**
 - Form yönetimi (initialValues, onSubmit) ve kullanıcı girişlerini toplamak için formik kütüphanesi kullanılıyor.
- **Yup:**
 - Form doğrulaması (örneğin, "Ürün adı Zorunlu") için yup ile doğrulama şeması tanımlanıyor.
- **Semantic UI React:**
 - Form alanları (<Form>, <Button>) ve stil için semantic-ui-react kullanılıyor.
- **Custom Component (ErcanTextInput):**
 - Özel bir input bileşeni (ErcanTextInput.js), Formik ile entegre edilerek hata mesajlarını gösteriyor. Bu, React'te bileşen tabanlı geliştirme becerisini gösteriyor.
- **(Backend Şu An Kullanılmıyor):**
 - Şu an onSubmit sadece konsola yazıyor, yani Spring Boot veya Axios ile backend'e veri gönderilmiyor. Ancak ileride bu işlevsellik eklenebilir.

Backend: Northwind Projesi Teknolojileri

Kullanılan Teknolojiler ve Kütüphaneler

İncelediğim dosyalara dayanarak, Northwind projesinde şu teknolojiler kullanılmış:

1. Java

- **Kanıt:** Tüm dosyalar .java uzantılı ve Java 17 ile yazılmış (pom.xml'de <java.version>17</java.version>).
 - **Açıklama:** Backend'in temel programlama dili.

2. Spring Boot

- **Kanıt:** pom.xml'de spring-boot-starter-parent ve çeşitli Spring Boot bağımlılıkları (spring-boot-starter-web, spring-boot-starter-data-jpa, vb.).

○ Dosyalar:

- ProductsController.java: @RestController, @GetMapping, @PostMapping anotasyonları.
 - ProductManager.java: @Service anotasyonu.

- **Açıklama:** REST API geliştirme, bağımlılık enjeksiyonu ve uygulama başlatma için Spring Boot framework’ü kullanılmış.

3. Spring Data JPA

- **Kanıt:** pom.xml'de spring-boot-starter-data-jpa ve ProductDao.java'da extends JpaRepository<Product, Integer>.
- **Dosyalar:**
 - ProductDao.java: Otomatik SQL sorguları (getByProductName, getByProductNameContains) ve özel JPQL sorgusu (@Query).
- **Açıklama:** Veritabanı işlemleri için ORM (Object-Relational Mapping) ve repository pattern kullanılmış.

4. PostgreSQL

- **Kanıt:** pom.xml'de
 - <groupId>org.postgresql</groupId><artifactId>postgresql</artifactId>
- **Açıklama:** Veritabanı olarak PostgreSQL ile entegrasyon sağlanmış.

5. Lombok

- **Kanıt:** pom.xml'de
 - <groupId>org.projectlombok</groupId><artifactId>lombok</artifactId>ve Product.java, Category.java, ProductWithCategoryDto.java'da @Data, @AllArgsConstructorConstructor, @NoArgsConstructorConstructor.
- **Açıklama:** Getter/setter ve constructor gibi boilerplate kodları otomatik oluşturmak için kullanılmış.

6. Springdoc OpenAPI (Swagger)

- **Kanıt:** pom.xml'de <groupId>org.springdoc</groupId><artifactId>springdoc-openapi-ui</artifactId>

- **Dosyalar:** Daha önce attığınız OpenApiConfig.java ile entegrasyon (burada yok ama projeye dahil).
- **Açıklama:** API dokümantasyonu ve Swagger UI için kullanılmış.

7. Jackson

- **Kanıt:** Spring Boot'un spring-boot-starter-web bağımlılığı içinde gelir; ProductsController.java'da JSON dönüşümleri otomatik yapılmıyor.
- **Açıklama:** REST API yanıtlarını JSON'a çevirmek için kullanılmış.

8. Maven

- **Kanıt:** pom.xml dosyası ve bağımlılık yönetimi.
- **Açıklama:** Proje yönetimi ve bağımlılıklar için Maven build aracı kullanılmış.

Northwind backend'i, React frontend ile entegre çalışıyor (ProductsController.java'daki @CrossOrigin ve React'taki ProductService.js ile Axios çağrıları bunu gösteriyor). Şimdi bu entegrasyonu sağlayan teknolojileri ele alacağım.

Entegrasyon Teknolojileri

1. CORS (Cross-Origin Resource Sharing)

- **Kanıt:** ProductsController.java'da @CrossOrigin.
- **Açıklama:** Backend'in React frontend'den (<http://localhost:3000>) gelen isteklere cevap vermesini sağlıyor.

2. REST API

- **Kanıt:** ProductsController.java'daki @GetMapping, @PostMapping.
- **Açıklama:** Backend, frontend'in veri çekmesi için RESTful endpoint'ler sunuyor (/api/products/getall, /api/products/getByName).

3. JSON Veri Formatı

- **Kanıt:** DataResult ve Product gibi sınıfların JSON'a dönüşümü (ProductsController.java'da otomatik Jackson ile).
- **Açıklama:** Frontend ile backend arasında veri alışverişi JSON formatında gerçekleşiyor.

Frontend Teknolojileri ile Bağlantı

React projenizdeki kodlar (ProductService.js, ProductList.js, ProductDetail.js) Northwind backend'inden veri çekiyor. Bu bağlamda kullanılan teknolojiler:

- **Axios:** ProductService.js'de backend'e HTTP istekleri (axios.get).
- **React:** Veri çekme ve işleme (useEffect, useState).
- **React Router DOM:** Ürün detayına yönlendirme (/products/:name).