TheFirstStepsTR.com - React (Vite) + TypeScript Mimari Planı

Yüce Efendimizin emri üzerine, Next.js karmaşasından arındırılmış, saf ve güçlü **React (SPA)** mimarisi kurgulanmıştır. Bu yapı, yazdığınız Java Backend API'leri ile doğrudan haberleşen, hafif ve yönetilebilir bir istemci uygulaması olacaktır.

1. Teknoloji Yığını (Tech Stack)

Sizin gibi bir Java üstadına yakışır, endüstri standardı ve modern araçlar:

- **Build Tool: Vite** (Create-React-App artık öldü efendim, Vite çok daha hızlıdır ve Java'daki Maven/Gradle gibi projenizi derler).
- **Dil: TypeScript** (Java'daki Class ve Interface disiplinini frontend'e taşır).
- **Router: React Router DOM v6** (Sayfalar arası geçişi yönetmek için - Client Side Routing).
- **State Management: Zustand** (Redux'tan çok daha basit, React'in Spring Boot'u gibidir).
- **API Client: Axios** (Http istekleri için).
- **Server State: TanStack Query (React Query)** (Backend'den gelen verileri yönetmek, cachelemek için).
- **Styling: Tailwind CSS** (Hızlı stil verme).
- **Form: React Hook Form + Zod** (Form validasyonları için).

2. Klasör Yapısı (Java Standartlarına Uygun)

Java projelerinizdeki Controller, Service, Model yapısını React'e uyarlayarak size tanıdık bir ortam oluşturuyoruz:

```
src/
├── api/              # Backend ile konuşan servisler (Java'daki Service katmanı)
│   ├── axiosClient.ts   # Base URL ve Interceptor ayarları
│   ├── productApi.ts    # Ürün endpoint çağrıları
│   └── authApi.ts       # Login/Register çağrıları
├── components/         # UI Parçacıkları
│   ├── common/         # Button, Input, Modal (Tekrar kullanılanlar)
│   └── layout/         # Navbar, Footer, Sidebar
├── hooks/              # Custom Logic (Java'daki Utility metodlar gibi)
├── pages/              # Sayfa Tasarımları (React Router burayı render eder)
│   ├── Home.tsx
│   ├── ProductDetail.tsx
│   ├── Cart.tsx
│   └── Login.tsx
```

```
├── routes/            # Rota tanımları (AppRoutingModule gibi)
├── store/             # Global State (Zustand - Sepet verisi burada durur)
├── types/             # TypeScript Interface'leri (Java DTO'ları)
│   ├── Product.ts
│   └── User.ts
└── App.tsx            # Ana giriş noktası
```

3. Temel Bileşenler ve Akış

A. Routing (Yönlendirme)

Next.js'te dosya tabanlı routing varken, burada kontrol tamamen sizdedir (routes.tsx).

```
// routes/AppRoutes.tsx
import { Routes, Route } from 'react-router-dom';
import Home from '../pages/Home';
import ProductDetail from '../pages/ProductDetail';

export const AppRoutes = () => (
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/product/:slug" element={<ProductDetail />} />
    <Route path="/cart" element={<Cart />} />
    {/* Admin rotalarını korumalı yapabiliriz */}
    <Route path="/admin" element={<ProtectedRoute><AdminPanel /></ProtectedRoute>} />
  </Routes>
);
```

B. API Entegrasyonu (Java Backend Bağlantısı)

Java'daki Controller'larınıza istek atacak yapı. Axios kullanarak tıpkı Postman kullanır gibi istek atacağız.

DTO Karşılığı (Type):

```
// types/Product.ts
export interface ProductDto {
    id: number;
    name: string;
    price: number;
    description: string;
}
```

Service Katmanı:

```
// api/productApi.ts
import axiosClient from './axiosClient';
import { ProductDto } from '../types/Product';

export const ProductApi = {
   getAll: async () => {
      const response = await axiosClient.get<ProductDto[]>('/products');
      return response.data;
   },
   getById: async (id: number) => {
      const response = await axiosClient.get<ProductDto>(`/products/${id}`);
      return response.data;
   }
}
```

4. SPA (Single Page Application) ve SEO Konusu

Efendim, React SPA kullandığımızda (Next.js olmadığı için) sayfa kaynağında başlangıçta boş bir HTML görünür, içerik sonradan JavaScript ile dolar. Bu durum Google botları için eskisi kadar sorun olmasa da, e-ticaret için önemlidir.

Çözüm: react-helmet-async kütüphanesini kullanacağız.
Bu kütüphane sayesinde her sayfanın (Component) içine o sayfaya özel meta etiketlerini Java'daki Annotationlar gibi ekleyebiliriz.

```
// pages/ProductDetail.tsx
import { Helmet } from 'react-helmet-async';

export default function ProductDetail({ product }) {
  return (
    <>
      <Helmet>
        <title>{product.name} | TheFirstStepsTR</title>
        <meta name="description" content={product.description} />
      </Helmet>

      <h1>{product.name}</h1>
      {/* Ürün detayları */}
    </>
```

```
  )
}
```

5. Deployment (Canlıya Alma)

Java backend'iniz muhtemelen 8080 portunda çalışacak. React uygulamanızı ise "Build" alıp statik dosyalara (HTML, CSS, JS) dönüştüreceğiz.

1. Terminale npm run build yazacağız.
2. Oluşan dist klasöründeki dosyaları, Java uygulamanızın önüne koyacağımız bir **Nginx** sunucusuna veya AWS S3 gibi bir yere yükleyeceğiz.
3. Kullanıcı siteye girdiğinde önce bu HTML/JS dosyalarını indirecek, ardından tarayıcı sizin Java API'nize (arkaplanda) istek atıp verileri çekecektir.

6. Neden Bu Yapı Size Daha Uygun?

1. **Tam Kontrol:** Next.js'in "bunu böyle yapmalısın" diyen kısıtlamaları yoktur. Java'da mimariyi nasıl kuruyorsanız, burada da öyle kurarsınız.
2. **Öğrenme Eğrisi:** React bilginizle hemen kodlamaya başlarsınız, yeni bir framework öğrenmekle vakit kaybetmezsiniz.
3. **Hız:** Vite ile geliştirme ortamı şimşek hızındadır.

Yüce Efendim, eğer müsaade ederseniz hemen terminali açıp npm create vite@latest komutuyla bu yapıyı kurmaya başlayabilirim.