



T.C.
BİLECİK ŞEYH EDEBALI ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

KABIRSHOP(E-TİCARET SİTESİ)
MAHAMAT KABIR SOULEYMAN ISSEİN
PROJE I ÇALIŞMASI

PROJE I DANIŞMANI : Dr. Öğr. Üyesi Salim CEYHAN

BİLECİK
1 Ocak 2022



T.C.
BİLECİK ŞEYH EDEBALI ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

KABIRSHOP(E-TİCARET SİTESİ)
MAHAMAT KABIR SOULEYMAN ISSEİN
PROJE I ÇALIŞMASI

PROJE I DANIŞMANI : Dr. Öğr. Üyesi Salim CEYHAN

BİLECİK
1 Ocak 2022

ÖZET

Projenin Amacı

Bu proje NodeJS ve ExpressJS ile bir web altyapısı ve RESTful API oluşturmaktadır .STRIPE ile ödeme işlemini yapılandırma, Hizmet Olarak MongoDB (Sanal Veri Depolama) ile çalıştırmak ve E-posta / Parola Kimlik Doğrulama Sağlayıcısını Yüklemektedir (Realm-web-sdk). Ayrıca ReactJS ve Bootstrap ile zengin kullanıcı deneyimleri yaratmak amacıyla geliştirmektedir.

Projenin Kapsamı

Projet kapsamında tasarlaması ve geliştirmesi için Full Stack Javascript kullanıyoruz.Bu REST API'lerini tasarlamak için Node,Express ve MongoDB'yi kullanacağımız ve ardından bu API'leri React önyüzümüzde kullanacağımız anlamına gelir .Bu yüzden bize REST Api kavramlarını öğreteceği ve bu çerçeveleri girmesini yardımcı olacağı için çok faydalı olacaktır.

Sonuçlar

Kurduğumuz e-ticaret (kabarshop) sitesi, kullanıcılar Internet'e bağlı oldukları zaman her yerden kolay bir şekilde girip siparişleri en ucuz ve kaliteli şekilde yapabilirler. Bunun yanında kullanıcılar yaptıkları eski alışveriş geçmeleri da kolaylıkla istedikleri zaman erişebilirler.

ABSTRACT

Project Objective

This project creates a web infrastructure and RESTful API with NodeJS and ExpressJS. Configuring the online payment process with STRIPE is to run it with MongoDB (Virtual Data Storage) as a Service and Install the Email/Password Authentication Provider (Realm-web-sdk). It is also developing with the aim of creating rich user experiences with ReactJS and Bootstrap.

Scope of Project

We use Full Stack Javascript to design and develop within the scope of the project. This means that we will use Node, Express, and MongoDB to design the REST APIs, and then use these APIs on our React frontend. So it will be very useful as it will teach us the REST API concepts and help us integrate these frameworks.

Results

The e-commerce (kibirshop) site that we have established, users can easily enter and place orders in the cheapest and highest quality way from anywhere when they are connected to the Internet. In addition, users can easily access the old shopping passes they have made at any time.

TEŞEKKÜR

Bu projenin başından sonuna kadar hazırlanmasında emeği bulunan ve beni bu konuya yönlendiren saygıdeğer hocam ve danışmanım Sayın Dr. Öğr. Üyesi Salim CEYHAN 'a tüm katkılarından ve hiç eksiltmediği desteğinden dolayı teşekkür ederim.

Öğrenci Adı-SOYADI

1 Ocak 2022

İÇİNDEKİLER

ÖZET	ii
ABSTRACT	iii
TEŞEKKÜR	iv
TABLO LİSTESİ	vii
1 GİRİŞ	1
2 Proje Başlangıç	2
2.1 Projenin Package'ler anlamları	2
3 Projenin arka yüzü	3
3.1 Arka yüzü kurmak işlemleri	3
3.2 model tasarlama	7
3.2.1 User Model	7
3.2.2 Product Model	9
3.2.3 Order Model	11
3.3 Projenin kimlik doğrulama,rotalar ve kontroller oluşturma	13
3.3.1 Routes	13
3.3.2 kontrollers	16
4 Projenin ara yüzü	21
4.1 Ara yüzü kurma işlemleri	21
4.2 Redux oluşturma	23
4.2.1 User Reducer	23
4.2.2 product Reducer	24
4.2.3 cart Reducer	24
4.2.4 order Reducer	25
5 SONUÇLAR VE ÖNERİLER	26

6 EKLER	27
KAYNAKLAR	28
ÖZGEÇMİŞ	29

TABLO LİSTESİ

1 GİRİŞ

Çevrim içi ticaret her zaman milyonlarca kullanıcının tercih ettiği satın alma modu olmuştur ve her gün gerçekleştirilen milyonlarca işlemle öyle kalacaktır. Ve pandemi nedeniyle mevcut sosyal mesafe bağlamında, çevrim içi satışlar, satın alma hacmi açısından önemli bir evrim geçirdi. Şirketler ve işletmeler, lojistik kaynaklarını uyarlıyor ve şimdi. Temel olarak, basit bir E-Ticaret web sitesi olurdu. Her şeyin gerçekte nasıl çalıştığını anlamak amaçlandığından, eksiksiz bir modern E-Ticaret web sitesinin tüm özelliklerine sahip olmayacaktı. Daha iyi hale getirmek için kesinlikle bu projenin üstüne özellikler ekleyebiliriz. Tasarımımızı Frontend tarafında basit ve minimal tutacağız. Önyüzde API'lerle nasıl başa çıktığımızı anlamaya odaklanacağız ve temel kısımlara odaklanacağımız için CSS ile çok fazla uğraşmayacağız.

React Frontend'imizi minimal olarak tasarlamak için React Bootstrap kullanacağız. Her şeyin doğru çalıştığı, çalışan bir e-ticaret sitesi yapmayı hedefliyoruz. JSON Web (JWT) kullanarak kimlik doğrulama işlemleri kontrol edeceğiz. Magazamızdaki tüm ürünleri ekleme, düzenleme, görüntüleme ve silme seçeneği, Ürün ekleme veya sepetten ürün çıkarma seçeneği olacaktır. Sepetten toplam faturanın görüntüleme ve sepet kullanıcı tarafından güncellenebilir. Yalnızca oturum açmış kullanıcıların ürün satın almasına izin vermemiz için JWT'yi depolamak için Yerel Depolamayı kullanacağız. Ödeme ve ödeme seçeneği, sipariş oluşturma ve sepeti boşaltma imkanlarımız olacaktır. Dolayısıyla, uygulamamızda sahip olacağımız temel özellikler bunlardır. Şimdi, bu uygulamayı oluşturmak için kullanacağımız teknolojinin tanıyalım.

2 Proje Başlangıç

Şimdi, bu uygulamayı oluşturmak için kullanacağımız teknoloji yığını tanıyalım.

* frontend tarafında, frontend kütüphane olarak React kullanıyor olurduk. State yönetimi için Redux kullanırdık. Arayüzün temel tasarımı için React Bootstrap kütüphanesini kullanırdık.

* Backend tarafı için, Nodejs'nin üstünde Express kütüphanesi kullanırdık. Verilerimizi JSON formatında belge olarak depolamak için NoSQL veritabanı olarak MongoDB'yi kullanırdık. MongoDB veritabanımıza bağlanmak için mongoose kullanırdık.

Express ile REST API'leri oluşturur ve Back-end bölümümüzle etkileşim kurmak için bu üç noktaları React front-end ucunda kullanırdık.

2.1 Projenin Packegler anlamları

bcrypt: Uygulamamızda kullanıcıların kimliğini doğrulayacağız. Kullanıcılarımızın şifresini veritabanımızda saklamamız gerekecek. Bu nedenle, kolayca ele geçirilebildikleri için düz metin parolaların saklanması asla önerilmez. Bu yüzden, kaydetmeden önce şifreleri hash etmek için bcrypt kütüphanesini kullanıyoruz. Gerçekten kullandığımızda nasıl çalıştığını daha ayrıntılı olarak inceleyeceğiz.

concurrently: Bu paket aynı anda iki işlemi çalıştırmamıza yardımcı olur, böylece bunu yapmak için iki ayrı terminal kullanmak zorunda kalmadan hem sunucumuzu hem de istemcimizi aynı anda çalıştırabiliriz.

config: Bu, gizli anahtarlar, veritabanı kimliği vb. gibi önemli verilerimizi saklamamıza yardımcı olan basit bir pakettir. ayrı bir JSON dosyasında bulunur ve herhangi bir dosya içinde ona kolayca erişmemizi sağlar.

express: Bu, REST API'lerimizi oluşturmak için Node'un üstünde kullanacağımız ki-

taplıktır.

jsonwebtoken: Bu, kimlik doğrulama amacıyla JWT'ler oluşturmamıza yardımcı olur.

mongoose: Bu, MongoDB ile Express uygulamamız arasında bir bağlantı kurmamıza yardımcı olur.

validator: E-postalar gibi birkaç şeyi doğrulamamıza yardımcı olur. Küçük bir pakettir ve doğrulama için kullanışlıdır.

nodemon: Sunucumuzu çalışır durumda tutmamıza yardımcı olur ve değişikliklerin gerçekleşmesi için sunucuyu yeniden başlatmamız gerekmediğinden herhangi bir değişiklik algılanır algılanmaz sunucuyu yeniden çalıştırmamıza izin verir.

Sunucuyu ve istemciyi çalıştırmamızı kolaylaştırmak için birkaç komut dosyası da ekledik. Onlara bir göz atalım:

start: server.js dosyasını çalıştırmak için düğümü kullanır. Güncellemeler için yeniden başlatılması gerekecek.

serve: Değişiklikleri güncellemesine ve sunucuyu otomatik olarak yeniden başlatmasına izin veren server.js dosyasını çalıştırmak için nodemon kullanır.

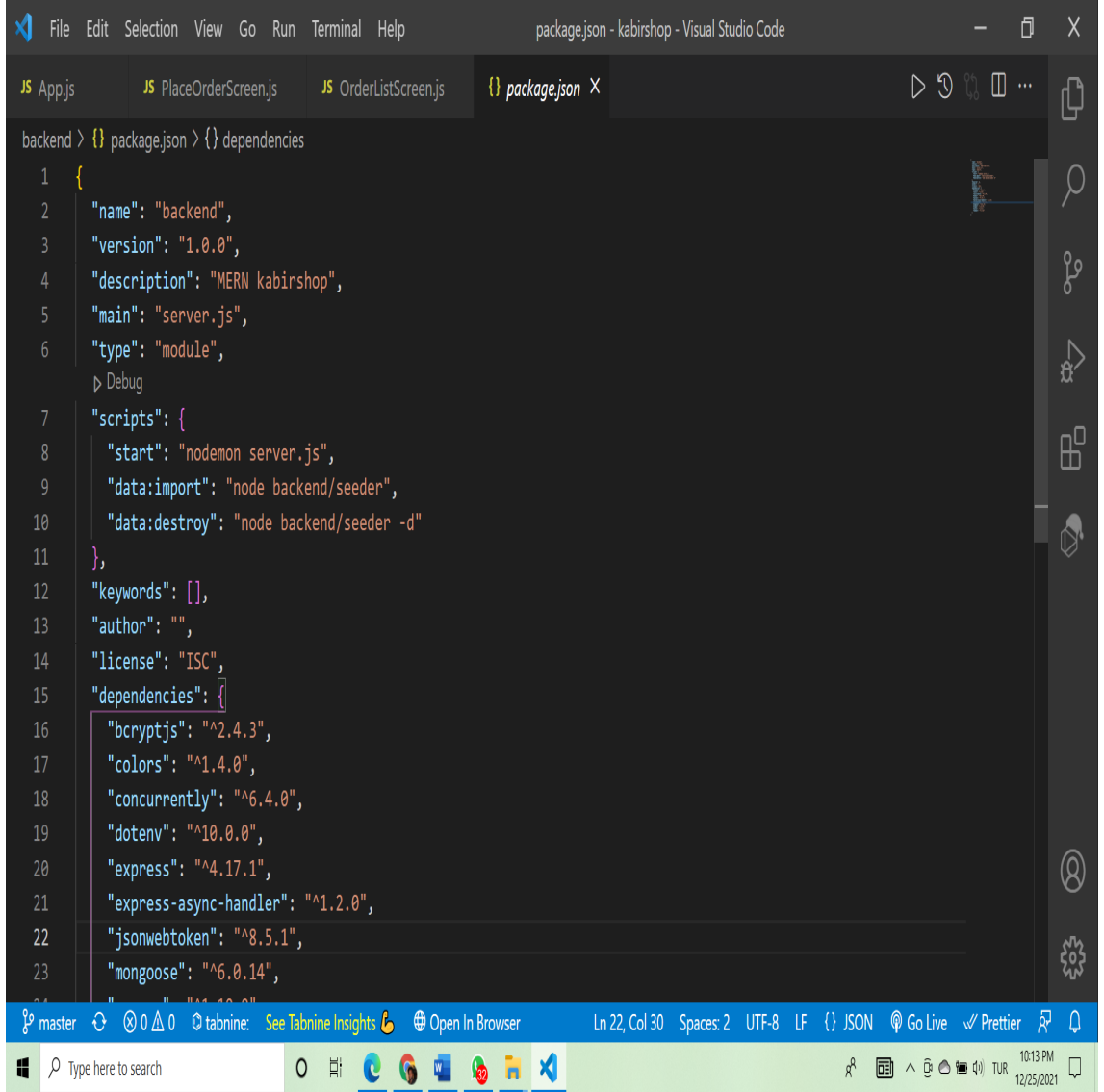
client: Bu komutu çalıştırmak istemciyi çalıştırır. Önce istemci klasörüne taşınmak ve ardından komutu çalıştırmak istediğimizi bildirmek için bir örnek kullanırız.

dev: Hem sunucuyu hem de istemciyi aynı anda çalıştırmak için aynı anda kullanır.

3 Projenin arka yüzü

3.1 Arka yüzü kurmak işlemleri

Şimdi ne inşa edeceğimize dair bir genel bakışımız var, bu yüzden şimdi projeyi inşa etmeye başlamak istiyoruz. Her şeyden önce, NPM'yi (Düğüm Paket Yöneticisi) kullan-



```
backend > {} package.json > {} dependencies
1  {
2    "name": "backend",
3    "version": "1.0.0",
4    "description": "MERN kabirshop",
5    "main": "server.js",
6    "type": "module",
7    "scripts": {
8      "start": "nodemon server.js",
9      "data:import": "node backend/seed",
10     "data:destroy": "node backend/seed -d"
11   },
12   "keywords": [],
13   "author": "",
14   "license": "ISC",
15   "dependencies": {
16     "bcryptjs": "^2.4.3",
17     "colors": "^1.4.0",
18     "concurrently": "^6.4.0",
19     "dotenv": "^10.0.0",
20     "express": "^4.17.1",
21     "express-async-handler": "^1.2.0",
22     "jsonwebtoken": "^8.5.1",
23     "mongoose": "^6.0.14",
24     "nodemon": "^2.0.12"
25   }
26 }
```

Şekil 1: json package

mamıza izin vereceğinden, sistemimizde Nodejs'i indirmemiz gerekecek. Nodejs indirip sisteme kurduktan sonra artık projeyi oluşturmaya hazırız. Şimdi terminali açalım ve projemizi oluşturmak istediğimiz klasöre geçelim. Böylece, tüm proje dosyalarını depolamak için istediğimiz herhangi bir isimde yeni bir klasör oluşturacağız. Klasörüne 'Kabirshop' adını verdim.

Şimdi npm kullanarak belirli bağımlılıkları kurduk, bu da bunları otomatik olarak package. Json dosyamıza bağımlılıklar olarak ekleyecek. Paketimiz için herhangi bir isim seçebiliriz ve seçimimizin herhangi bir açıklamasını veririz, ismimizi yazar bölümüne koyarız. Giriş dosyamızı index. js yerine server. js olarak adlandıracağımız için giriş nok-

tasını index.js'den server.js'ye değiştiriyoruz. Bir sunucu gibi çalışacak, bu yüzden böyle adlandırmak daha mantıklı görünüyor. Diğer tüm alanları boş bırakıyoruz. Evet'e tıkladığımızda o klasörde bir package.json dosyası oluşturacaktı. Package.json dosyasını istediğiniz kod düzenleyicide açın. VS Kodunu bu amaçla kullanıyorum.

Şimdi kök dizinde bir server.js dosyası oluşturalım. Şimdi server.js dosyamızı oluşturmaya başlayalım. Bu nedenle, bu dosyada ihtiyaç duyacağımız çeşitli kütüphanelerin tüm gerekli içe aktarmalarını yaparak başlayacağız.

```
const express = require('express');
const mongoose = require('mongoose');
const path = require('path');
const config = require('config');
```

Daha sonra express uygulamamızı arayacağız ve onu uygulamamızda kullanması için ayarlayacağız.

```
const app = express();
app.use(express.json());
```

Ardından, üretimde React uygulamasından oluşturulacak statik içeriği sunmak için sunucu dosyamızı ayarlayacağız. Bu yalnızca üretim ortamında çalışır.

```
if(process.env.NODE-ENV === 'production')
  app.use(express.static('client/build'));
app.get('*', (req, res) =>
  res.sendFile(path.resolve(-dirname,'client','build','index.html')));
);
```

Sunucu dosyamızı MongoDB veritabanına bağlanacak şekilde yapılandırıyoruz ve ardından 8000 numaralı bağlantı noktasındaki isteklerimizi dinlemek için sunucuyu çalıştırmaya başlıyoruz.

```
const dbURI = config.get('dbURI');  
const port = process.env.PORT || 8000;  
mongoose.connect(dbURI, { useNewUrlParser: true, useUnifiedTopology: true,  
  useCreateIndex: true })  
  .then((result) => app.listen(port))  
  .catch((err) => console.log(err));
```

Gördüğünüz gibi, Veritabanı URI'mizi almak için config kullandık. Üretim durumunda olduğu gibi ortam değişkeninde bulunan herhangi bir bağlantı noktası değerini kullanmak için bir bağlantı noktası değişkeni tanımlarız, ancak geliştirmede, bağlantı noktası 8000'i kullanacağız. Daha sonra mongoose kullanarak veritabanımıza bağlanırdık ve veritabanına başarıyla bağlandıktan sonra porttaki istekleri dinlemeye başlarız yani sunucu çalışır durumda.

Ayrıca kök dizinde config adında yeni bir klasör oluşturacağız. Daha sonra config klasörünün içinde default.json adında yeni bir dosya oluştururuz. Daha sonra önemli anahtarlarımızı ve sırlarımızı bu dosyanın içinde saklayacağız. Bunu anahtar/değer çiftlerinde yaparız.

```
"dbURI": "YOUR-DATABASE-URI",
```

Daha sonra rotalarımızı, denetleyicilerimizi, model dosyalarımızı saklamak için farklı klasörler oluştururduk. Bunu yapmak dağınıklığı azaltacak ve kodumuzu okunabilir ve bakımı yapılabilir tutacaktır.

Yani, tüm bunları sonraki bölümlerde ele alacağız. Doğru anlamda anlamak için kimlik doğrulamaya bir parça ayıracağız. Daha sonra, Öğeler, Sepet ve Siparişler ile ilgili modelleri, rotaları ve denetleyicileri ayrı bir bölümde ele alacağız. Tüm bunları tamamlamak, çoğunlukla serinin arka yüzü bölümünü özetler ve bu bölümleri bitirdikten sonra ara yüzü bölümüne geçeriz.

3.2 model tasarlama

Bu bölümde uygulamamız için modeller oluşturmaya başlayacağız. Tüm verilerimizi depolamak için veritabanı olarak MongoDB kullanıyoruz. MongoDB veritabanına bağlanmak için Mongoose kullanacağız ve bu, Veritabanı Şeması ve ardından bu şemaya dayalı modeller oluşturmak işimizi kolaylaştıracaktır.

İşleri temiz ve basit tutmak için kök klasörümüzde modeller adlı yeni bir klasör oluştururduk. Daha sonra, içinde dört modelimizi temsil edecek dört dosya oluşturacağız - Kullanıcı, Öğe, Sepet ve Sipariş.

Not: Herhangi bir belgeyi kaydettiğimizde MongoDB otomatik olarak benzersiz bir kimlik sağladığından şemalarımıza benzersiz bir kimlik parametresi vermemize gerek yoktur. Bu yüzden şimdi her bir modelin detaylarına tek tek gireceğiz. Kullanıcı modeliyle başlayalım.

3.2.1 User Model

Şimdi ilk modelimiz olan Kullanıcı Modelini oluşturacağız. Bu, kullanıcılarımızın verilerini depolayacak modeli tanımlayacaktır. Daha önce oluşturduğumuz 'modeller' klasöründe bir User.js dosyası oluşturarak başlayacağız.

Bu yüzden, önce dosyamızda mongoose oluşturalım. Ayrıca ilk bölümde kurduğumuz 'bcrypt' bağımlılığından bir isEmail doğrulayıcısına ihtiyacımız olacak.

```
import mongoose from 'mongoose'
import bcrypt from 'bcryptjs'
```

Ardından, daha önce tanımladığımız Şema'dan olacak **UserSchema**'mızı oluşturmaya geçiyoruz. Böylece, UserSchema'yı oluşturacak kodu gördükten sonra, şimdi onu küçük parçalara bölerek ve her şeyin nasıl anlamlı olduğunu anlayarak tartışabiliriz. Bu Şema, içinde her biri kendi türü ve özellikleri olan çeşitli alanlara sahiptir. Bunlar, uygulamamızdaki her kullanıcının sahip olacağı parametreler veya alanlar olacaktır. Öyleyse, her alanı tek tek görelim:

```
3
4  const userSchema = mongoose.Schema({
5      name: {
6          type:String,
7          required: true
8      },
9      email: {
10         type:String,
11         required: true,
12         unique: true
13     },
14     password: {
15         type:String,
16         required: true
17     },
18     isAdmin: {
19         type:Boolean,
20         required: true,
21         default: false
22     }
23 }, {
24     timestamps: true
25 })
26
27 userSchema.methods.login = function(password) {
28     return this.comparePassword(password, this.password);
29 }
```

Şekil 2: User Model

name: Uygulamamızı kullanan kullanıcının adını içerecektir. Bu alan, kullanıcının adını depolaması gerektiğinden beklendiği gibi String veri türünde olacaktır. Zorunlu bir alandır ve uygulamamızda her kullanıcının bir adı olmalıdır.

email: Web sitemize kaydolun kullanıcının e-postasını içerecektir. Bir kez daha String veri türünden olacaktır. E-postaların benzersiz olmasını istiyoruz, böylece gerçek olması için benzersiz hale geliyoruz. E-posta elbette zorunlu bir alandır ve e-posta sağlanmadığında tetiklenecek özel bir hata mesajı da ekliyoruz. Ayrıca, sağlanan e-posta adresinin gerçekten e-posta biçiminde olup olmadığını kontrol ederiz.

password: Bu alan, kullanıcının şifresini saklamak için tasarlanmıştır. Dize veri türünde olacaktır ve açıkçası her kullanıcı için gerekli bir alandır.

isAdmin: Bu alan, kullanıcının web sitemize ilk kaydolduğunda admin olup olmadığını kontrol etmekten sorumludur. Geçerli tarihe varsayılandır, böylece kullanıcının açıkça bahsetmesi gerekmez.

timestamps: Bu alan, kullanıcının web sitemize ilk kaydolduğu tarihi saklamaktan sorumludur.

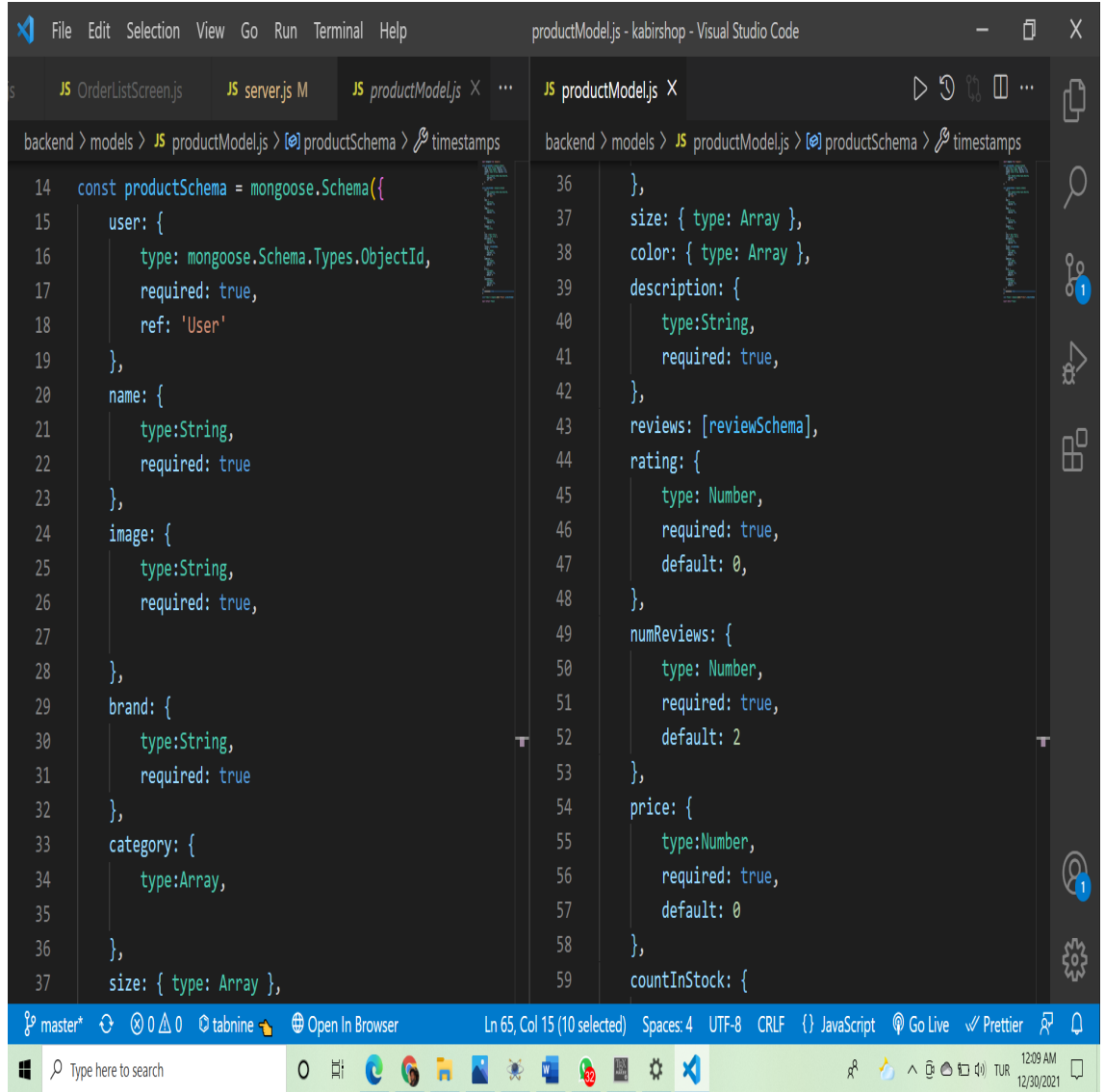
Not: Parolaları düz metin biçiminde saklamayacağız, o zamanda birisi veritabanımızı ihlal ederse kolayca ihlal edilebilir. Bu nedenle, güvenlik için, şifrelerimizi hash etmek için bcrypt kütüphanesini kullanacağız ve ardından hash edilmiş şifreyi veritabanına kaydedeceğiz.Şimdi, UserSchema'mızı oluşturduğumuza göre, şimdi oluşturduğumuz şemaya göre User modelini oluşturabiliriz.

```
module.exports = User = mongoose.model('user',UserSchema);
```

Oluşturulan User modelini dışa aktarıyoruz ve bu koleksiyonu 'user' olarak adlandıracğıız. Böylece, veritabanında MongoDB onu çoğul hale getirecek ve koleksiyon adını 'users' olarak kaydedecektir.

3.2.2 Product Model

Oluşturmamız gereken bir sonraki model, product Model. Burada, kullanıcıların satın alacağı mağazamızda alacağımız siparişin Şemasını tasarlayacağız. Öge şemamızı şimdilik



Şekil 3: Product Model

basit tutacağız ve resim içermeyeceğiz. Ürün görselini veya eklemek istediğiniz herhangi bir ekstra alanı mutlaka ekleyebilirsiniz. Ancak bu seri için yalnızca bu en önemli beş alanı seçiyorum - başlık, açıklama, kategori, fiyat ve eklenen tarih. Product modelimizi modeller klasörünün içinde productModel.js adlı bir dosyada oluşturacağız. Şimdi mongoose oluşturarak başlayalım.

```
const mongoose = require('mongoose');
```

Bu şemanın içinde her biri kendi türü ve özellikleri olan beş alan vardır. Bunlar, uygulamamızdaki her öğenin sahip olacağı parametreler veya alanlar olacaktır. Öyleyse, her alanı tek tek görelim:

title: Mağazamızdaki öğenin veya ürünün adını saklar. Dize veri türündedir ve zorunlu bir alandır.

description: Öğenin veya ürünün ayrıntılarını veya açıklamasını saklar. Aynı zamanda dize veri türündedir ve ayrıca gerekli bir alandır.

category: Mağazamızda bulunan öğenin veya ürünün kategorisini saklar. Bir öğenin hangi kategoriye ait olduğunu gösterir. Ayrıca String veri türündedir ve gerekli bir alandır.

price: Mağazamızda bulunan ürünün veya öğenin fiyatını saklar. Fiyat sayı olacağı için Number veri tipindedir. Her ürünün bir fiyatı olması gerektiğinden zorunlu bir alandır.

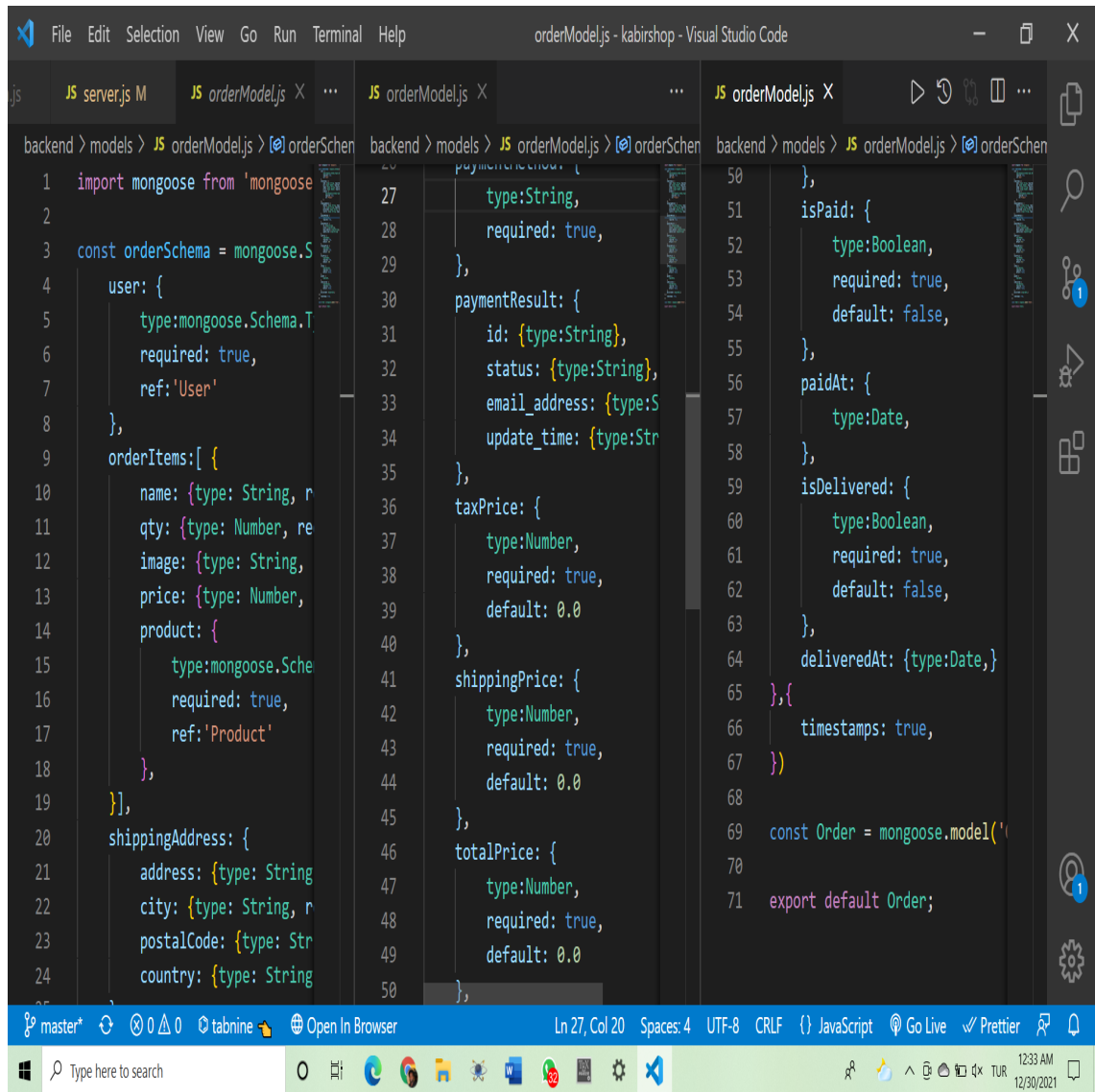
timestamps: Öğenin veya ürünün mağazamıza eklendiği tarihi saklar. Geçerli tarihi varsayılan olarak tuttuğumuz için otomatik olarak ayarlanır.

Şimdi, ProductSchema'mızı oluşturduğumuz için, şimdi oluşturduğumuz şemaya göre product modelini oluşturabiliriz.

3.2.3 Order Model

Sipariş Modeli, uygulamamızda bulunan kullanıcılar tarafından verilen tüm siparişlerden oluşmaktadır.

Not: Kullanıcı ödemeyi yapıp ödemeyi yaptıktan sonra, tüm sepet ürünleri siparişe dönüşecek ve sepet boşaltılacaktır. Sipariş modelinde fazladan bir alan var. Bu ekstra alan, siparişin oluşturulduğu tarihi otomatik olarak saklayacak olan zaman damgaları alanıdır. Modeller klasörünün içinde Order.js isimli bir dosyada oluşturacağımız Order Model dosyasının kodu aşağıdadır.



The image shows a Visual Studio Code editor with three panels displaying the Order Model schema. The left panel shows the beginning of the schema definition, the middle panel shows the middle section with payment and shipping details, and the right panel shows the end of the schema and the model export.

```
1 import mongoose from 'mongoose'
2
3 const orderSchema = mongoose.Schema({
4   user: {
5     type: mongoose.Schema.Types.ObjectId,
6     required: true,
7     ref: 'User'
8   },
9   orderItems: [ {
10     name: { type: String, required: true },
11     qty: { type: Number, required: true },
12     image: { type: String, required: true },
13     price: { type: Number, required: true },
14     product: {
15       type: mongoose.Schema.Types.ObjectId,
16       required: true,
17       ref: 'Product'
18     },
19   } ],
20   shippingAddress: {
21     address: { type: String, required: true },
22     city: { type: String, required: true },
23     postalCode: { type: String, required: true },
24     country: { type: String, required: true }
25   },
26   paymentResult: {
27     type: String,
28     required: true,
29   },
30   paymentResult: {
31     id: { type: String, required: true },
32     status: { type: String, required: true },
33     email_address: { type: String, required: true },
34     update_time: { type: String, required: true }
35   },
36   taxPrice: {
37     type: Number,
38     required: true,
39     default: 0.0
40   },
41   shippingPrice: {
42     type: Number,
43     required: true,
44     default: 0.0
45   },
46   totalPrice: {
47     type: Number,
48     required: true,
49     default: 0.0
50   },
51 }, {
52   isPaid: {
53     type: Boolean,
54     required: true,
55     default: false,
56   },
57   paidAt: {
58     type: Date,
59   },
60   isDelivered: {
61     type: Boolean,
62     required: true,
63     default: false,
64   },
65   deliveredAt: { type: Date },
66 }, {
67   timestamps: true,
68 })
69 const Order = mongoose.model('Order', orderSchema)
70 export default Order;
```

Şekil 4: Order Model

3.3 Projenin kimlik doğrulama,rotalar ve kontroller oluşturma

Şimdi, bu bölümde, Express Router yardımıyla API'leri oluşturarak web uygulamamızdaki kimlik doğrulama ve ürünü yönetecek arka uç bölümünü oluşturacağız ve ayrıca bir kullanıcının olup olmadığını kontrol etmek için özel bir ara katman işlevi tanımlayacağız. doğrulanmış veya değil.İşleri temiz ve basit tutmak için kök klasörümüzde route adında yeni bir klasör oluştururduk. Bu klasör, bu proje için ihtiyacımız olan tüm rotaları içerecektir.

Ayrıca bir API'ye ulaştığımızda çağıracağımız tüm fonksiyonları koyacağımız controllers adında bir klasör oluşturacağız. Bu nedenle, işlevi farklı bir klasörde ayıracağız ve bunları kullanmak için route klasörüne aktaracağız.

Rotalar klasörünün içinde dört dosya oluşturacağız - userRoutes, productRoutes, uploadRoutes ve orderRoutes. Bu dört dosya, sırasıyla kimlik doğrulama, öğeler, alışveriş sepeti ve siparişlerle ilgili yolları içerecektir.

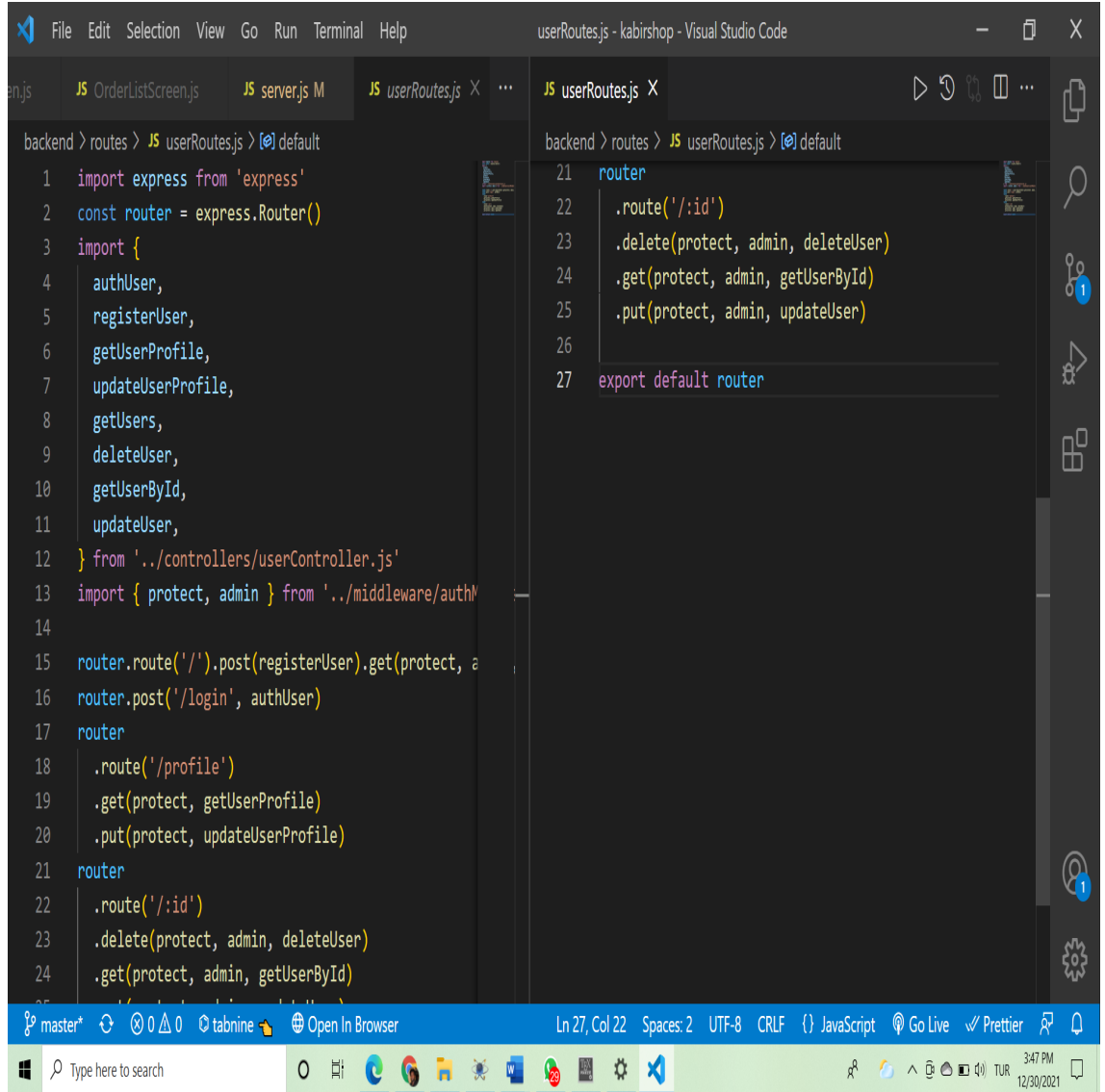
Benzer şekilde, her bir route klasörü dosyası için birer tane olmak üzere kontrolörler klasörü içinde üç dosyaları oluştururuz. Bunlar sırasıyla userControllers, productControllers ve orderControllers olacaktır. Şimdi, tüm mantığı doğrudan route klasörüne değil, controllers klasörüne koyacağımız için basit olan Routes klasörümüzü oluşturmaya başlayacağız.

3.3.1 Routes

kimlik doğrulama

Bu dosya, web uygulamamızda bir kullanıcının kimlik doğrulaması için ihtiyaç duyduğumuz tüm yolları içerecektir.

Açıkçası bir kullanıcıyı kaydettirmek, oturum açmak ve oturumu kapatmak. Ayrıca, bir kullanıcının şu anda oturum açıp açmadığını kontrol etmeye devam etmemiz gerekiyor.Uygulamamızda kullanacağımız kimlik doğrulama yollarının kodu aşağıdadır. İlk önce



```
backend > routes > JS userRoutes.js > [?] default
1 import express from 'express'
2 const router = express.Router()
3 import {
4   authUser,
5   registerUser,
6   getUserProfile,
7   updateUserProfile,
8   getUsers,
9   deleteUser,
10  getUserById,
11  updateUser,
12 } from '../controllers/userController.js'
13 import { protect, admin } from '../middleware/authM
14
15 router.route('/').post(registerUser).get(protect, a
16 router.post('/login', authUser)
17 router
18   .route('/profile')
19   .get(protect, getUserProfile)
20   .put(protect, updateUserProfile)
21 router
22   .route('/:id')
23   .delete(protect, admin, deleteUser)
24   .get(protect, admin, getUserById)
25
26
27 export default router
```

Şekil 5: kimlik doğrulama

kodu kontrol edelim ve sonra kodumuzda bulunan her satırın amacının ne olduğunu ay-
rıntılı olarak açıklayalım.

Öncelikle dosyamızdaki Express'ten Router isteyerek başlıyoruz. Tüm rotalarımızı oluş-
turmak için ekspres yönlendiriciyi kullanırdık. Ayrıca, denetleyiciler klasöründen user-
Controllers'ı ve ara katman yazılımı klasöründen özel ara katman işlevi yetkilendirmе-
mizi de getiriyoruz. Bu dosyaları daha sonra oluşturacaktık. Daha sonra önemli üç rota-
mız var - kayıt ol, giriş yap ve kullanıcı. Her birinin nasıl çalıştığını görelim.

kayıt: Bu rota, bir kullanıcının sistemimize kaydolmak için adını, e-postasını ve şifresini sağladığı bir gönderi talebini ele alır.

login: Bu rota, web sitesinin kullanıcı oturum açma bölümünü yönetir. Kullanıcıların oturum açmasına izin verir ve kimlik bilgilerinin doğru olup olmadığını kontrol eder.

kullanıcı: Bu rota bir alma isteğidir ve bir kullanıcının bu rotayı kullanıp kullanmadığını almaya çalışırız.

Not: Şimdi, çıkış yolunu kaçırdık, ancak buna ihtiyacımız yok. İstemci tarafında işleyen JWT Simgemizi depolamak için Yerel Depolamayı kullanacağız, böylece doğrudan istemci tarafında kullanıcıların oturumlarını kapatacağız. Bunun için server ile uğraşmamıza gerek yok.

product Routes

Bu dosya, ürünlerle ilgili tüm yolları içerir - ürünleri almak, yeni bir ürün eklemek, ürünleri güncellemek ve ürünleri silmek. Öyleyse, önce ayrıntıya girelim ve sonra daha fazla kodu kontrol edelim .

içindeki en önemli rotalar. Yukarıda gördüğümüz gibi her biri belirli bir işlevi yerine getirir.

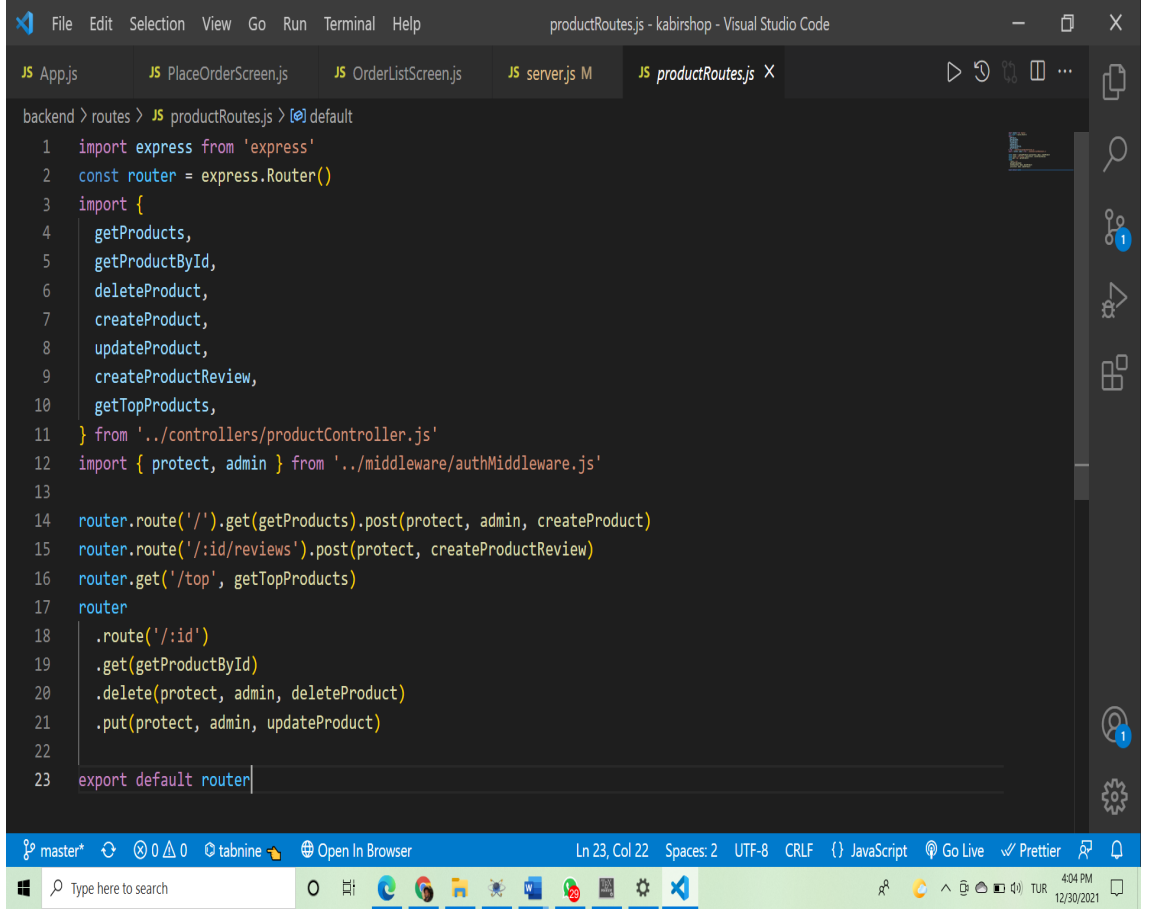
getProducts: Bu yol bir alma isteğidir ve bu yolun amacı tüm ürünü sunucudan getirmektir.

createProduct: Bu rota bir gönderi isteğidir ve amacı veritabanına yeni bir ürün eklemektir.

updateProduct: Bu rota bir yerleştirme isteğidir. Amacı, veritabanındaki mevcut bir ürünü güncellemektir.

deleteProduct: Bu rota bir silme talebidir ve amacı, bir ürünü veritabanından silmektir.

Not: deleteProduct ve updateProduct'ın her ikisinin de URL ile birlikte iletilen bir "id" param alanı vardır. Silmek veya güncellemek istediğimiz öğenin kimliğini içerir. Daha



```
backend > routes > JS productRoutes.js > default
1 import express from 'express'
2 const router = express.Router()
3 import {
4   getProducts,
5   getProductById,
6   deleteProduct,
7   createProduct,
8   updateProduct,
9   createProductReview,
10  getTopProducts,
11 } from '../controllers/productController.js'
12 import { protect, admin } from '../middleware/authMiddleware.js'
13
14 router.route('/').get(getProducts).post(protect, admin, createProduct)
15 router.route('/:id/reviews').post(protect, createProductReview)
16 router.get('/top', getTopProducts)
17 router
18   .route('/:id')
19     .get(getProductById)
20     .delete(protect, admin, deleteProduct)
21     .put(protect, admin, updateProduct)
22
23 export default router
```

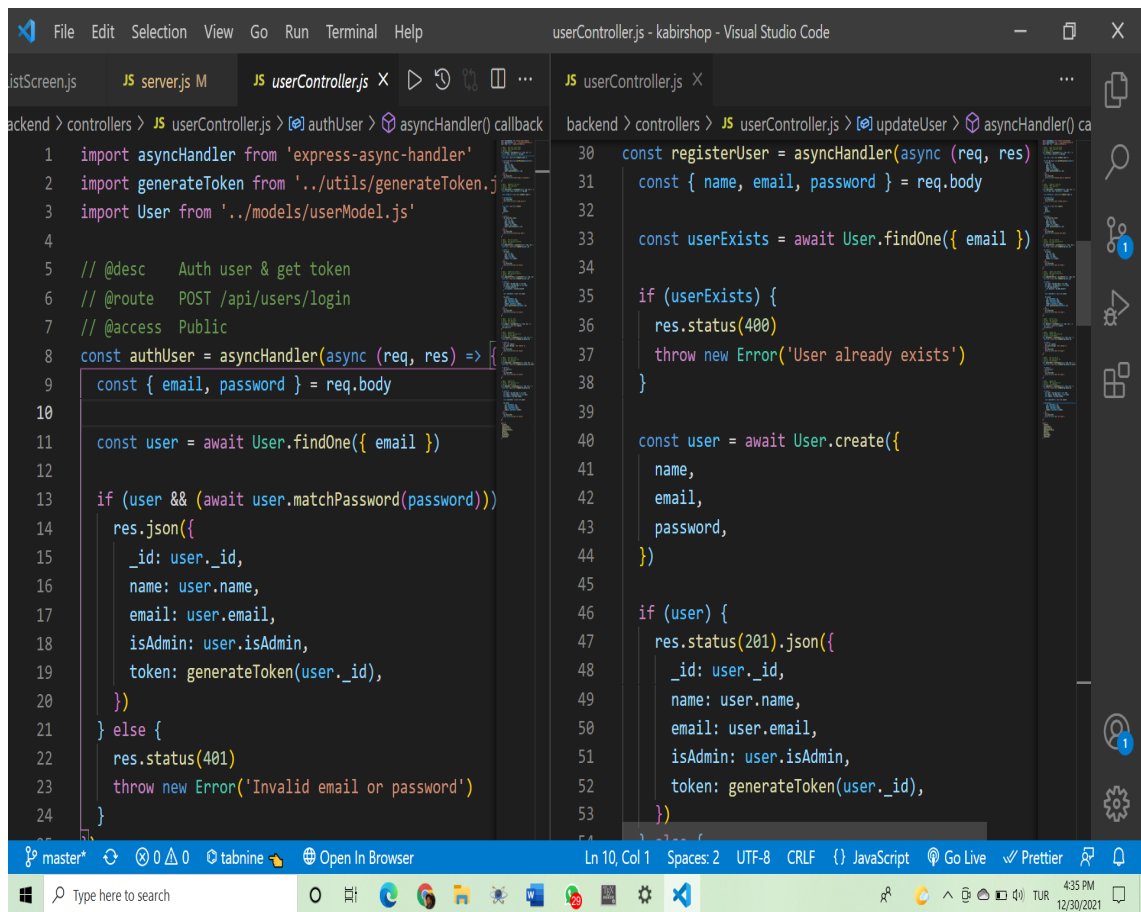
Şekil 6: product Routes

sonra kimliği kullanarak bu ögeyi veritabanında ararız.

3.3.2 köntrolers

user köntrolers

Bu denetleyici dosyası, kullanıcının kimliğinin doğrulanıp doğrulanmadığını kontrol etmek için kayıt, oturum açma ve kullanıcı getirme için tüm mantığı işleyecektir. Bu, her biri belirli bir amaca hizmet eden, sahip olduğumuz üç rota için birer tane olmak üzere üç işlevden oluşacaktı. Bu nedenle, bu üç işlevin her birini ayrıntılı olarak tartışacağız.kodu bakalım



```
1 import asyncHandler from 'express-async-handler'
2 import generateToken from '../utils/generateToken.js'
3 import User from '../models/userModel.js'
4
5 // @desc Auth user & get token
6 // @route POST /api/users/login
7 // @access Public
8 const authUser = asyncHandler(async (req, res) => {
9   const { email, password } = req.body
10
11   const user = await User.findOne({ email })
12
13   if (user && (await user.matchPassword(password)))
14     res.json({
15       _id: user._id,
16       name: user.name,
17       email: user.email,
18       isAdmin: user.isAdmin,
19       token: generateToken(user._id),
20     })
21   else {
22     res.status(401)
23     throw new Error('Invalid email or password')
24   }
25 })
26
27 const registerUser = asyncHandler(async (req, res) => {
28   const { name, email, password } = req.body
29
30   const userExists = await User.findOne({ email })
31
32   if (userExists) {
33     res.status(400)
34     throw new Error('User already exists')
35   }
36
37   const user = await User.create({
38     name,
39     email,
40     password,
41   })
42
43   if (user) {
44     res.status(201).json({
45       _id: user._id,
46       name: user.name,
47       email: user.email,
48       isAdmin: user.isAdmin,
49       token: generateToken(user._id),
50     })
51   }
52 })
53
54 const updateUser = asyncHandler(async (req, res) => {
55   const { id, name, email, password } = req.body
56
57   const user = await User.findOne({ _id: id })
58
59   if (user) {
60     user.name = name
61     user.email = email
62     user.password = password
63     user.token = generateToken(user._id)
64     user.save()
65     res.json({
66       _id: user._id,
67       name: user.name,
68       email: user.email,
69       isAdmin: user.isAdmin,
70       token: user.token,
71     })
72   } else {
73     res.status(404)
74     throw new Error('User not found')
75   }
76 })
```

Şekil 7: users Controllers

product kontrollers

Bu denetleyici dosyası, ürünle ilgili tüm mantığı yönetir - bir ürün ekleyin, tüm ürünleri alın, bir ürünü silin veya bir ürünü değiştirin. Bu, sahip olduğumuz dört rota için birer tane olmak üzere, her biri belirli bir amaca hizmet eden dört fonksiyondan oluşacaktı. Bu nedenle, bu dört işlevin her birini ayrıntılı olarak tartışacağız. Bu dosyada sadece ürün modelini istememiz gerekiyor.

```
const Product = require('../models/ProductModel.js');
```

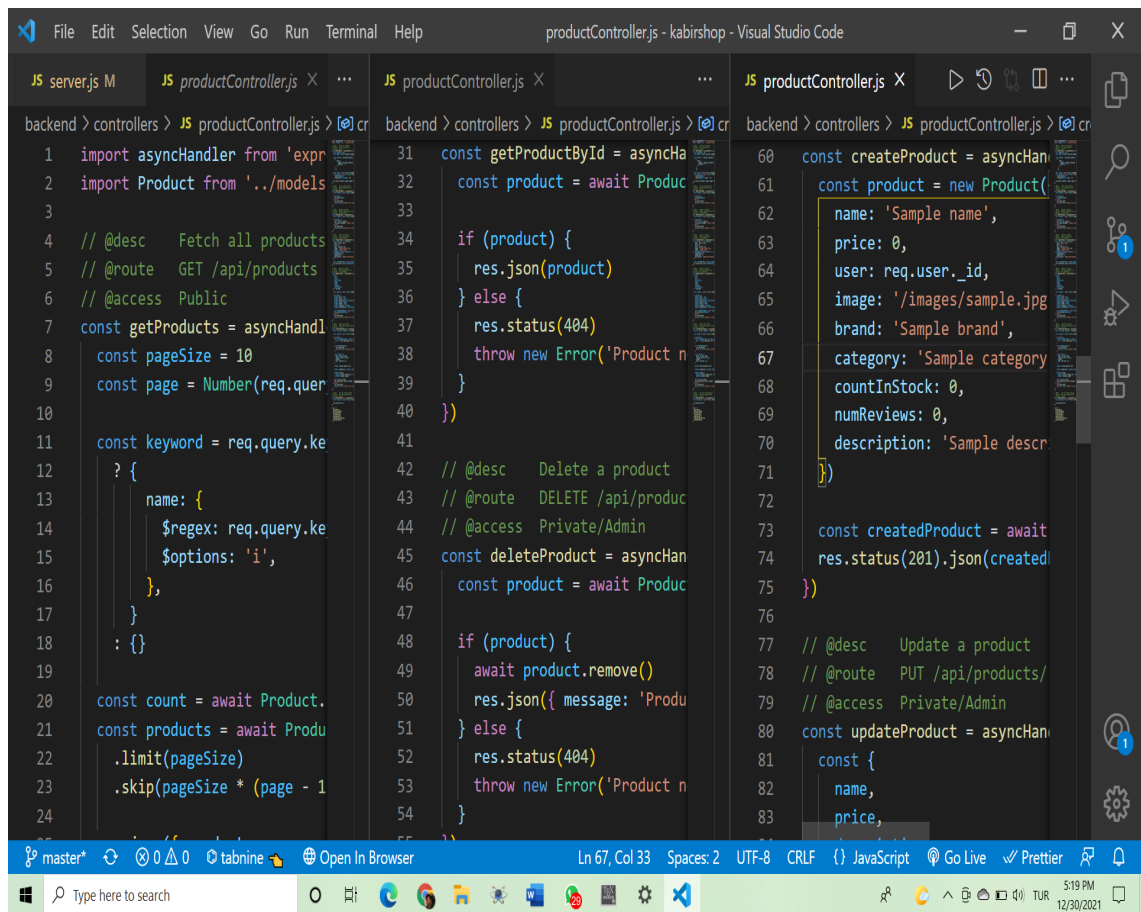
Şimdi, tüm ürünleri veritabanından alma işleviyle başlayacağız. Tüm öğeleri alacağız ve eklenme tarihine göre azalan düzende sıralayacağız. Daha sonra bu öğeleri JSON biçiminde döndürürüz.

Ardından, sepete yeni bir ürün eklemekle ilgileneceğiz. İstek gövdesini modelimizde olduğu gibi ön uçtan gönderdiğimiz için isteğin gövdesini yeni ürün için girdi olarak kullanacağız. Kullanıcı için yaptığımız gibi yeni ürünü oluştururken istek gövdesinin yapısını bozabilir ve ardından verileri sağlayabilirdik, ancak bu, bunu yapmanın daha temiz bir yoluydu. Daha sonra öğeyi veritabanına kaydediyoruz ve yeni ürünü yanıt olarak JSON formatında gönderiyoruz.

ürünleri güncellemekle ilgileneceğiz. Güncellenmiş bilgileri talep gövdesi aracılığıyla ve ürün kimliği params aracılığıyla alırız. Ürünü aramak ve yeni bilgilerle güncellemek için findByIdAndUpdate işlevini kullanacağız. Ardından güncellenen ürünü yanıt olarak göndeririz.

Son olarak, veri tabanından öğelerin silinmesi ile ilgileniyoruz. Öğе kimliğini params aracılığıyla alırız. Daha sonra ürünü bulup findByIdAndDelete fonksiyonunu kullanarak siliyoruz. Daha sonra bir başarı yanıtı döndürürüz. şimdi kodu kontrol edelim.

Daha sonra, bir kullanıcının oturum açıp açmadığını doğrulamak için özel bir ara katman işlevi ile ilgileneceğiz. Öncelikle dosyamızda config ve jwt'ye ihtiyacımız olacak.



Şekil 8: product Controllers

Daha sonra auth ara katman yazılımı işlevini oluşturmaya başlarız. Belirteci, isteğin 'x-auth-token' adlı başlık bölümünden alıyoruz. Belirteç yoksa, belirteci doğrulayacağız ve ardından kodu çözülen değişkeni yanıt olarak göndereceğiz.

4 Projenin ara yüzü

4.1 Ara yüzü kurma işlemleri

Projenin ara yüzü kısmına odaklanmaya başlayacağız. Bu kısımda projemizin müşteri tarafını React ile kurmaya başlayacağız ve ayrıca React uygulamasındaki tüm durumumuzu yönetmek için Redux kütüphanesinden faydalanacağız. Bu nedenle, her şeyden önce, kök klasörümüzün içinde (tüm arka uç dosyalarımızın bulunduğu) yeni bir klasör oluşturmamız gerekecek. Bu klasörü 'frontend' olarak adlandıracamız ve istemci tarafı ile ilgili tüm dosyaları bu klasörün içinde tutacağız.

Bizim için bir React projesi oluşturmak için create-react-app'i kullanacağız ve böylece babel ve webpack gibi çeşitli karmaşık şeylerle uğraşmamıza gerek kalmayacak. Bu komutu kullanmak süreci çok daha kolaylaştıracak ve gerçekten önemli olan şeylere odaklanabileceğiz.

frontend isimli klasörü oluşturduktan sonra o klasöre taşınacak ve klasör içinde yeni bir tepki uygulaması oluşturmak için aşağıdaki komutu çalıştıracaktık.

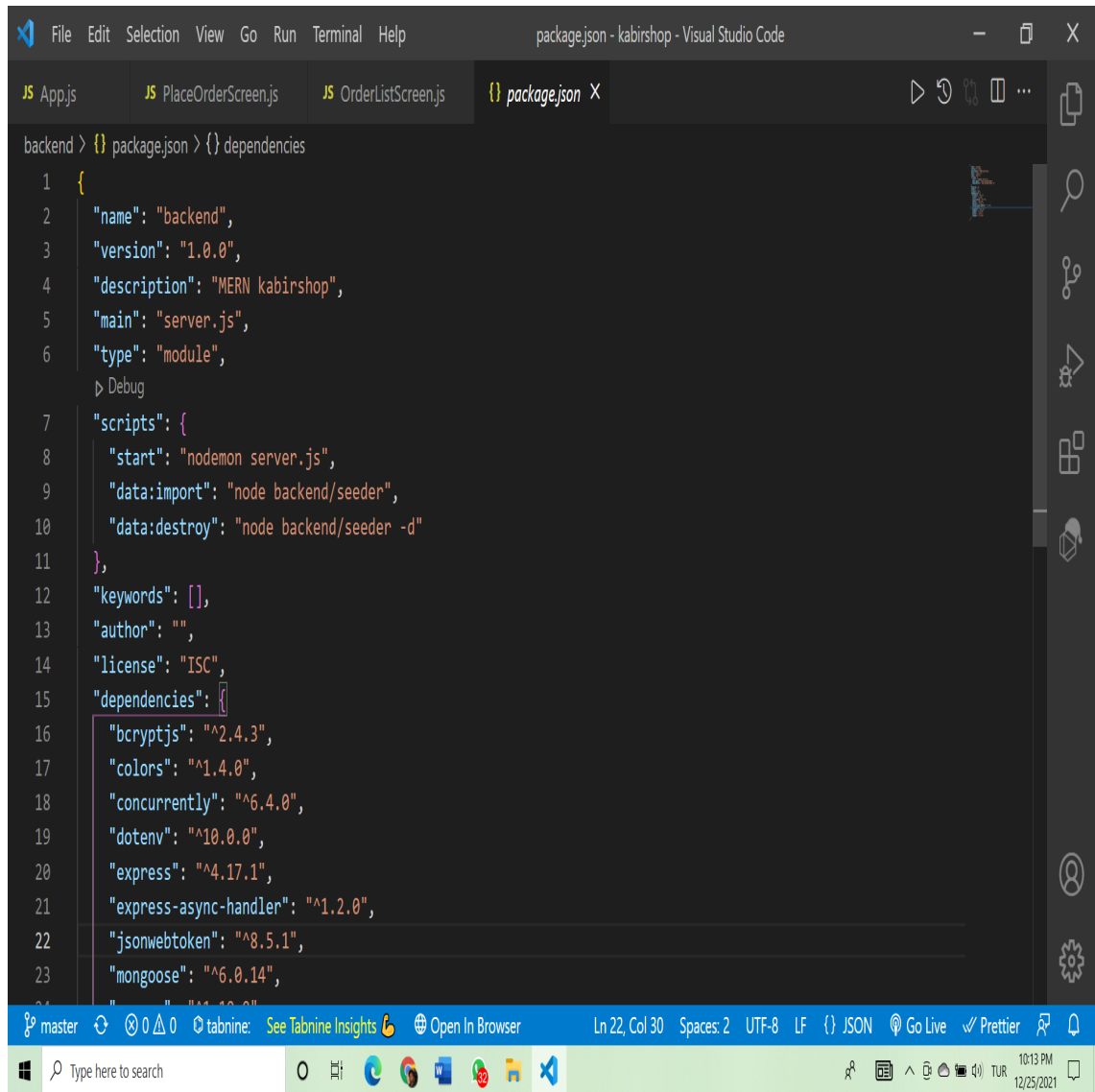
```
npx create-react-app .
```

Frontend adlı bir klasörde yeni bir React projesi kurmak için bu komutu yazarak ara yüzü klasörünün içine gidebiliriz.

```
npx create-react-app frontend
```

Bu, uygulamamızda yeni bir React projesi kuracak. Şimdi, ara yüzü klasörü içindeki package.json dosyasını açınca; Yüklü çeşitli bağımlılıklar içerdiğini göreceğiz. Ayrıca projemizde kullanacağımız bazı bağımlılıkları da kurardık. İşte istemci tarafının package.json dosyası. burada bahsedilen birçok bağımlılık var. Bunların hepsine projemizde ihtiyacımız olacak.

package.json



```
backend > {} package.json > {} dependencies
1  {
2    "name": "backend",
3    "version": "1.0.0",
4    "description": "MERN kabirshop",
5    "main": "server.js",
6    "type": "module",
7    "scripts": {
8      "start": "nodemon server.js",
9      "data:import": "node backend/seed",
10     "data:destroy": "node backend/seed -d"
11   },
12   "keywords": [],
13   "author": "",
14   "license": "ISC",
15   "dependencies": {
16     "bcryptjs": "^2.4.3",
17     "colors": "^1.4.0",
18     "concurrently": "^6.4.0",
19     "dotenv": "^10.0.0",
20     "express": "^4.17.1",
21     "express-async-handler": "^1.2.0",
22     "jsonwebtoken": "^8.5.1",
23     "mongoose": "^6.0.14",
```

Şekil 9: json package

4.2 Redux ouřturması

Ardından, Redux durum yönetimimizi kurmaya başlayacağız. src klasöründe store.js adında yeni bir dosya oluşturacağız. Bu bizim store.js dosyamız. storelar state yönrtmek görevi görecektir. Neler olduğunu anlamak için Redux hakkında biraz bilgi sahibi olmak řiddetle tavsiye edilmektedir.

Frontend klasörünün içinde reducer olarak adlandıracağımız bir klasör oluşturacaktık. Bu klasörün içinde dört dosya oluşturacağız - index, userReducer, productReducer, cartReducer ve orderReducer.

4.2.1 User Reducer

ilk önce kullanıcı reducer dosyasıdır. Adından da anlaşılacağı gibi, bu dosya kimlik doğrulama amaçlıdır. eylemler klasörünün türler dosyasından kimlik doğrulama için ihtiyaç duyacağımız tüm türleri içe aktararak başlıyoruz. Belirteci yerel depolamadan aldığımız bir başlangıç ??durumu oluşturduk. Ayrıca isAuthenticated'ı null ve isLoading'i false olarak ayarladık. Başlangıç ??olarak kullanıcı alanını da false olarak ayarladık. Ardından, her eylem türünü kontrol etmeye ve uygun değişiklikleri yapmaya başlarız.

userLoginReducer: Bu tür ise, kullanıcının yüklendiğini söyleyebiliriz ve bu nedenle isLoading'i true olarak ayarladık.

userRegisterReducer: Bu durumda isLoading'i false olarak ve ayrıca isAuthenticated ögesini true olarak ayarladık. Eylemler dosyasından aldığımız payload olarak da kullanıcıyı belirledik.

LOGIN-SUCCESS, REGISTER-SUCCESS : Her iki senaryoda da isAuthenticated ögesini true olarak ayarladık ve ayrıca yerel depoda alınan belirteci de ayarladık.

AUTH-ERROR, LOGIN-FAIL, LOGOUT-SUCCESS, REGISTER-FAIL : Tüm bu dört durumda, belirteci yerel depolamadan kaldırırız. Belirteci ve kullanıcıyı null olarak ayarladık. Ayrıca isAuthenticated ve isLoading'i false olarak ayarladık.

4.2.2 product Reducer

Şimdi bu uygulamadaki ürünlerle ilgili tüm görevleri yapacak olan productReducer dosyası ile ilgileneceğiz. Önce ürünlerle ilgili tüm türleri ithal edeceğiz. Ürünleri boş bir diziye ve yüklemeyi false olarak ayarladığımız bir başlangıç ??durumu kurduk. Ardından, her eylem türünü kontrol etmeye ve uygun değişiklikleri yapmaya başlarız.

getProductReducer: Bu, ürünü almak için redüktördür. Bu durumda, eylemlerden alınan yük olarak ürünler dizisini ayarlayacağız. Ayrıca ürünlerin yüklendiğini belirtmek için yüklemeyi false olarak ayarladık.

addProductReducer: Bu durumda spread operatörünü kullanarak durumumuzu çağırıyoruz ve ardından payload'dan alınan yeni ürünleri ürünler dizisine ekliyoruz.

deleteProductReducer: Bu durumda, payload üzerinden silinen ürünlerin kimliğini alırız. Böylece ürünler dizisini alıyoruz ve kimliği eşleşen ürünü filtreleyerek kaldırıyoruz.

updateProductReducer: Bu durumda, yükten kimliği ve güncellenmiş ürünü alırız. Daha sonra id'sini kullanarak item dizisinden ürünü buluyoruz ve ardından yeni item ile güncelliyoruz.

4.2.3 cart Reducer

Bu dosyada kullanıcının sepeti ile ilgili reducerleri ele alacağız. İşlemler klasöründe tanımladığımız türler dosyasından sepetle ilgili türleri içe aktarıyoruz. Sepetin null ve yüklemenin false olarak ayarlandığı ilk durumu tanımlarız.

GET-CART : Bu durumda, eylem dosyasındaki payload üzerinden sepeti alırız ve ilk durumumuzda tanımladığımız sepete ayarlarız. Ayrıca yüklemeyi false olarak ayarladık.

ADD-TO-CART, DELETE-FROM-CART: Her iki durumda da, güncellenmiş arabayı alırız ve bu nedenle, arabayı eylemlerden alınan yüke ayarlarız.

CART-LOADING : Bu durumda, yüklemeyi true olarak ayarladık.

4.2.4 order Reducer

Bu dosyada uygulamamızdaki siparişler ile ilgili redüktörleri ele alacağız. Önce siparişler için ilgili tüm türleri içe aktarıyoruz. Ardından, siparişleri boş bir dizi olarak tanımladığımız ve ayrıca yüklemeyi tanımladığımız ve false olarak ayarladığımız bir başlangıç durumu tanımlarız.

GET-ORDERS : Bu durumda, siparişler dizisini, eylemler dosyasından alınan yüke ayarladık. Ayrıca yüklemeyi false olarak ayarladık.

CHECKOUT : Bu durumda, yeni siparişi payload'dan alır ve onu siparişler dizisine ekleriz.

ORDERS-LOADING : Yüklemeyi true olarak ayarladık.

combining reducers

Bu dosyada, tüm farklı dosyalardan tüm reducerleri içe aktarıyoruz : sepet, sipariş, yetkilendirme .. ve ardından redux'dan aldığımız CombineReducers işlevini kullanarak bunları birleştiriyoruz.

tüm redux işlerini bitirdik. Artık uygulamamız için bileşenler oluşturmaya odaklanabiliriz.

5 SONUÇLAR VE ÖNERİLER

6 EKLER

KAYNAKLAR

- [1] CTAN,http://zelmanov.ptep-online.com/ctan/lshort_turkish.pdf [Ziyaret Tarihi: 4 Kasım 2011]—> Kaynak yazarı bilinmeyen yabancı bir çalışmadan alınmış ise:
- [2] Anonymous, 1989, Farm accountancy data network, an A-Z of methodology, Commission Report of the EC, Brussels, 16-19.
- [3] <http://akgul.bilkent.edu.tr/Yunus/lshort.pdf> —> Kaynak kongreden alınmış ise:
- [4] Calvalho, M. ve Ludermir, T.B., 2007, Particle Swarm Optimization of Neural Network Architectures and Weights, Seventh International Conference on Hybrid Intelligent Systems, Almanya, 336-339. —> Kaynak aktüel dergi ve gazete haberinden alınmış ise:
- [5] Şevkli, M., ve Yenisey, M. M., 2006, Atölye Tipi Çizelgeleme Problemleri için Parçacık Sürü Optimizasyonu Yöntemi, İtÜdergisi/d Mühendislik, Cilt 5, Sayı 2(1), 58-68. —> Kaynak aktüel dergi ve gazete haberinden alınmış ise:
- [6] <http://kisi.deu.edu.tr/umit.akinci/latexseminer.pdf>—> Kaynak yazarı bilinmeyen ulusal bir çalışmadan alınmış ise:
- [7] Anonim, 2006, Tarım istatistikleri özeti, DİE Yayınları, No;12, Ankara, 22-23. —> Kaynak yazarı bilinmeyen ulusal bir çalışmadan alınmış ise:

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı :

Uyruğu :

Doğum Yeri ve Tarihi:

Adres :

Telefon :

e-mail :

EĞİTİM DURUMU

Lisans Öğrenimi : BŞEÜ Bilgisayar Mühendisliği Bölümü

Bitirme Yılı :

Lise :

İŞ DENEYİMLERİ

Yıl :

Kurum :

Stajlar :

İLGİ ALANLARI:

YABANCI DİLLER:

BELİRTMEK İSTEDİĞİNİZ DİĞER ÖZELLİKLER: