grafik, ekran görüntüsü, simge, sembol, tasarım içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.,

**E-Ticaret Uygulaması: Happier Shoppier**

**BULUT BİLİŞİM VE UYGULAMALARI BLM3522 DERSİ VİZE ÖDEVİ**

**4.PROJE: E-TİCARET UYGULAMASI**

**Hazırlayanlar:**

**Muhammed Faruk Gözay 22290673**

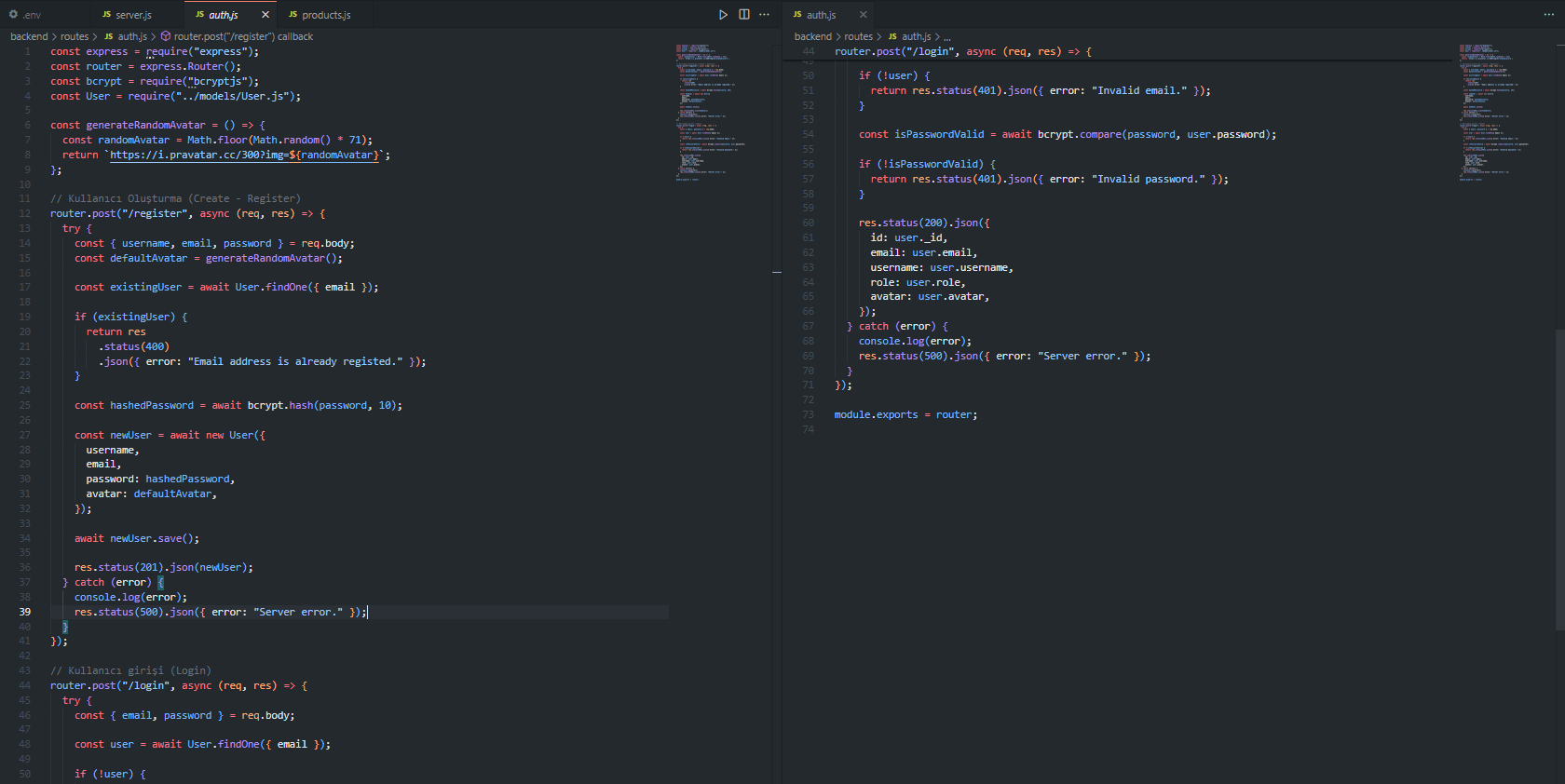
**Batuhan Dede 22290502**

**Yavuz Selim Demirhan 22290565**

**Github Repo Adresi:**

[**https://github.com/BLM3522-Grubu/Proje-4-E-Ticaret-Uygulamasi-HappierShoppier--Node.js-React-MongoDB-Azure-**](https://github.com/BLM3522-Grubu/Proje-4-E-Ticaret-Uygulamasi-HappierShoppier--Node.js-React-MongoDB-Azure-)

**https://salmon-ground-005958803.6.azurestaticapps.net/**

**BACKEND AÇIKLAMALARI:**

**auth.js Kimlik Doğrulama İşlemleri**

**Kullanıcı Kayıt (POST /register)**

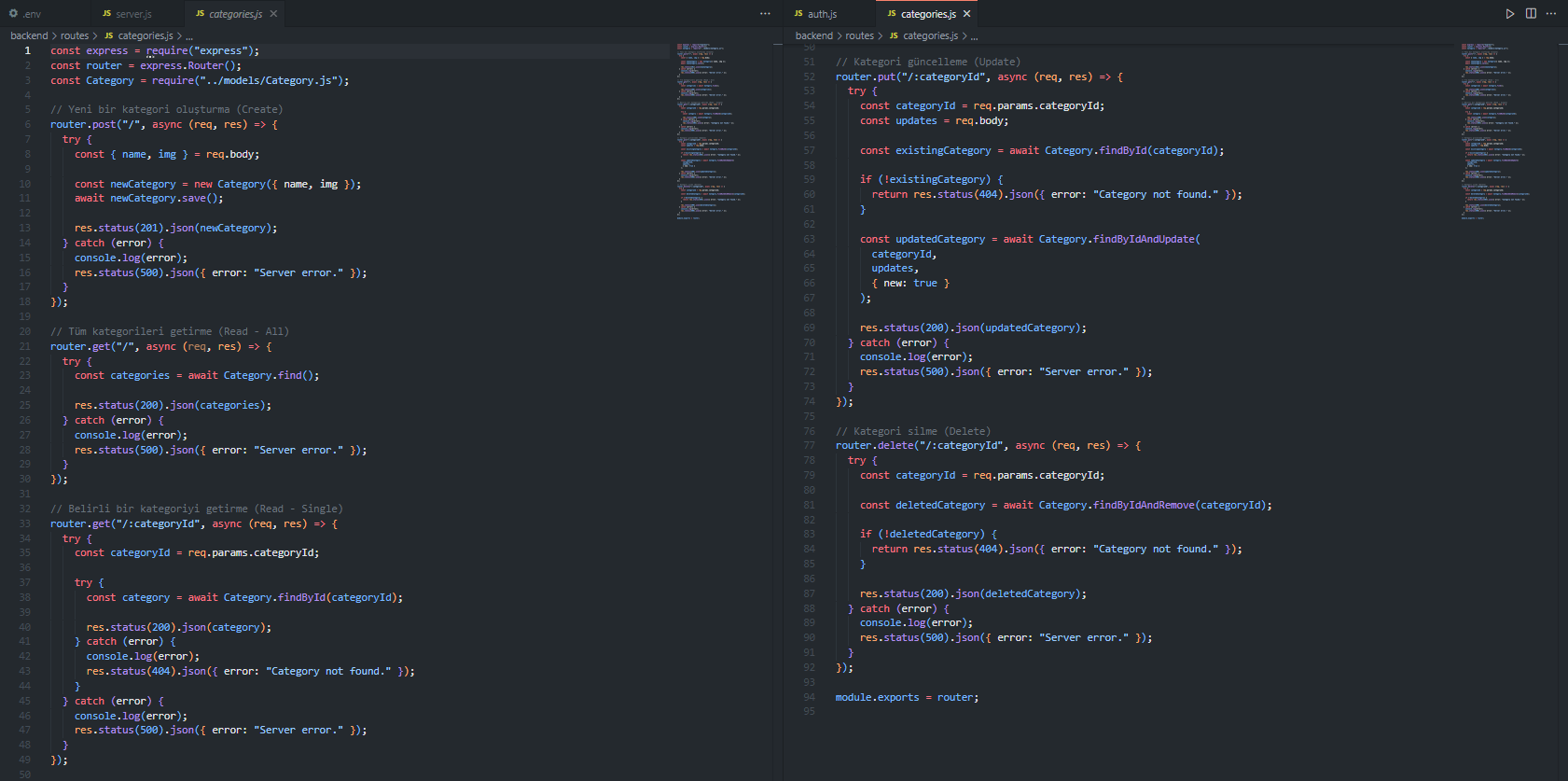
* **Kullanıcıdan username, email ve password alınır.**
* **bcrypt ile parola şifrelenir.**
* **Aynı email varsa kayıt reddedilir.**
* **Rastgele avatar atanır.**
* **Yeni kullanıcı veritabanına kaydedilir.**
* **Başarılı olursa kullanıcı bilgileri JSON olarak döner.**

**Kullanıcı Giriş (POST /login)**

* **Kullanıcıdan email ve password alınır.**
* **Email veritabanında aranır.**
* **Parola, bcrypt.compare() ile doğrulanır.**
* **Bilgiler doğruysa giriş başarılı olur ve kullanıcı bilgileri JSON olarak döner.**
* **Hatalı girişte uygun hata mesajı gönderilir.**

**Güvenlik**

* **Şifreler düz metin olarak değil, hash’lenmiş şekilde saklanır.**
* **Hatalar try-catch blokları ile yönetilir.**

****

**categories.js Kategoriler İşlemleri**

**Bu dosyada kategorilerle ilgili CRUD (Oluşturma, Okuma, Güncelleme, Silme) işlemleri gerçekleştirilir.**

**✅ Yeni Kategori Oluşturma (Create) Yol: POST /api/categories**

**name ve img bilgileri alınır.**

**Yeni Category nesnesi oluşturulup kaydedilir.**

**Başarılı olursa 201 Created, hata olursa 500 Server Error.**

**📥 Tüm Kategorileri Getir (Read - All) Yol: GET /api/categories**

**Veritabanındaki tüm kategoriler Category.find() ile getirilir.**

**200 OK ile döner.**

**🔍 Belirli Kategoriyi Getir (Read - Single) Yol: GET /api/categories/:categoryId**

**req.params.categoryId ile istenen ID'deki kategori bulunur.**

**Bulunamazsa 404 Not Found, başarılıysa 200 OK.**

**📝 Kategori Güncelleme (Update) Yol: PUT /api/categories/:categoryId**

**params.categoryId ile hedef kategori bulunur.**

**Gönderilen body ile güncellenir (findByIdAndUpdate).**

**new: true ile güncellenmiş hali döner. Bulunamazsa 404, başarılıysa 200 OK.**

**❌ Kategori Silme (Delete)**

**Yol: DELETE /api/categories/:categoryId**

**ID ile kategori bulunur ve silinir.**

**Silinemezse 404, başarılıysa 200 OK.**

**⚠️ Hata Yönetimi**

**Her işlem try-catch blokları ile hata kontrolüne sahiptir.**

**Sunucu hatalarında 500 Server Error mesajı döner.**

**coupons.js Kupon İşlemleri (CRUD Açıklamaları)**

**Bu dosyada kupon (coupon) ile ilgili tüm işlemler gerçekleştirilir: ekleme, listeleme, güncelleme, silme ve arama.**

**✅ Yeni Kupon Oluşturma (Create)**

**Yol: POST /api/coupons**

* **Gövdeden (req.body) code, discountPercent, vb. bilgiler alınır.**
* **Aynı code varsa 400 hatası döner: "This coupon is already exists."**
* **Yoksa yeni kupon veritabanına kaydedilir.**
* **201 Created cevabı döner.**

**📥 Tüm Kuponları Getir (Read - All)**

**Yol: GET /api/coupons**

* **Veritabanındaki tüm kuponlar Coupon.find() ile çekilir.**
* **200 OK ile liste döner.**

**🔍 Kupon Getir (ID ile) (Read - Single by ID)**

**Yol: GET /api/coupons/:couponId**

* **URL'den gelen couponId ile kupon aranır (findById).**
* **Bulunamazsa 404, varsa 200 döner.**

**🔍 Kupon Getir (Kod ile) (Read - Single by Code)**

**Yol: GET /api/coupons/code/:couponCode**

* **Kupon code ile aranır (findOne).**
* **Kod yoksa 404, varsa discountPercent ile 200 OK döner.**

**📝 Kupon Güncelleme (Update)**

**Yol: PUT /api/coupons/:couponId**

* **Belirtilen couponId ile kupon bulunur.**
* **Güncelleme işlemi findByIdAndUpdate() ile yapılır.**
* **new: true ile güncellenmiş hali döner.**
* **200 OK, bulunmazsa 404.**

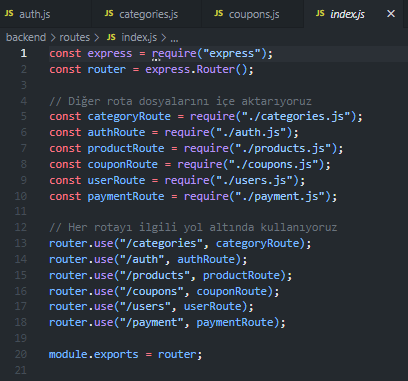
**❌ Kupon Silme (Delete)**

**Yol: DELETE /api/coupons/:couponId**

* **couponId ile kupon silinir (findByIdAndRemove).**
* **Bulunamazsa 404, başarılıysa 200.**

**⚠️ Hata Yönetimi**

**Tüm işlemler try-catch ile korunur.  
Sunucu taraflı hatalarda 500 Internal Server Error mesajı döner.**

****

**routes/index.js Açıklaması (Router Merkez Dosyası)**

**📌 1. Gerekli modüller yükleniyor:**

**const express = require("express");**

**const router = express.Router();**

* **Express modülü ile yeni bir yönlendirici (router) oluşturuluyor.**

**📌 2. Alt rota dosyaları içe aktarılıyor:**

**const categoryRoute = require("./categories.js");**

**const authRoute = require("./auth.js");**

**const productRoute = require("./products.js");**

**const couponRoute = require("./coupons.js");**

**const userRoute = require("./users.js");**

**const paymentRoute = require("./payment.js");**

* **Her biri kendi içindeki CRUD işlemlerini barındıran route dosyalarıdır:**
  + **categories.js → kategoriler**
  + **auth.js → kimlik doğrulama**
  + **products.js → ürün işlemleri**
  + **coupons.js → kupon işlemleri**
  + **users.js → kullanıcı işlemleri**
  + **payment.js → ödeme işlemleri**

**📌 3. Rotalar bağlanıyor:**

**router.use("/categories", categoryRoute);**

**router.use("/auth", authRoute);**

**router.use("/products", productRoute);**

**router.use("/coupons", couponRoute);**

**router.use("/users", userRoute);**

**router.use("/payment", paymentRoute);**

**Bu satırlar sayesinde aşağıdaki yollar aktif olur:**

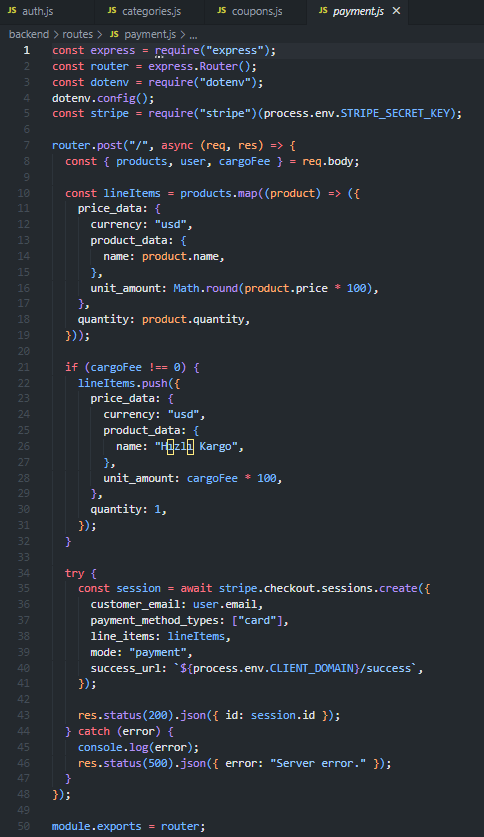
* **/api/categories/...**
* **/api/auth/...**
* **/api/products/...**
* **/api/coupons/...**
* **/api/users/...**
* **/api/payment/...**

**app.use("/api", routes) şeklinde bir ana kullanım varsa (örneğin server.js dosyasında), bu yollar http://localhost:PORT/api/categories, api/products gibi çalışır.**

**📌 4. Dışa aktarım:**

**module.exports = router;**

* **Bu router, sunucunun ana dosyasında (server.js) app.use("/api", router) şeklinde kullanılır.**

****

**payment.js Açıklaması**

**📌 1. Modüller yükleniyor:**

**const express = require("express");**

**const router = express.Router();**

**const dotenv = require("dotenv");**

**dotenv.config();**

**const stripe = require("stripe")(process.env.STRIPE\_SECRET\_KEY);**

* **Express ile yönlendirici oluşturuluyor.**
* **.env dosyasındaki Stripe API anahtarı yükleniyor.**
* **Stripe SDK’sı başlatılıyor.**

**📌 2. POST / — Ödeme Oturumu Oluşturma**

**router.post("/", async (req, res) => {**

**const { products, user, cargoFee } = req.body;**

**Frontend’den gönderilen istek products, user ve cargoFee bilgilerini içeriyor.**

**📌 3. Ürünleri Stripe formatına dönüştürme:**

**const lineItems = products.map((product) => ({**

**price\_data: {**

**currency: "usd",**

**product\_data: { name: product.name },**

**unit\_amount: Math.round(product.price \* 100),**

**},**

**quantity: product.quantity,**

**}));**

* **Her ürün Stripe'ın line\_items yapısına dönüştürülüyor.**
* **Stripe, fiyatı cent cinsinden beklediği için 100 ile çarpılıyor.**

**📌 4. Kargo varsa onu da ekle:**

**if (cargoFee !== 0) {**

**lineItems.push({**

**price\_data: {**

**currency: "usd",**

**product\_data: { name: "📦 Kargo" },**

**unit\_amount: cargoFee \* 100,**

**},**

**quantity: 1,**

**});**

**}**

* **Eğer kargo ücreti varsa, sepete ayrı bir "Kargo" ürünü olarak ekleniyor.**

**📌 5. Stripe Checkout oturumu oluşturma:**

**const session = await stripe.checkout.sessions.create({**

**customer\_email: user.email,**

**payment\_method\_types: ["card"],**

**line\_items: lineItems,**

**mode: "payment",**

**success\_url: `${process.env.CLIENT\_DOMAIN}/success`,**

**});**

* **Kullanıcı maili ile ödeme oturumu başlatılıyor.**
* **Ödeme başarılı olursa yönlendirilecek URL .env dosyasından alınıyor (CLIENT\_DOMAIN).**

**📌 6. Yanıtlar:**

**res.status(200).json({ id: session.id });**

* **Stripe oturum ID’si frontend’e gönderiliyor.**

**Hata durumunda:**

**res.status(500).json({ error: "Server error." });**

**📌 7. Dışa Aktarım**

**module.exports = router;**

**Bu dosya routes/index.js aracılığıyla /api/payment altında erişilebilir.**

**🔗 Örnek istek (POST /api/payment):**

**{**

**"user": { "email": "test@example.com" },**

**"products": [**

**{ "name": "T-shirt", "price": 29.99, "quantity": 2 }**

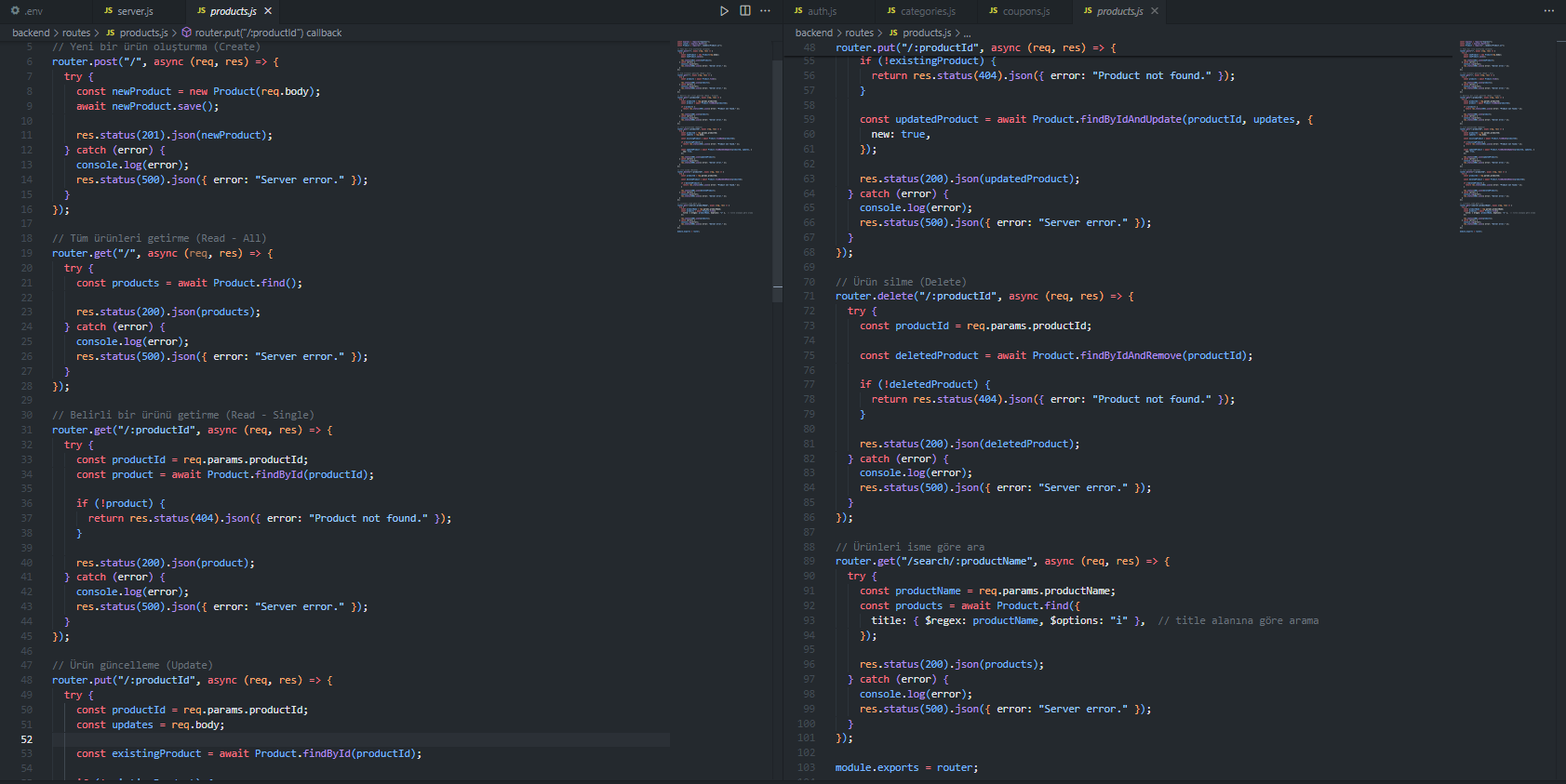
**],**

**"cargoFee": 10**

**}**

**Stripe başarılı olursa dönüş:**

**{ "id": "cs\_test\_123456789" }**

****

**Açıklama: products.js**

**📌 1. Giriş ve Model Tanımı**

**const express = require("express");**

**const router = express.Router();**

**const Product = require("../models/Product.js");**

* **Express Router başlatılıyor.**
* **Product modeli kullanılarak MongoDB ile etkileşim sağlanacak.**

**✅ 2. Ürün Oluşturma (POST /)**

**router.post("/", async (req, res) => {**

**try {**

**const newProduct = new Product(req.body);**

**await newProduct.save();**

**res.status(201).json(newProduct);**

**} catch (error) {**

**console.log(error);**

**res.status(500).json({ error: "Server error." });**

**}**

**});**

* **req.body'de gelen bilgilerle ürün oluşturulur.**
* **Başarıyla kaydedilirse 201 ile yeni ürün döner.**

**📥 3. Tüm Ürünleri Getirme (GET /)**

**router.get("/", async (req, res) => {**

**const products = await Product.find();**

**res.status(200).json(products);**

**});**

* **Tüm ürünler veritabanından çekilir.**
* **Frontend bu endpoint’e bağlanır.**

**🔍 4. Tek Ürün Getirme (GET /:productId)**

**router.get("/:productId", async (req, res) => {**

**const product = await Product.findById(req.params.productId);**

**if (!product) return res.status(404).json({ error: "Product not found." });**

**res.status(200).json(product);**

**});**

* **Belirli productId'ye göre ürün döner.**
* **Ürün yoksa 404 verir.**

**✏️ 5. Ürün Güncelleme (PUT /:productId)**

**router.put("/:productId", async (req, res) => {**

**const productId = req.params.productId;**

**const updates = req.body;**

**const existingProduct = await Product.findById(productId);**

**if (!existingProduct) return res.status(404).json({ error: "Product not found." });**

**const updatedProduct = await Product.findByIdAndUpdate(productId, updates, { new: true });**

**res.status(200).json(updatedProduct);**

**});**

* **İlgili productId'ye sahip ürün güncellenir.**
* **new: true => güncellenmiş hali döner.**

**❌ 6. Ürün Silme (DELETE /:productId)**

**router.delete("/:productId", async (req, res) => {**

**const deletedProduct = await Product.findByIdAndRemove(req.params.productId);**

**if (!deletedProduct) return res.status(404).json({ error: "Product not found." });**

**res.status(200).json(deletedProduct);**

**});**

* **Belirli productId'li ürün veritabanından silinir.**

**🔍 7. Ürün Arama (GET /search/:productName)**

**router.get("/search/:productName", async (req, res) => {**

**const productName = req.params.productName;**

**const products = await Product.find({**

**title: { $regex: productName, $options: "i" }**

**});**

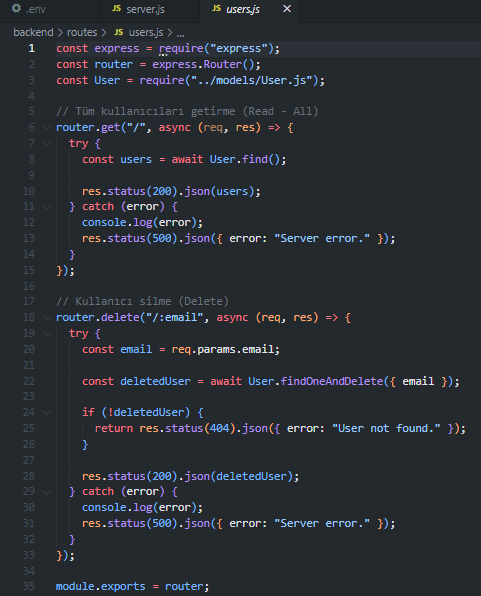
**res.status(200).json(products);**

**});**

* **productName içeren başlıklarla eşleşen ürünleri döner (büyük/küçük harfe duyarsız).**

**Bu endpoint'lerin hepsi index.js üzerinden /products altına yönlendirildiği için örneğin:**

* **Tüm ürünler → GET /api/products**
* **Ürün oluştur → POST /api/products**
* **Güncelle → PUT /api/products/:productId**
* **Sil → DELETE /api/products/:productId**

****

**Açıklama: users.js**

**📦 Giriş**

**const express = require("express");**

**const router = express.Router();**

**const User = require("../models/User.js");**

* **Express Router başlatılıyor.**
* **User modeli kullanılarak MongoDB'deki users koleksiyonuna erişim sağlanıyor.**

**🧾 1. Tüm Kullanıcıları Getirme – GET /api/users**

**router.get("/", async (req, res) => {**

**const users = await User.find();**

**res.status(200).json(users);**

**});**

* **Veritabanındaki tüm kullanıcıları çeker.**
* **Başarılıysa 200 OK ile kullanıcı listesi döner.**
* **Hata varsa 500 Server Error.**

**❌ 2. Kullanıcı Silme – DELETE /api/users/:email**

**router.delete("/:email", async (req, res) => {**

**const email = req.params.email;**

**const deletedUser = await User.findOneAndDelete({ email });**

**if (!deletedUser) {**

**return res.status(404).json({ error: "User not found." });**

**}**

**res.status(200).json(deletedUser);**

**});**

* **E-posta adresine göre kullanıcı silinir.**
* **Eşleşen kullanıcı yoksa 404 döner.**
* **Silinen kullanıcı JSON olarak döndürülür.**

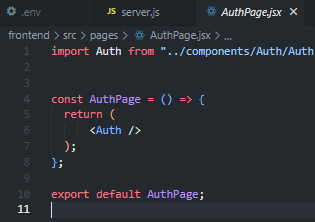
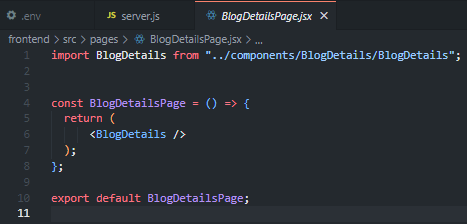
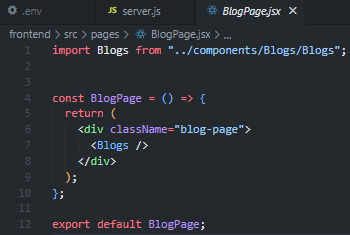
**🔗 Route Kullanımı (index.js dosyasına bağlı)**

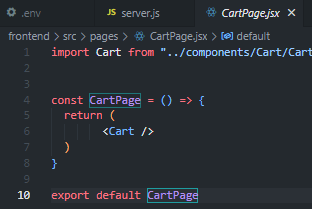
**Bu users.js router’ı, index.js içinde şu şekilde bağlanmalı:**

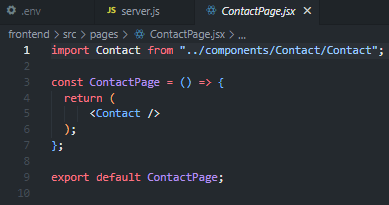
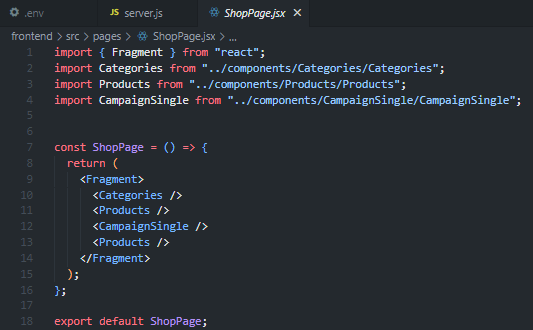
**router.use("/users", userRoute);**

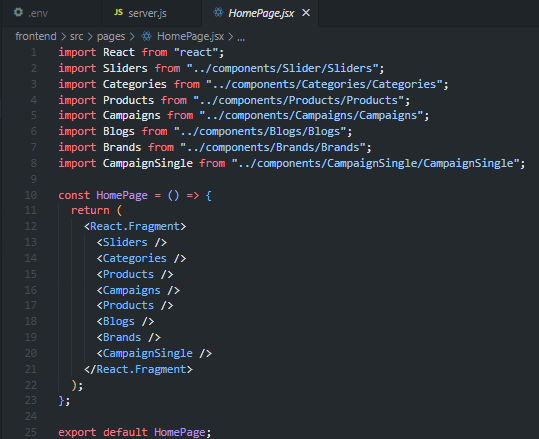
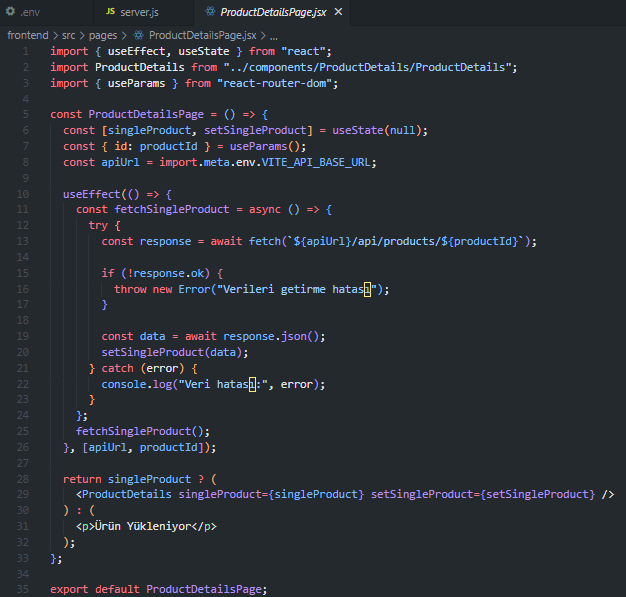
**Yani:**

* **Tüm kullanıcıları listele: GET /api/users**
* **Kullanıcı sil: DELETE** [**/api/users/example@example.com**](mailto:/api/users/example@example.com)

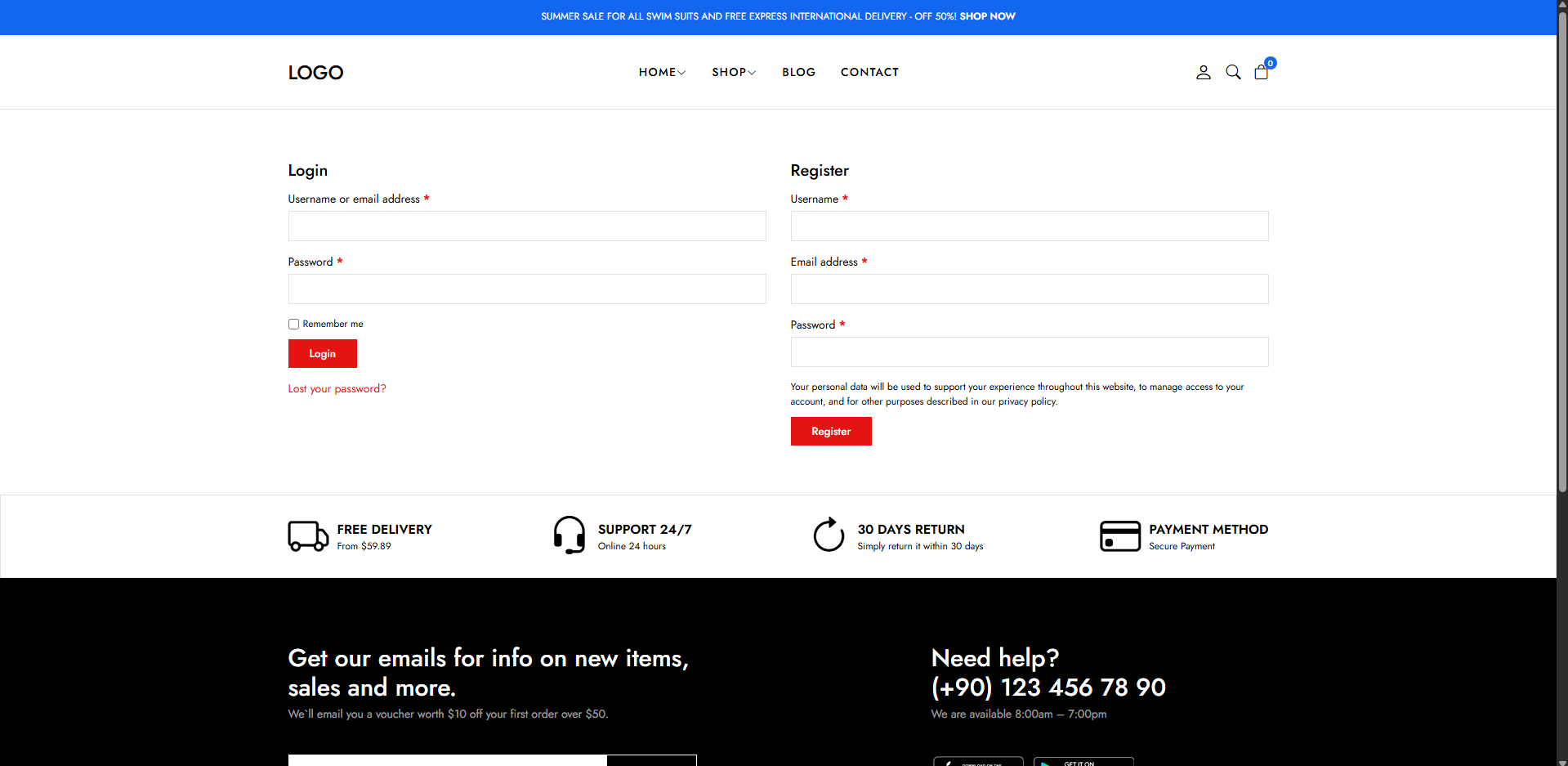
**FRONTEND AÇIKLAMALARI**

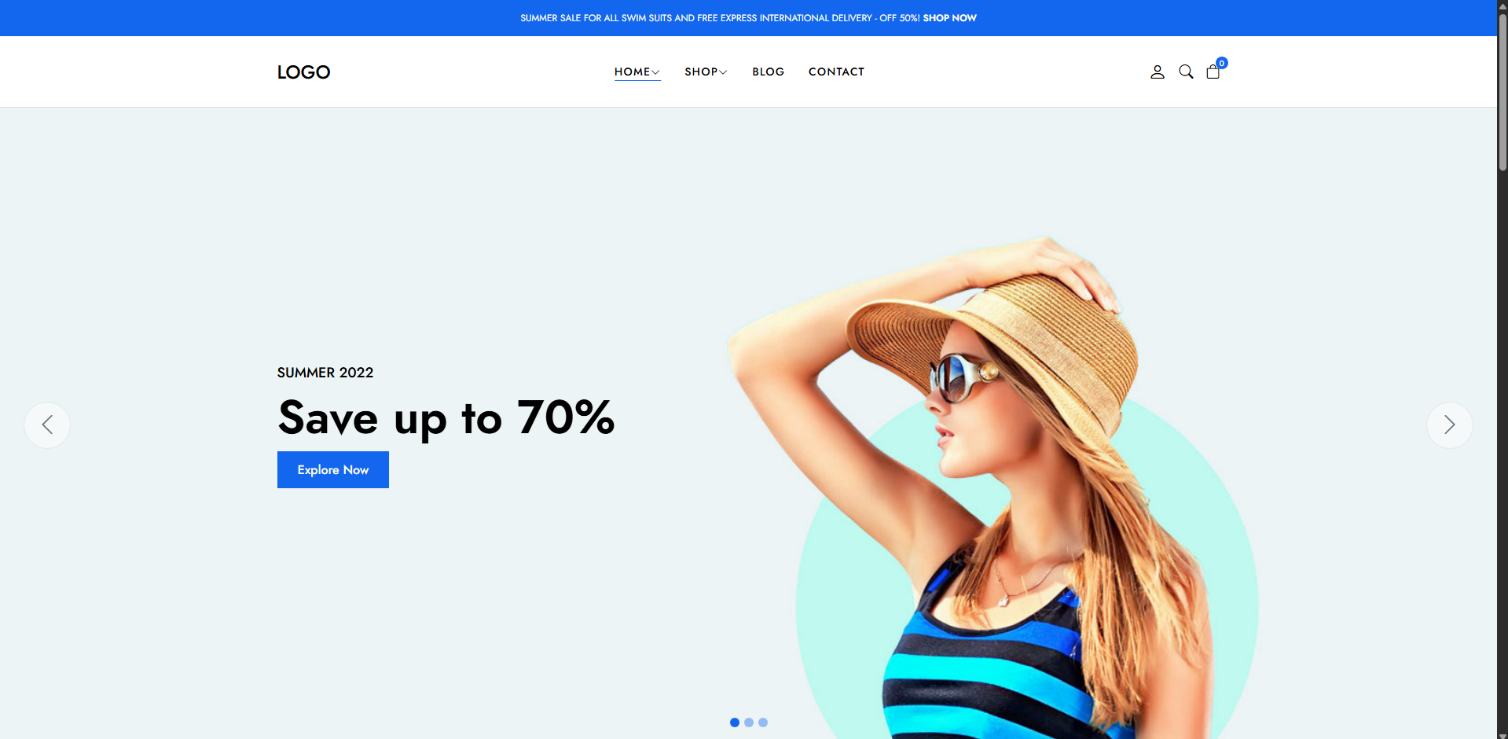
****

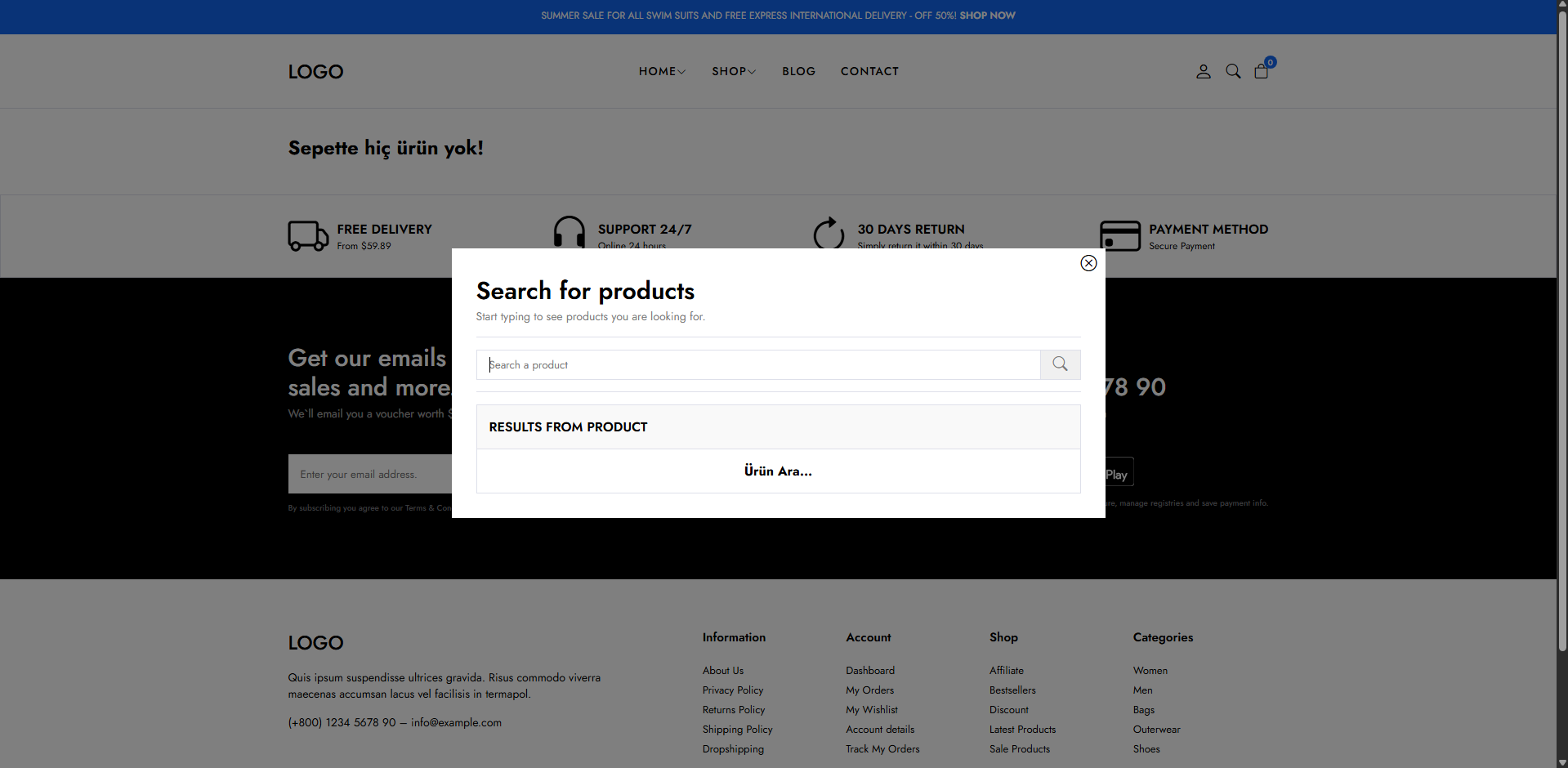
****

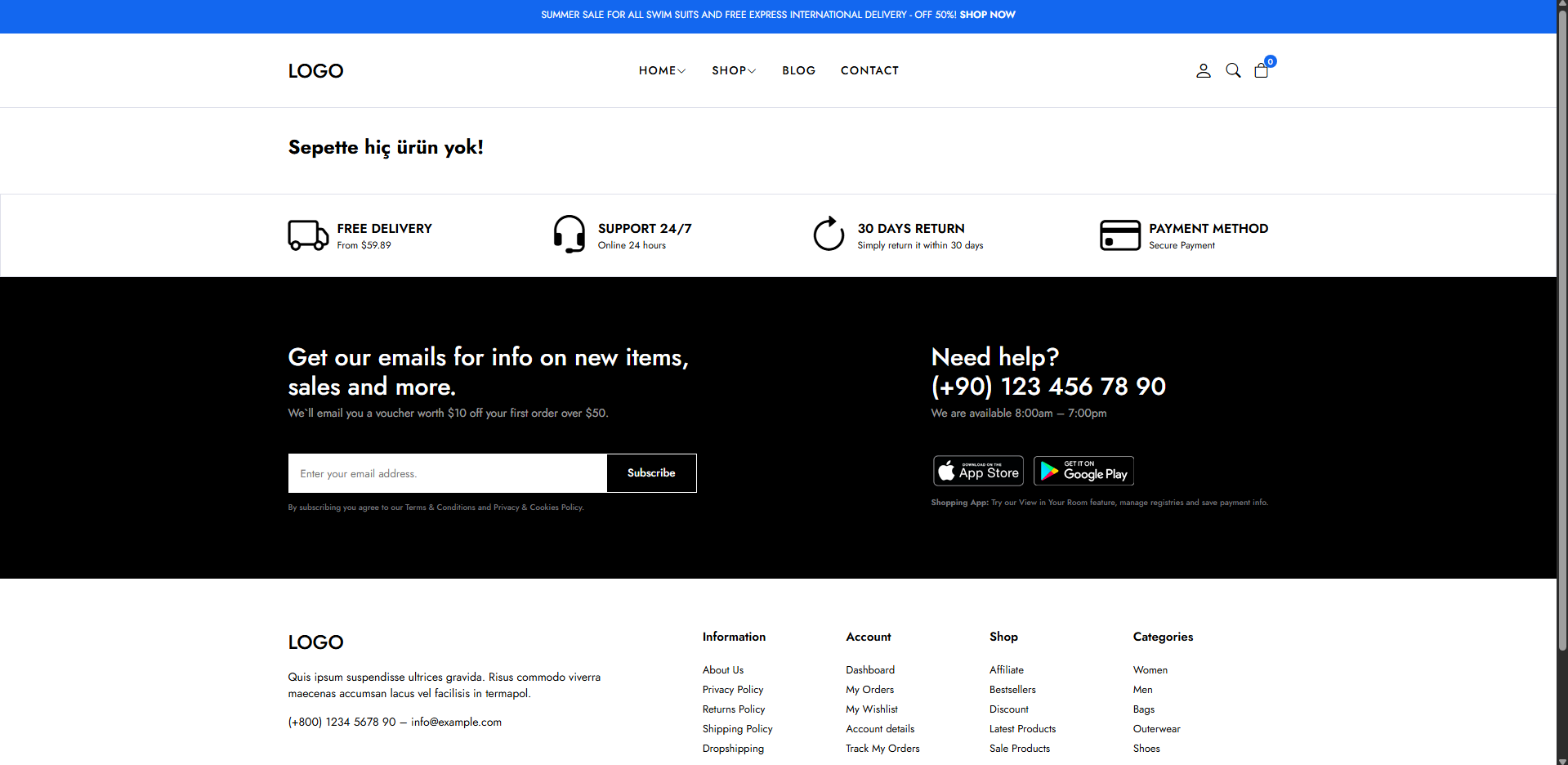
****

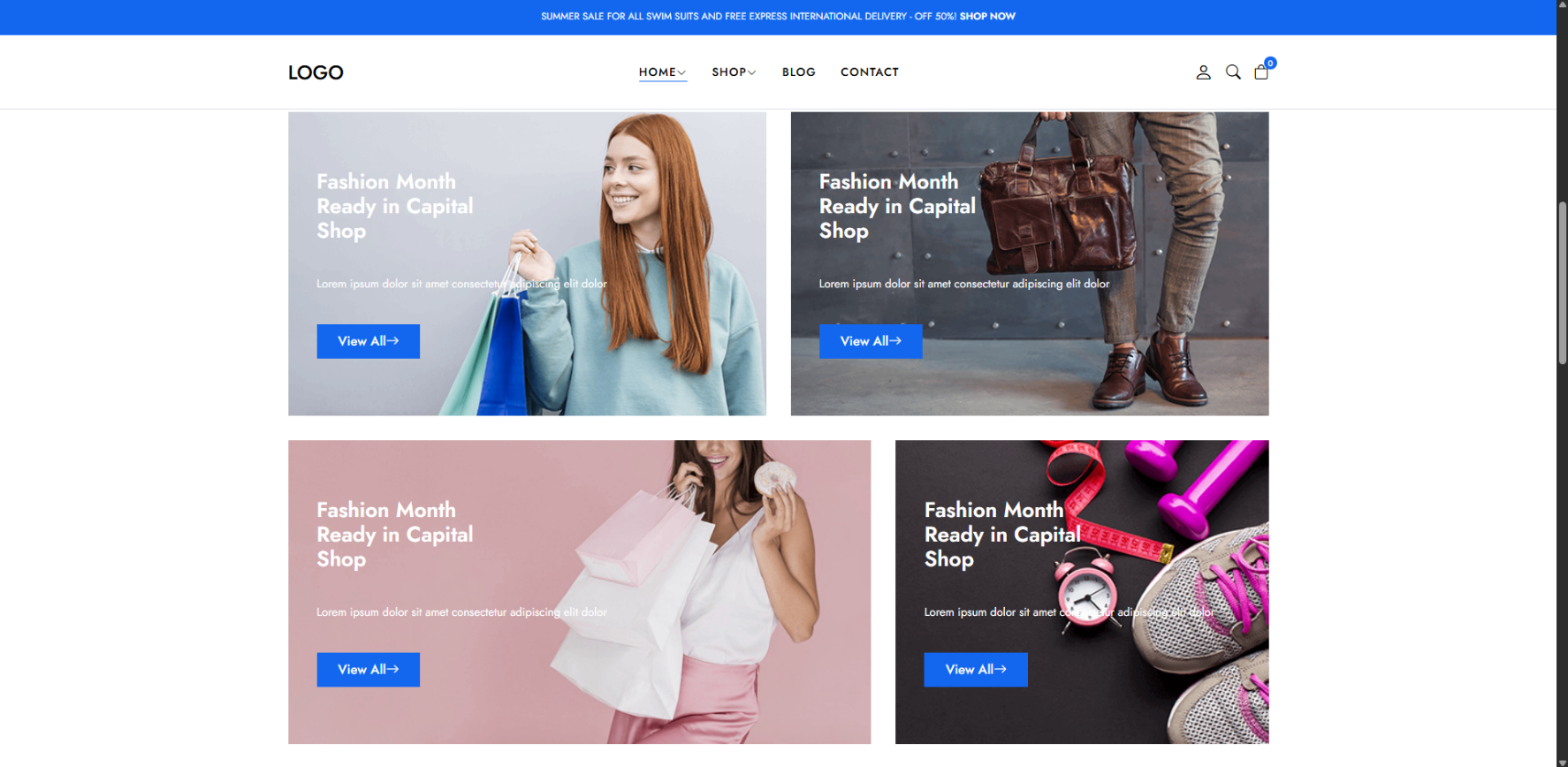
**Front-end hazır react şablonlarından oluşturulmuştur.**

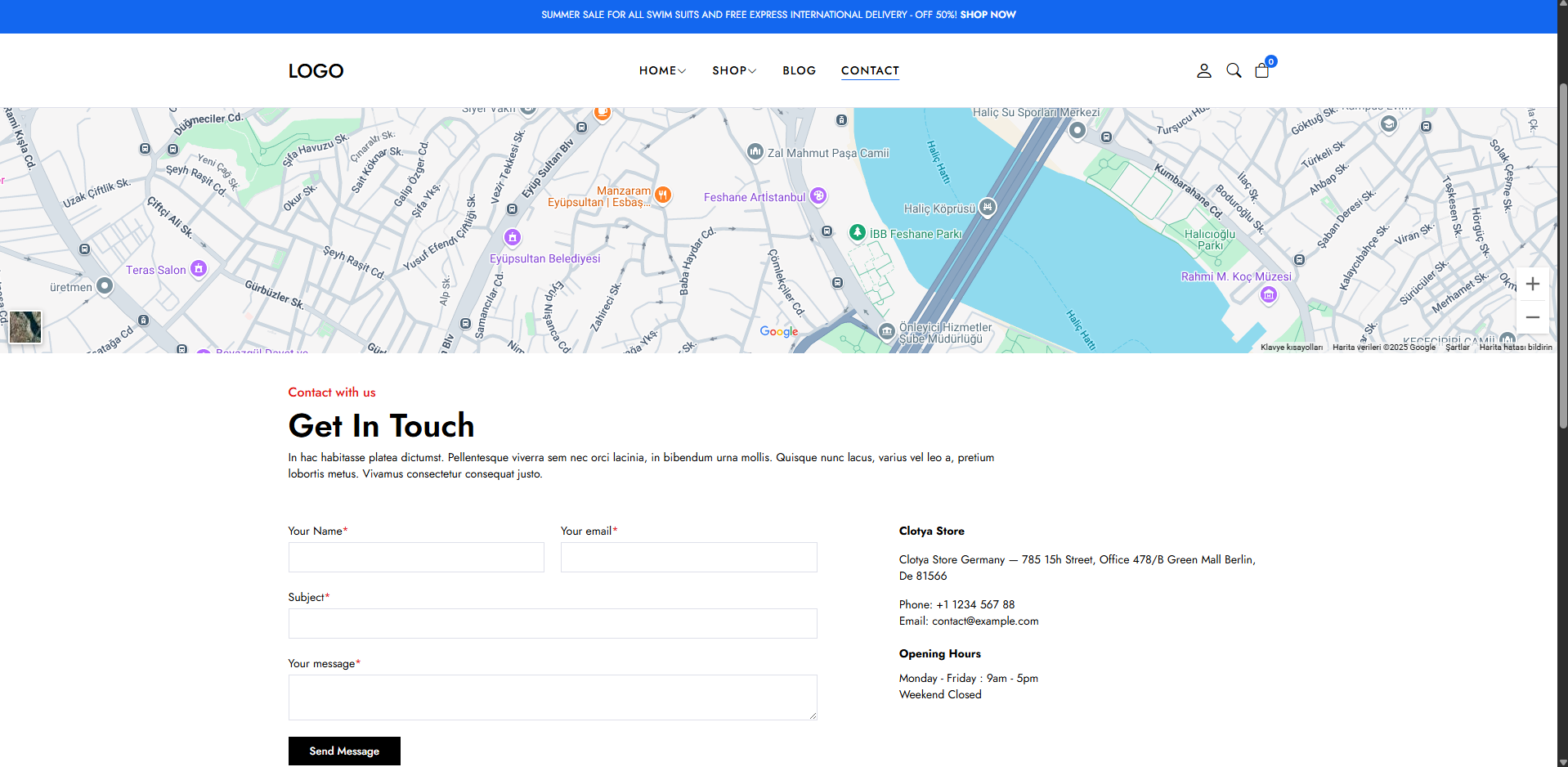
**SİTE GÖRÜNTÜLERİ**

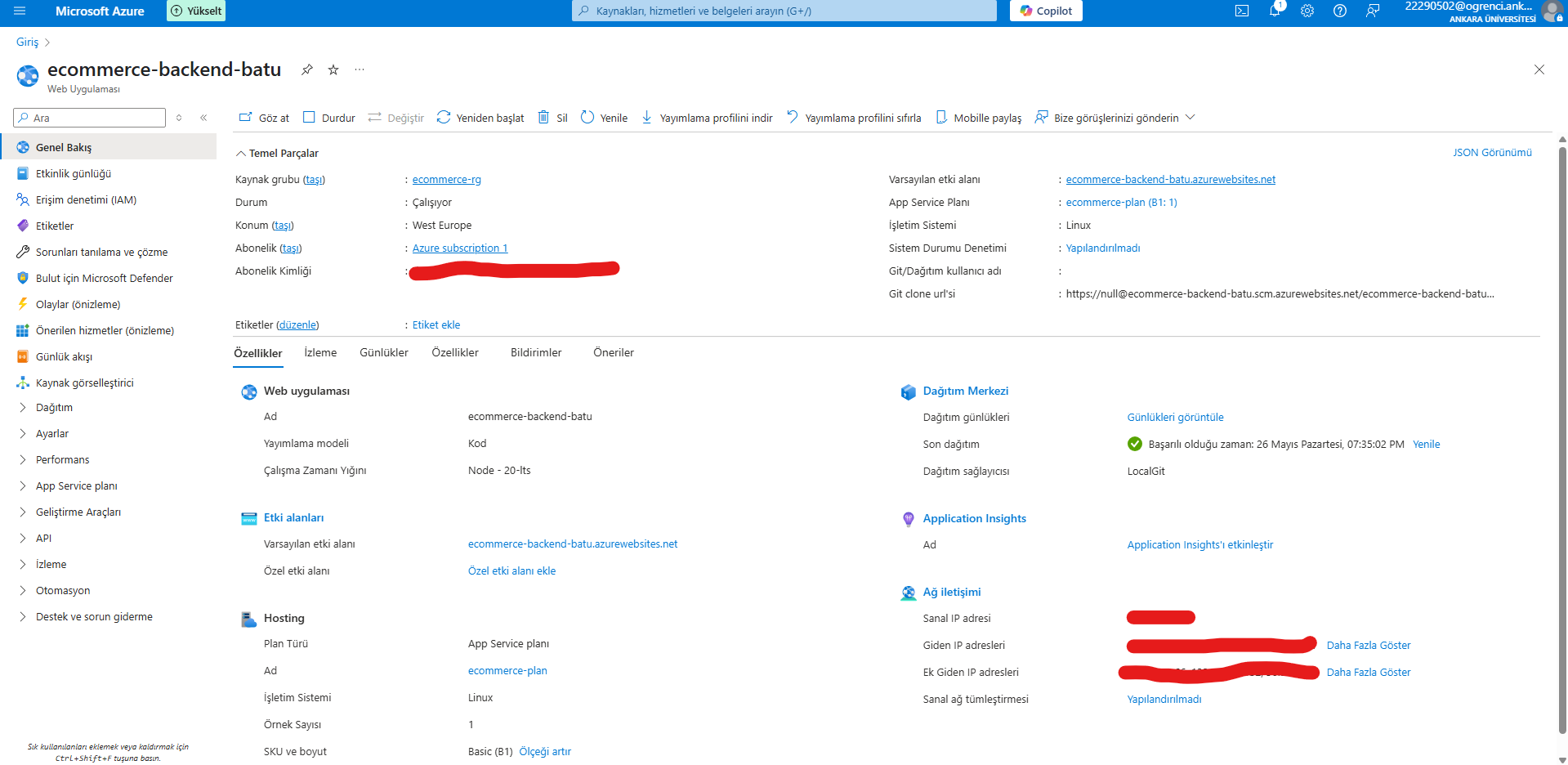
****

****

****

****

****

**BACKEND VE FRONTENDIN BULUTA BAĞLANMASI**

**Azure’da Backend Dağıtımı ve Buluta Bağlantı Süreci**

**1. Uygulama Oluşturma**

**Backend uygulamamı Node.js çalışma zamanı ile birlikte Azure App Service üzerinden oluşturdum. Uygulama Linux tabanlı ve çalıştığı ortamda Node 20 LTS sürümü kullanılmakta.**

**2. Kodun Azure’a Aktarılması (Deployment)**

**Kodu Azure’a yüklemek için Local Git dağıtım yöntemini kullandım. Azure portalinde oluşturulan git bağlantı adresi üzerinden:**

**git remote add azure https://<kullanıcı>@ecommerce-backend-batu.scm.azurewebsites.net/ecommerce-backend-batu.git**

**git push azure main**

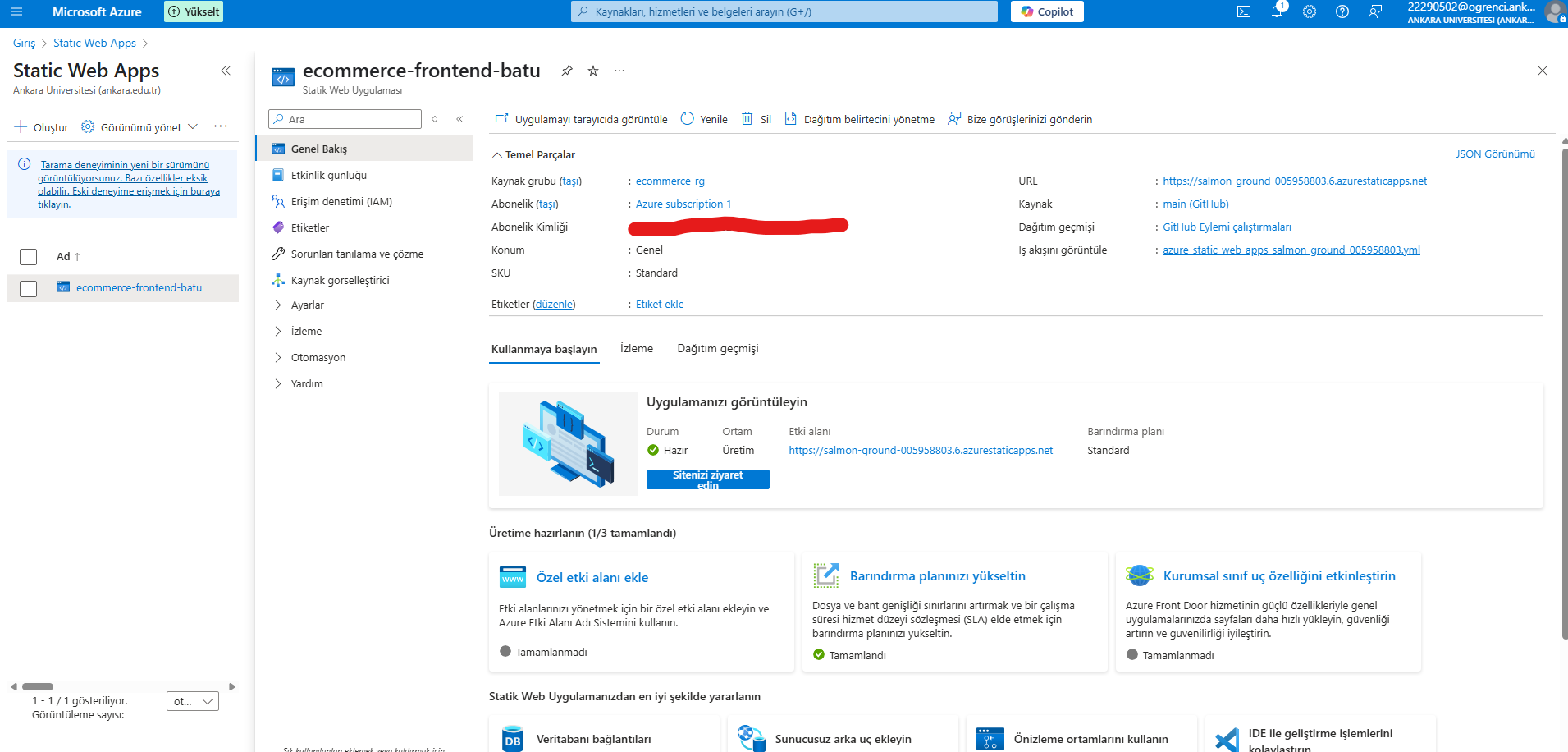
**komutları ile projemin tamamını Azure App Service'e gönderdim. Gönderimden sonra Azure otomatik olarak npm install ve npm start işlemlerini çalıştırarak uygulamayı başlattı.**

**3. Ortam Değişkenleri ve Veritabanı Bağlantısı**

**MongoDB bağlantısı .env dosyasında tanımlı olan MONGO\_URI değişkeni üzerinden sağlanmaktadır. Azure App Service içerisinde "Configuration" sekmesinden bu değişkeni gizli olarak tanımladım. Böylece uygulama MongoDB Atlas üzerinden veritabanına bağlanarak veri işlemlerini gerçekleştirebiliyor.**

**4. Dağıtım Durumu ve İzleme**

**Azure Portal üzerinden:**

* **Uygulamanın çalıştığı,**
* **Dağıtımın başarılı olduğu,**
* **LocalGit yöntemiyle yüklendiği,**
* **Çalışma zamanının ve logların izlenebilir olduğu bilgiler görülebilir.**

**Azure’da Frontend Yayınlama (Static Web App)**

**1. Uygulamanın Oluşturulması**

**React tabanlı frontend uygulamamı Azure Static Web Apps servisi üzerinden oluşturdum. Uygulama Azure portalinde oluşturulurken:**

* **Kaynak: GitHub**
* **Depo: ecommerce-frontend**
* **Dal: main**
* **Uygulama dizini: .**
* **Derleme çıktısı dizini: dist**

**olarak ayarlanmıştır.**

**2. Yayınlama Süreci**

**Azure, GitHub Actions aracılığıyla her push işleminde otomatik olarak frontend uygulamasını yeniden derleyip yayına alacak şekilde yapılandırılmıştır. Bu işlem GitHub Action iş akışı dosyası (azure-static-web-apps-\*.yml) üzerinden yürütülmektedir.**

**app\_location: "."**

**output\_location: "dist"**

**gibi ayarlarla Vite kullanan uygulama doğru şekilde build edilmiştir.**

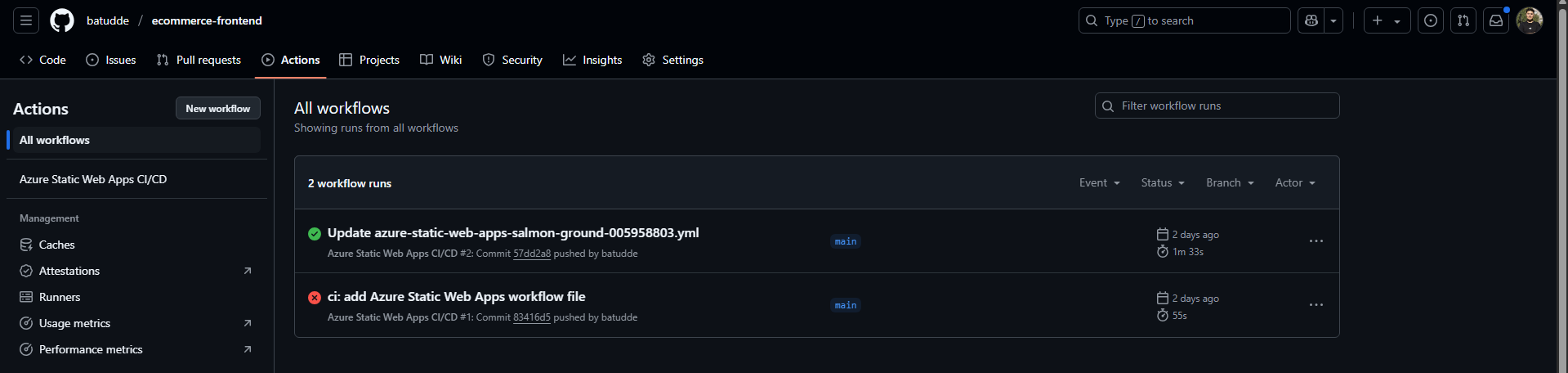
**3. Erişim Bilgileri ve Durum**

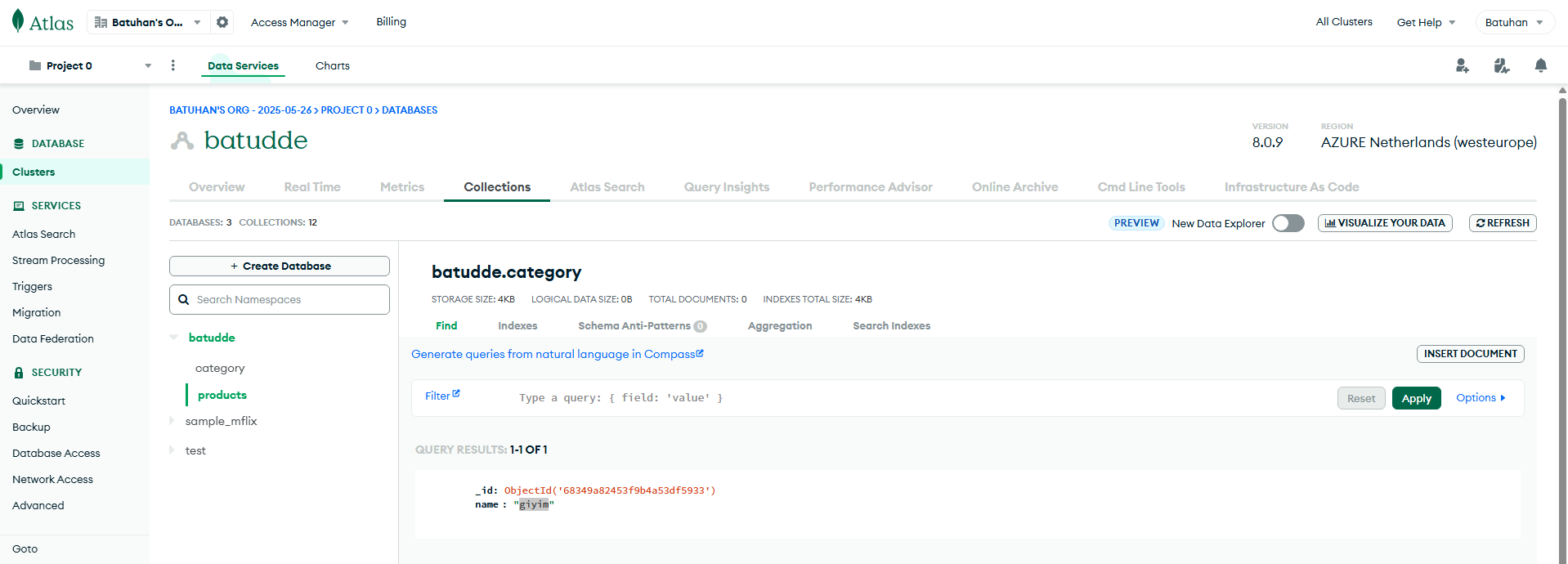
**Azure portalinde uygulama hazır ve çalışır durumda görünmektedir. Uygulamaya ait genel URL:**

**https://salmon-ground-005958803.6.azurestaticapps.net**

**adresidir ve tarayıcıdan herkese açık olarak erişilebilmektedir.**

**4. Veri Bağlantısı ve Backend Entegrasyonu**

**Frontend tarafındaki API çağrıları, ayrı olarak yayınlanan backend uygulamasına (https://ecommerce-backend-batu.azurewebsites.net/api/...) yönlendirilmiştir. Uygulama, backend verileriyle çalışacak şekilde konfigüre edilmiştir, ancak bazı sayfalarda veri çekme başarısızlıkları yaşanmıştır. Bu hatalar muhtemelen frontend kaynak kodunda API URL’lerinin tanımlandığı yerlerdeki eksik ya da yanlış kullanımlardan kaynaklanmaktadır.**

**VERİ TABANI DATABASE (MONGODB)**

**MongoDB Atlas Veritabanı ve Koleksiyon Oluşturma Süreci**

**1. MongoDB Atlas Kullanımı**

**Veri tabanımı MongoDB Atlas üzerinden oluşturdum. Atlas, bulut tabanlı MongoDB veritabanı hizmetidir ve Azure bölgesi olarak "AZURE Netherlands (westeurope)" seçilmiştir.**

**2. Veritabanı Oluşturma**

* **Veritabanı Adı: batudde  
  MongoDB Atlas arayüzünden *Create Database* butonuna tıklayarak yeni bir veritabanı oluşturdum. Bu veritabanı altında projeye özel koleksiyonlar tanımlanmıştır.**

**3. Koleksiyonlar (Collections)**

**Bu veritabanı içerisinde iki temel koleksiyon oluşturdum:**

* **products: Ürün verilerinin tutulduğu koleksiyon**
* **category: Ürünlerin ait olduğu kategorilerin tutulduğu koleksiyon**

**4. Veri Girişi**

**Kategoriler için örnek bir belge (document) şu şekilde eklenmiştir:**

**{**

**"\_id": ObjectId("68349a82453f9b4a53df5933"),**

**"name": "giyim"**

**}**

**Bu belgeyle, products koleksiyonundaki ürünler kategori alanında bu ObjectId’ye referans verir. Böylece MongoDB içinde ilişkilendirme kurulmuş olur.**

**5. Sonuç: MongoDB Atlas sayesinde backend sunucumun ihtiyaç duyduğu veriler bulutta güvenli ve esnek bir şekilde depolanmaktadır. API üzerinden bu verilere erişilerek frontend arayüzüne aktarılmaktadır.**