**الجمهورية العربية السورية**

**وزارة التعليم العالي والبحث العلمي**

**الجامعة الافتراضية السورية**

**ماجستير علوم الويب**

**مقرر Distributed Computation**

وظيفة مادة

Distributed Computation

نظام التجارة الإلكترونية الموزع

**الدكتورة المشرفة**

**د.سيرا أستور**

اعداد الطلاب :

|  |
| --- |
| MWS\_WDC\_HW1\_F24\_C1\_Bahaa\_314468 |
| MWS\_WDC\_HW1\_F24\_C2\_ Jilan\_339202 |
| MWS\_WDC\_HW1\_F24\_C2\_ Khaldoun\_332508 |

تطبيق موزع بتقنية الخدمات المصغرة Microservices

**معلومات المشروع**

اسم المشروع : نظام التجارة الإلكترونية الموزع باستخدام Microservices

التقنية المستخدمة : C# معSqlServer .NET 7

نوع المعمارية : Microservices Architecture

Ocelot: API Gateway

تاريخ الإنجاز : يونيو 2025

**إعدادات التشغيل**

تشغيل ملف run-system.bat الموجود في ملفات المشروع طبعا بعد تنصيب

1. .net 7
2. Sql server

**📋 ملخص المشروع**

تم تطوير نظام تجارة إلكترونية موزع يعتمد على معمارية الخدمات المصغرة (Microservices) باستخدام C# و.NET 7. يتكون النظام من أربع خدمات مستقلة مع بوابة API موحدة تدير التوجيه وموازنة التحميل.

🏗️ معمارية النظام

**الخدمات المصغرة الأربع:**

1. **خدمة المنتجات (Products Service) المنفذ: 5001**نفذ: 5001

المسؤوليات:

* إدارة كتالوج المنتجات
* عمليات CRUD للمنتجات
* إدارة المخزون
* البحث والتصفية

قاعدة البيانات: InMemory Database منفصلة

النقاط النهائية الرئيسية:

* GET /api/products - عرض جميع المنتجات
* POST /api/products - إضافة منتج جديد
* PUT /api/products/{id} - تحديث منتج
* DELETE /api/products/{id} - حذف منتج

1. **خدمة المستخدمين (Users Service) المنفذ: 5002: 5002**

المسؤوليات:

* إدارة حسابات المستخدمين
* التسجيل وتسجيل الدخول
* مصادقة JWT
* إدارة الملفات الشخصية

الأمان: تشفير كلمات المرور باستخدام BCrypt

النقاط النهائية الرئيسية:

* POST /api/users/register - تسجيل مستخدم جديد
* POST /api/users/login - تسجيل الدخول
* GET /api/users - عرض المستخدمين
* PUT /api/users/{id} - تحديث بيانات المستخدم

1. **خدمة الطلبات (Orders Service) المنفذ: 5003**

المسؤوليات:

* إدارة الطلبات
* تتبع حالة الطلبات
* إدارة عناصر الطلب
* ربط الطلبات بالمستخدمين والمنتجات

النقاط النهائية الرئيسية:

* GET /api/orders - عرض الطلبات
* POST /api/orders - إنشاء طلب جديد
* PATCH /api/orders/{id}/status - تحديث حالة الطلب

1. **خدمة الدفع (Payments Service) المنفذ: 5004**

المسؤوليات:

* معالجة المدفوعات
* إدارة طرق الدفع المختلفة
* تتبع حالة المعاملات
* عمليات الاسترداد

النقاط النهائية الرئيسية:

* POST /api/payments/process  - معالجة الدفع
  + - * + GET /api/payments/{id}  - عرض تفاصيل الدفع
* POST /api/payments/{id}/refund  - استرداد المبلغ

**API Gateway (Ocelot) - 🌐 التطبيق المفصل المنفذ:0500**

📁 ملفات Ocelot في المشروع:

* ملف التكوين : Gateway/ApiGateway/ocelot.json
* إعداد البرنامج : Gateway/ApiGateway/Program.cs
* حزمة NuGet:  Ocelot --version 20.0.0

الميزات المطبقة بالتفصيل:

1. **🔁 التوجيه (Routing)**

التطبيق في الكود:

* ملف ocelot.json : يحتوي على جميع مسارات التوجيه
* UpstreamPathTemplate : المسارات الواردة من العميل
* DownstreamPathTemplate: المسارات المرسلة للخدمات
* HTTP Methods : دعم GET, POST, PUT, DELETE, PATCH

أمثلة من الكود:

* "UpstreamPathTemplate": "/api/products/{everything}"
* "DownstreamPathTemplate": "/api/products/{everything}"
* "UpstreamHttpMethod": ["GET", "POST", "PUT", "DELETE", "PATCH"]

1. **⚖️موازنة التحميل (Load Balancing)**

التطبيق في الكود:

* خوارزمية Round Robin: مطبقة في كل route
* GlobalConfiguration : إعدادات عامة لموازنة التحميل
* LoadBalancerOptions: في كل خدمة منفصلة
* دعم عدة نسخ:  يمكن إضافة عدة منافذ لكل خدمة

أمثلة من الكود:

* "LoadBalancerOptions": { "Type": "RoundRobin" }
* "LoadBalancerOptions": { "Type": "RoundRobin", "Key": "LoadBalancer", "Expiry": 5000 }
* "DownstreamHostAndPorts": [{"Host": "localhost", "Port": 5001}]

**🔒 3. المصادقة والتوثيق (Authentication & Authorization)**

التطبيق في الكود:

* JWT Service : Services/UsersService/Services/JwtService.cs
* Token Generation:  إنشاء JWT tokens للمستخدمين
* Token Validation:  التحقق من صحة الـ tokens
* Claims-based Auth : استخدام Claims للمصادقة

أمثلة من الكود:

* public string GenerateToken(User user)
* public bool ValidateToken(string token)
* new Claim(ClaimTypes.NameIdentifier, user.Id.ToString())
* SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)

إعدادات JWT في appsettings.json:

"JwtSettings": { "SecretKey": "...", "Issuer": "ECommerceSystem", "Audience": "ECommerceUsers", "ExpirationMinutes": 60 }

**تطبيق الحماية في Controllers:**

* [Authorize] - للـ endpoints المحمية
* [AllowAnonymous] - للـ endpoints المفتوحة
* User.FindFirst(ClaimTypes.NameIdentifier) - للحصول على معرف المستخدم من Token
* app.UseAuthentication(); app.UseAuthorization(); - في Program.cs

**Endpoints محمية (تحتاج JWT Token):**

* GET /api/users - عرض جميع المستخدمين
* GET /api/users/me - بيانات المستخدم الحالي
* POST /api/products - إنشاء منتج جديد
* PUT /api/products/{id} - تحديث منتج
* DELETE /api/products/{id} - حذف منتج

**Endpoints مفتوحة (لا تحتاج Token):**

* POST /api/users/register - تسجيل مستخدم جديد
* POST /api/users/login - تسجيل الدخول
* GET /api/products - عرض المنتجات
* GET /api/products/{id} - عرض منتج محدد

**🔍 4. مراقبة الصحة (Health Checks)**

التطبيق في الكود:

* Health Endpoints: في كل خدمة منفصلة
* Gateway Health: فحص شامل عبر API Gateway
* Service Discovery: اكتشاف الخدمات المتاحة
* Status Monitoring: مراقبة حالة كل خدمة

أمثلة من الكود:

* app.MapGet("/health", () => new { Status = "Healthy", Service = "Products Service", Timestamp = DateTime.UtcNow })
* "UpstreamPathTemplate": "/health/products"
* "DownstreamPathTemplate": "/health"

**⚙️ 5. إعدادات إضافية في Ocelot**

ميزات أخرى مطبقة:

* Rate Limiting: تحديد معدل الطلبات
* CORS Support: دعم الطلبات من مصادر مختلفة
* Error Handling: معالجة الأخطاء المتقدمة
* Logging: تسجيل مفصل للعمليات

أمثلة من الكود:

* "RateLimitOptions": { "ClientIdHeader": "ClientId", "QuotaExceededMessage": "تم تجاوز الحد المسموح من الطلبات" }
* builder.Services.AddCors(options => { options.AddPolicy("AllowAll", policy => { policy.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader(); }); });

🔧 التقنيات والأدوات المستخدمة

**Backend Framework :**

* .NET 7 :  الإطار الأساسي
* ASP.NET Core Web API :  لبناء REST APIs
* Entity Framework Core :  لإدارة قواعد البيانات
* InMemory Database : لقواعد البيانات التجريبية

**API Gateway:**

* Ocelot : لإدارة API Gateway
* Load Balancing :  موازنة التحميل
* Rate Limiting :  تحديد معدل الطلبات

**الأمان :**

* JWT : للمصادقة
* BCrypt : لتشفير كلمات المرور
* CORS : للسماح بالطلبات من مصادر مختلفة

**التوثيق والاختبار:**

* Swagger/OpenAPI : لتوثيق APIs
* Swagger UI : لاختبار النقاط النهائية

**📊 النماذج والبيانات المشتركة**

**المكتبة المشتركة (SharedModels):**

* Product:  نموذج المنتج
* User:  نموذج المستخدم
* Order & OrderItem:  نماذج الطلبات
* Payment: نموذج الدفع
* DTOs: كائنات نقل البيانات
* ApiResponse : نموذج الاستجابة الموحد

**🚀 ميزات النظام المطبقة**

1. **الفصل الكامل بين الخدمات:**

* كل خدمة لها قاعدة بيانات منفصلة
* استقلالية كاملة في التطوير والنشر
* إمكانية التوسع المستقل

1. **التواصل غير المتزامن:**

* استخدام HTTP REST APIs
* معالجة الأخطاء المتقدمة
* إعادة المحاولة والتعافي

1. **الأمان المتقدم:**

* مصادقة JWT شاملة
* تشفير كلمات المرور
* حماية النقاط النهائية

1. **المراقبة والصحة:**

* Health Checks لجميع الخدمات
* سجلات مفصلة (Logging)
* معالجة الأخطاء المركزية

**🔄 تدفق العمليات**

1. تسجيل مستخدم جديد:

Client → API Gateway → Users Service → Database → Response

1. إنشاء طلب:

Client → API Gateway → Orders Service → Products Service (للتحقق) → Database → Response

1. معالجة الدفع:

Client → API Gateway → Payments Service → Orders Service (للتحديث) → Response

**📈 الفوائد المحققة**

1. **قابلية التوسع:**

* إمكانية توسيع كل خدمة بشكل مستقل
* دعم الأحمال العالية
* توزيع الموارد بكفاءة

1. **المرونة:**

* سهولة إضافة خدمات جديدة
* تحديث الخدمات بدون تأثير على الأخرى
* دعم تقنيات مختلفة لكل خدمة

**3. الموثوقية:**

* فشل خدمة واحدة لا يؤثر على الباقي
* آليات التعافي والإعادة
* مراقبة مستمرة للصحة

**🧪 الاختبار والتشغيل**

**متطلبات التشغيل:**

* .NET 7.0 SDK
* Visual Studio 2022 أو VS Code
* 5 منافذ متاحة (5000-5004)

**خطوات التشغيل:**

1. تشغيل ملف تشغيل\_النظام .bat
2. أو تشغيل كل خدمة يدوياً
3. الوصول عبر API Gateway على المنفذ 5000

**الاختبار:**

* Swagger UI متاح لكل خدمة
* نقاط نهائية للفحص الصحي
* بيانات تجريبية محملة مسبقاً

**📝 الخلاصة**

تم تطوير نظام تجارة إلكترونية موزع متكامل يطبق أفضل ممارسات معمارية الخدمات المصغرة. النظام يوفر:

✅ أربع خدمات مصغرة مستقلة

 API Gateway ✅ مع Ocelot

✅ موازنة التحميل والتوجيه

✅ مصادقة JWT شاملة

✅ قواعد بيانات منفصلة

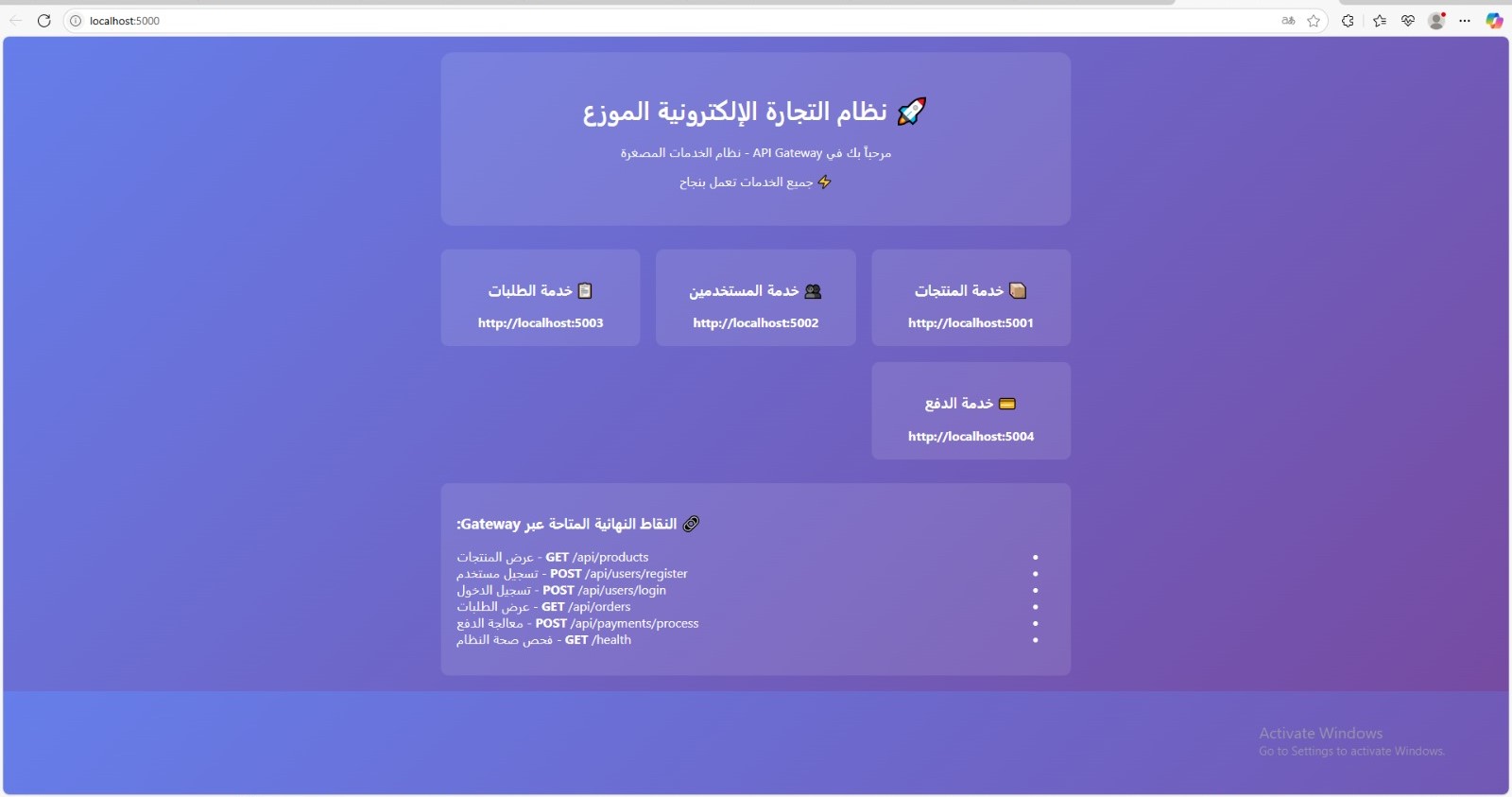
✅ مراقبة الصحة والأداء

✅ توثيق شامل مع Swagger

✅ سهولة التشغيل والاختبار

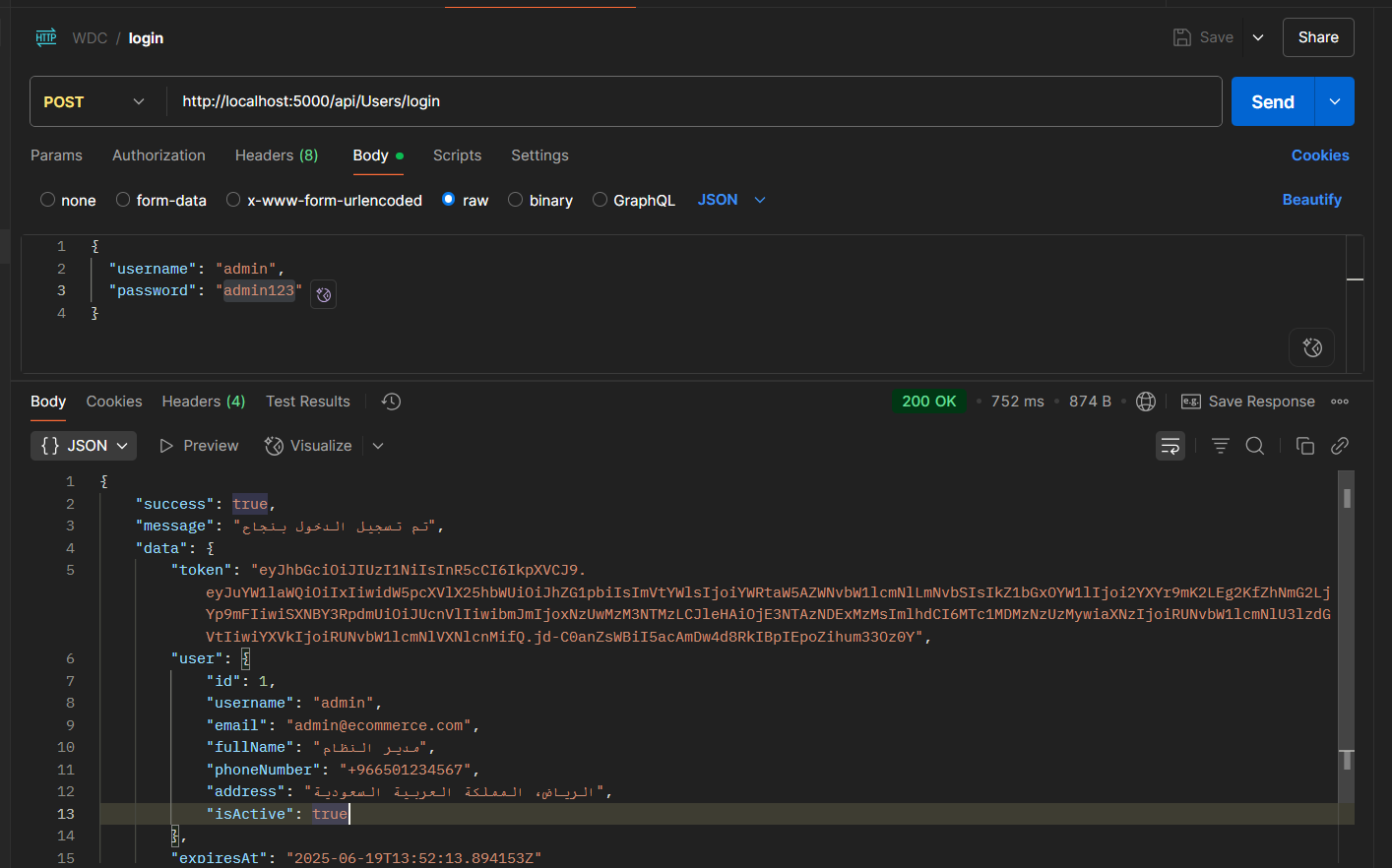
النظام جاهز للاستخدام ويمكن توسيعه بسهولة لإضافة المزيد من الخدمات والميزات.

تم إنجاز المشروع بنجاح وفقاً لجميع المتطلبات المحددة.

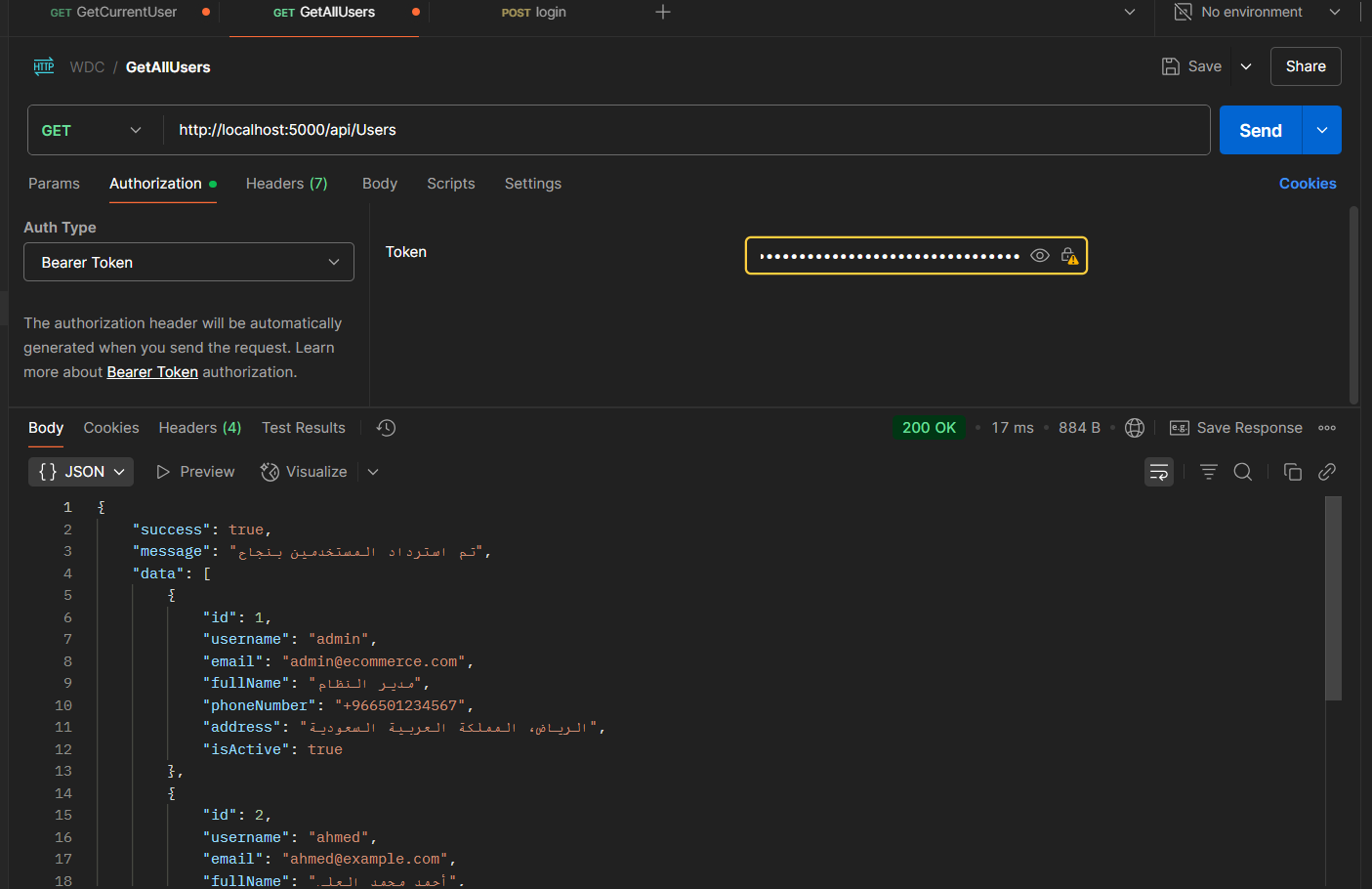


**عرض مقتطفات عن المشروع**

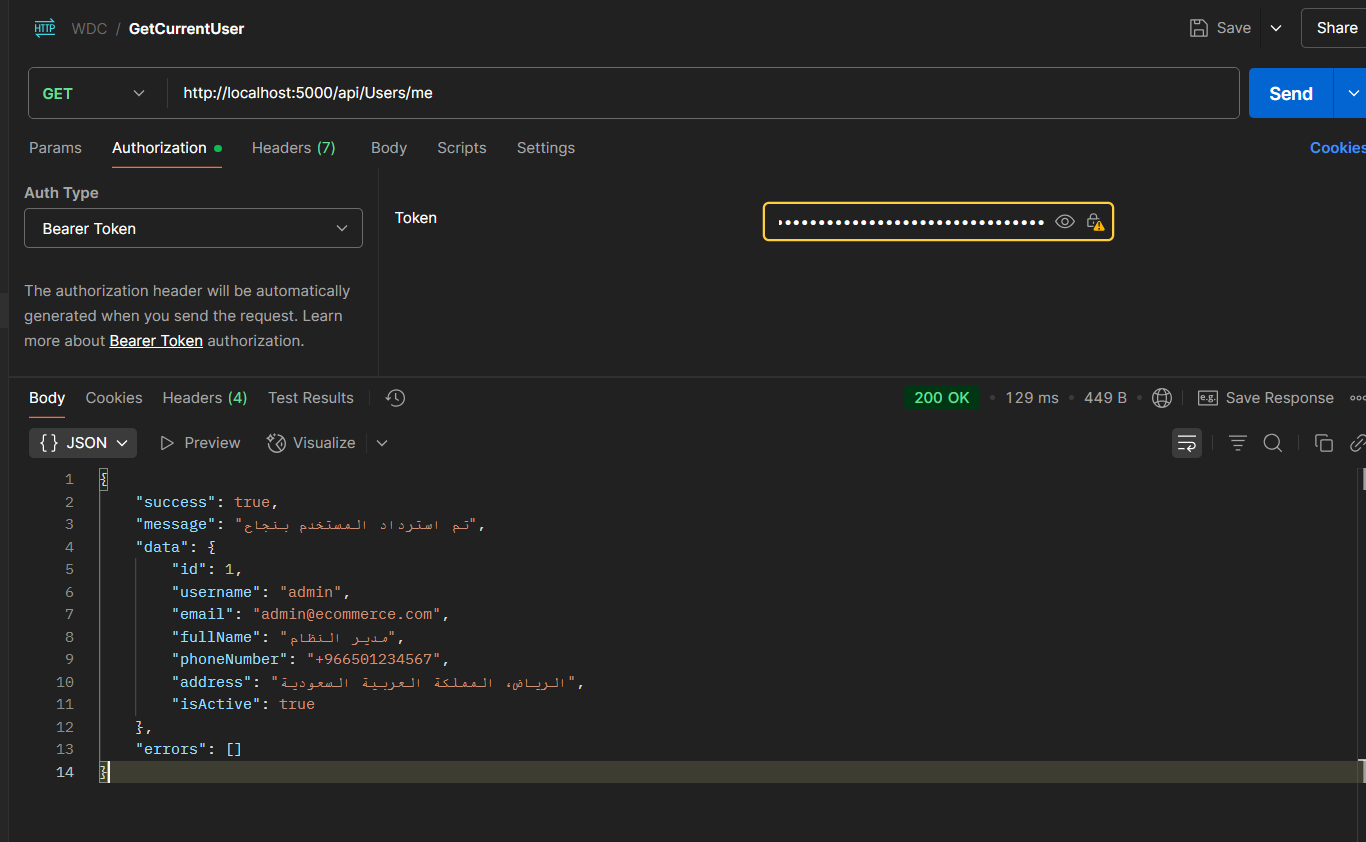
1. **Login**

****

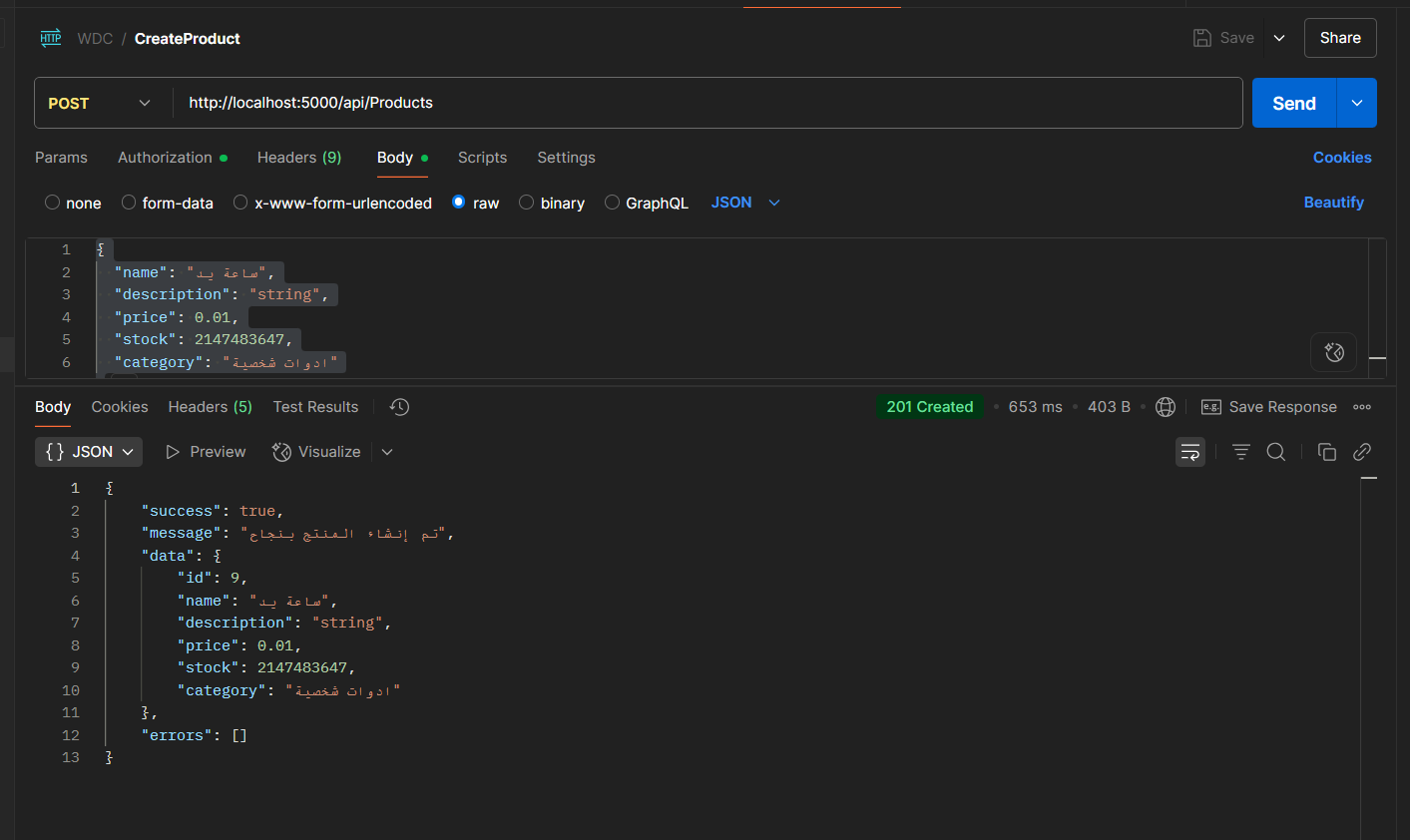
1. **GetAllUser**



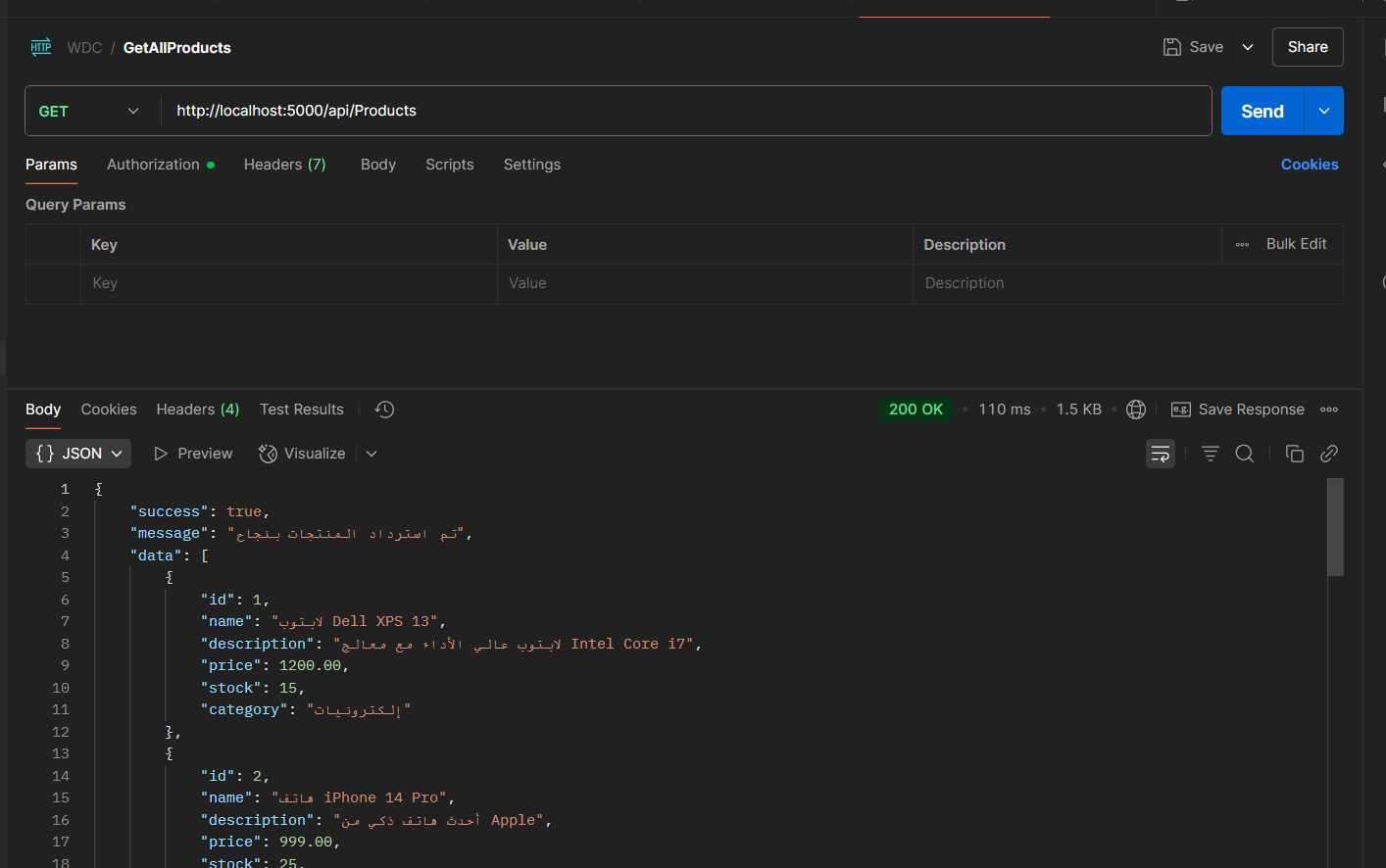
1. **GetCurrentUser**



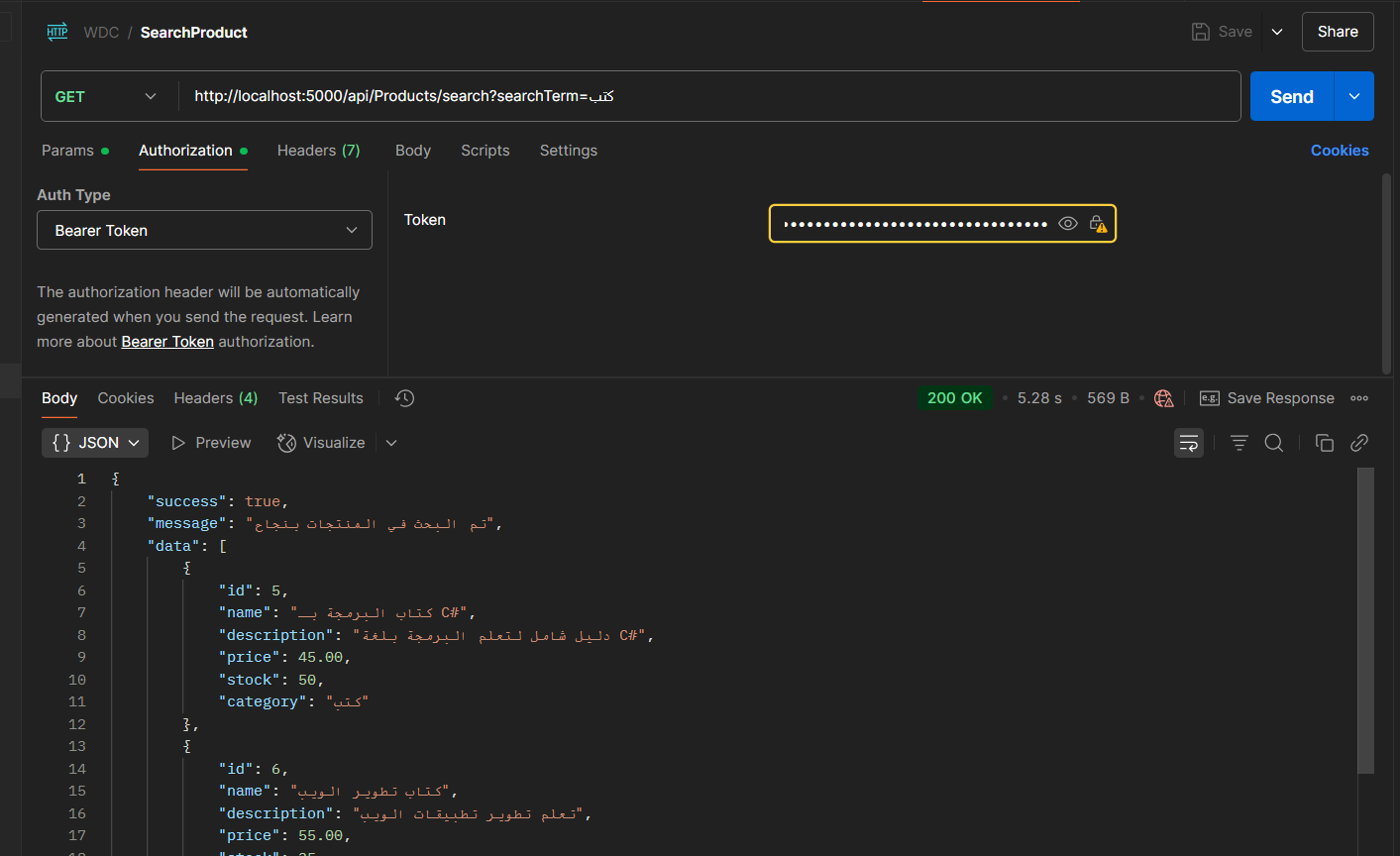
1. **CreateProduct**

****

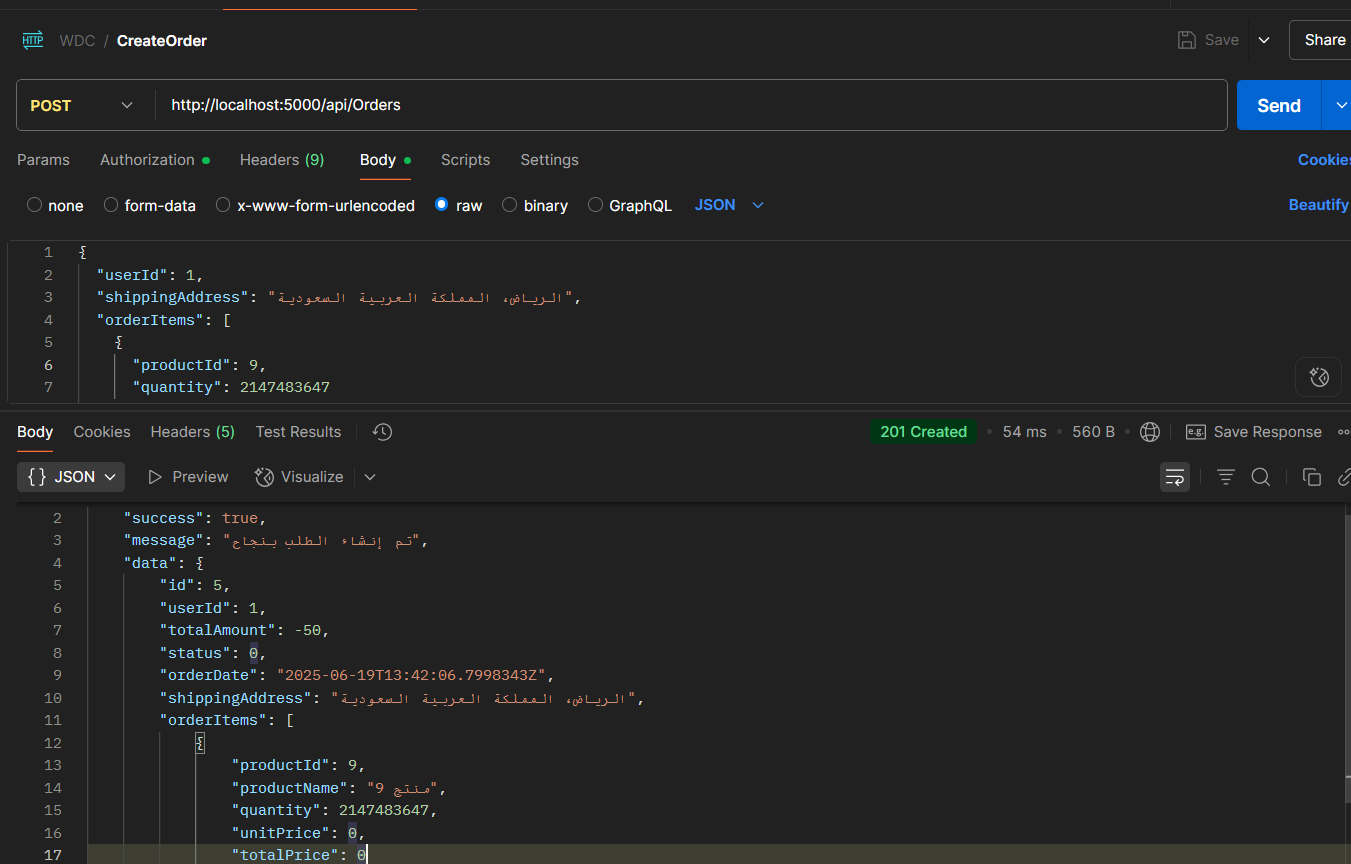
1. **GetAllProducts**

****

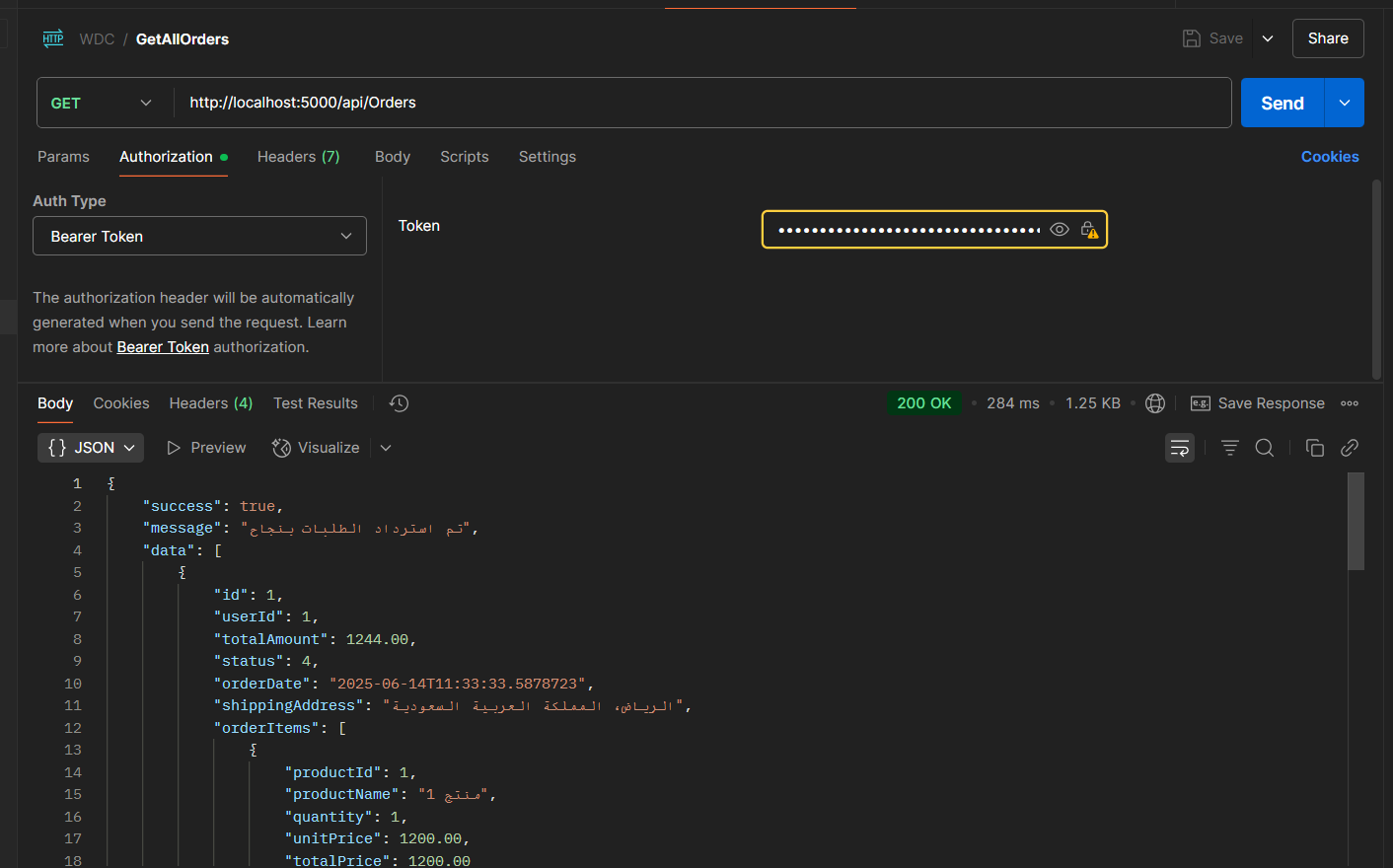
1. **SearchProduct**

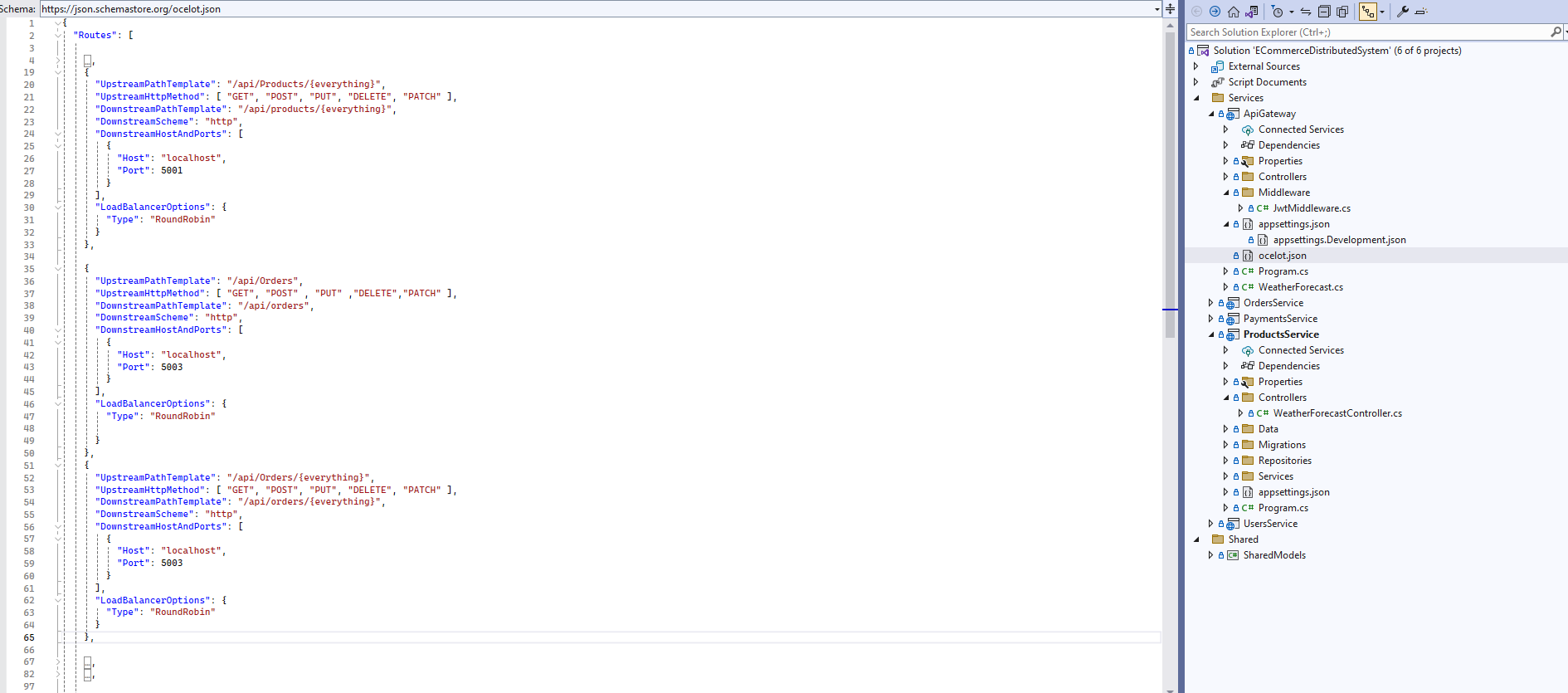
****

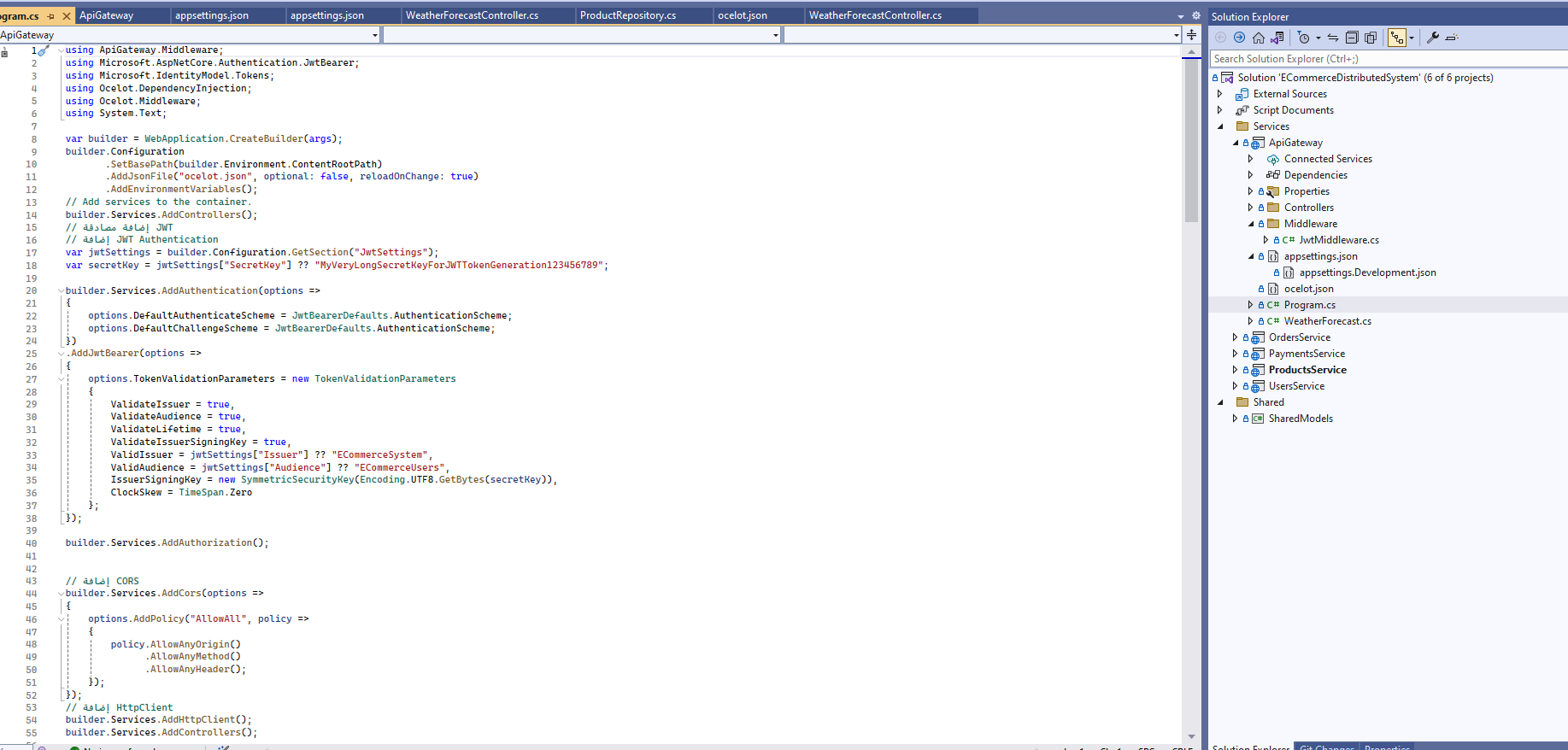
1. **CreateOrder**

****

1. **GetAllOrders**

****

1. **Ocelot.json**
2. **JWT Token**

****