🎓 Microservice Eğitim Programı

🧠 1. Microservice Nedir?

Microservice (mikro servis), büyük ve karmaşık bir uygulamayı küçük, bağımsız çalışan servisler halinde bölerek geliştirme yaklaşımıdır.
Her servis kendi veritabanına ve iş mantığına sahiptir, bağımsız olarak deploy edilir, güncellenebilir ve ölçeklenebilir.

🛠️ 2. Microservice Nasıl Tasarlanır?

- Her işlevsel alan için ayrı servis oluşturulur (örnek: CartService, OrderService)
- Her servis:
  - Kendi veritabanına sahiptir (örnek: MSSQL, PostgreSQL, MongoDB)
  - Kendi API'sini sağlar veya event üretir
  - Tek başına build/deploy edilir
- Servisler genellikle REST, gRPC ya da mesajlaşma sistemleriyle (event-driven) haberleşir

📌 Bounded Context Nedir?

Bounded Context, DDD (Domain-Driven Design) yaklaşımında bir kavramın kendi sınırları içerisinde net bir şekilde tanımlandığı bölgedir. Her bounded context, bağımsız bir microservice olarak modellenebilir. Örneğin:

- `Cart` yalnızca sepet işlemleriyle ilgilenir
- `Order` yalnızca siparişle ilgilenir

☑️ 3. Microservice Mimarisi – Artıları (Avantajları)

1. **Servisler Arası Bağımsızlık** Her servis bağımsız geliştirilip deploy edilebilir. Bir servisteki değişiklik diğerlerini etkilemez.
2. **Teknoloji Özgürlüğü** Servisler farklı programlama dilleri veya veritabanları kullanabilir (örneğin: Cart → MSSQL, Order → PostgreSQL)
3. **Takım Dağılımı Kolaylığı** Ekipler farklı servisler üzerinde paralel çalışabilir.
4. **Bağımsız Derleme ve Yayınlama** Tek bir servis değiştiğinde tüm sistemi yeniden build etmeye gerek kalmaz.
5. **Yatay Ölçeklenebilirlik** İhtiyaç duyulan servisler bağımsız şekilde çoğaltılabilir.
6. **Hızlı CI/CD** Her servis için ayrı build/test/release pipeline tanımlanabilir.
7. **Domain Odaklı Geliştirme** Servisler iş ihtiyaçlarına göre tasarlanır, teknik yapılara göre değil.

❌ 4. Microservice Mimarisi – Eksileri (Zorluklar)

1. **Servisler Arası Haberleşme Karmaşıklığı** API çağrıları, kuyruk sistemleri, retry, fallback gibi yapılar gerekir.
2. **Transaction Yönetimi Karmaşıklaşır** Farklı veritabanlarında işlem yapılırken klasik transaction yapılamaz. SAGA Pattern gibi yaklaşımlar gerekir.
3. **Geliştirme ve Test Ortamı Karmaşıktır** Tüm servislerin localde ayağa kalkması için Docker, docker-compose gibi çözümler gerekir.
4. **Logging & Monitoring Gereksinimi** Merkezi loglama (Elastic, Loki), izleme (Prometheus, OpenTelemetry) altyapıları gerekir.
5. **Gecikme ve Bağlantı Hataları** Her şey network üzerinden döndüğü için gecikme ve hata yönetimi önemlidir. Circuit breaker gibi yapılar kullanılmalıdır.
6. **Deployment Süreci Daha Yönetilmesi Gerekir** Servislerin ayrı ayrı versiyonlanması ve dağıtımı için CI/CD orkestrasyonu gerekir (örnek: Kubernetes).

🔁 5. Microservice vs Modular Monolith

Özellik	Microservice	Modular Monolith
Deploy	Her servis bağımsız deploy edilir	Tek deploy paketi
Veritabanı	Her servis kendi DB'sine sahiptir	Ortak veritabanı kullanılır
İletişim	API / Event (dış iletişim)	Metot (in-process)
Ekip Yapısı	Her ekip farklı servislerde çalışabilir	Genelde aynı ekip birlikte geliştirir
Performans	Yatay ölçeklenebilir	Local hızlı çalışır
Karmaşıklık	Fazladır (queue, fallback, saga)	Daha sadedir
Test	Servisler bağımsız test edilebilir	Tüm yapı birlikte test edilir

---

📚 6. Eğitim İçeriği

1. Microservice Nedir
2. Nasıl Tasarlanır?
3. Artıları ve Eksileri
4. Modular Monolith ile Farkları
5. Microservice Projesi Oluşturalım
   - .NET tabanlı örnek proje: CartService, OrderService
6. MSSQL ve PostgreSQL Bağlantısı

- Her servis kendi DB'sine bağlanacak şekilde yapılandırılır

7. Gateway Nedir?

- API Gateway, tüm servisleri tek bir giriş noktasından yönlendiren yapıdır

8. Gateway Kütüphaneleri

- Ocelot
- YARP (Yet Another Reverse Proxy)

🔁 9. Load Balancing (Yük Dengeleme)

🧱 Tanım

Load balancing, gelen istekleri birden fazla servis örneği (instance) arasında dengeli dağıtarak sistemin yükünü azaltan yapıdır.

📦 Ocelot ile Load Balancing

Ocelot'ta bir route için birden fazla `DownstreamHostAndPorts` tanımlarsan, Ocelot bu hedefler arasında istekleri dağıtır.

Örnek:

```
"DownstreamHostAndPorts": [
  { "Host": "localhost", "Port": 5001 },
  { "Host": "localhost", "Port": 5002 }
],
"LoadBalancerOptions": {
  "Type": "RoundRobin"
}
```

Desteklenen türler:

- `RoundRobin`
- `LeastConnection`
- `NoLoadBalancer`

📦 YARP ile Load Balancing

YARP, daha gelişmiş algoritmalar ve özelleştirme sunar:

- Round Robin
- Least Requests
- Random
- Power of Two Choices
- Özelleştirilebilir algoritmalar

🔁 Ocelot vs YARP Load Balancing

Özellik	Ocelot	YARP
Konfigürasyon	`ocelot.json` içinde `DownstreamHostAndPorts` ve `LoadBalancerOptions` tanımlanır	C# kodu veya `appsettings.json` içinde `Clusters` ve `LoadBalancing` ayarlarıyla tanımlanır
Algoritmalar	- RoundRobin - LeastConnection - NoLoadBalancer	- RoundRobin - LeastRequests - Random - PowerOfTwoChoices - Özel algoritmalar (custom)
Özelleştirilebilirlik	Sınırlı; desteklenen türlerle kısıtlı	Çok yüksek; kendi `ILoadBalancingPolicy` implementasyonu eklenebilir
Performans	Orta düzey	Yüksek performans, daha modern ve optimize edilmiş
Health Checks	Harici health-check middleware ile eklenmeli	Dahili health-check ve dynamic cluster management desteği
Yönetim & Gözlemleme	Temel logging ve metric toplama	Gelişmiş telemetry entegrasyonları (OpenTelemetry, Prometheus)
Kullanım Kolaylığı	Basit küçük projeler için hızlı kurulabilir	Büyük ve karmaşık senaryolar için daha esnek ve güçlü, fakat başlangıçta biraz daha konfigürasyon gerektirebilir

🔐 10. Authentication - Authorization

- **Authentication**: JWT token ile kimlik doğrulama genelde gateway'de yapılır
- **Authorization**: Her servis, kendi scope/role kontrolünü kendi içinde yapar Örneğin: `products.read`, `orders.write`

💥 11. Transaction Yönetimi

➕ SAGA Pattern

- Dağıtık transaction'lar için en çok tercih edilen yaklaşımdır
- İki türü vardır:
  - **Orchestration**: Merkezi bir SAGA yöneticisi (coordinator) adımları sırayla kontrol eder
  - **Choreography**: Servisler event yayınlar ve birbirlerinin event'lerine tepki verir

📢 Event-Driven Communication

- Servisler arasında olay (event) yayınlayarak asenkron iletişim sağlar
- MassTransit, RabbitMQ, Kafka gibi araçlar kullanılır
- Eventual consistency (sonunda tutarlılık) modeli üzerine kuruludur

Circuit Breaker

- Sürekli hata veren servislere yapılan çağrıları geçici olarak durdurur
- Polly gibi kütüphaneler ile `retry`, `timeout`, `fallback` işlemleri uygulanır
- Sistem dayanıklılığını (resilience) artırır

Kullanacağımız Kütüphaneler ve Nedenleri

Kütüphane	Kategori	Kısa Açıklama
ASP.NET Core Web API	Web Framework	Hafif, performanslı ve genişletilebilir HTTP API'ler oluşturmak için Microsoft'un resmi çatısı.
Microsoft.EntityFrameworkCore.SqlServer	ORM / MSSQL Provider	CartService gibi MSSQL kullanan servislerde veritabanı operasyonlarını kolaylaştırır.
Npgsql.EntityFrameworkCore.PostgreSQL	ORM / PostgreSQL	OrderService gibi PostgreSQL kullanan servislerde EF Core üzerinden veri erişimini sağlar.
MassTransit	Message Bus	SAGA Pattern ve event-driven iletişim için soyutlamalı bir publish/subscribe altyapısı sunar.
RabbitMQ.Client	Message Broker Client	MassTransit'in üzerinde çalıştığı, güvenilir kuyruk sistemi (messaging broker) sağlar.
Ocelot	API Gateway	Basit konfigürasyonla reverse-proxy, routing ve temel load-balancing ihtiyaçlarını karşılar.
YARP	Reverse Proxy	Microsoft destekli, esnek routing ve gelişmiş load-balancing algoritmaları sunar.
Microsoft.AspNetCore.Authentication.JwtBearer	Authentication	Gateway ya da servis düzeyinde gelen JWT'leri doğrulamak ve kullanıcı kimliğini çözmek için.
Duende IdentityServer	Identity Provider	OAuth2 / OpenID Connect ile merkezi kimlik doğrulama ve token yönetimi sunar.
Keycloak	Identity & Access Management	Açık kaynaklı IAM çözümü; OAuth2, OpenID Connect ve SAML protokollerini destekler, kullanıcı yönetimi sağlar.

Kütüphane	Kategori	Kısa Açıklama
Polly	Resilience / Fault Handling	Retry, timeout, circuit-breaker, fallback gibi dayanıklılık (resilience) politikalarını uygular.
Serilog	Logging	Yapılandırılabilir, sink-destekli ve performanslı uygulama log'ları için popüler logging kütüphanesi.
OpenTelemetry	Tracing & Metrics	Dağıtık izleme (distributed tracing) ve metrik toplama altyapısı sağlayarak servisleri gözlemler.
Swashbuckle.AspNetCore	API Documentation	Swagger UI ve OpenAPI tanımı ile servislerin self-documenting olmasını sağlar.
AutoMapper	Object Mapping	DTO ↔ Entity dönüşümlerini konfigürasyon bazlı, hızlı ve hatasız yapmaya yardımcı olur.
FluentValidation	Validation	Zengin bir doğrulama API'si ile istek modelleri için kuralları açık ve yeniden kullanılabilir tanımlar.