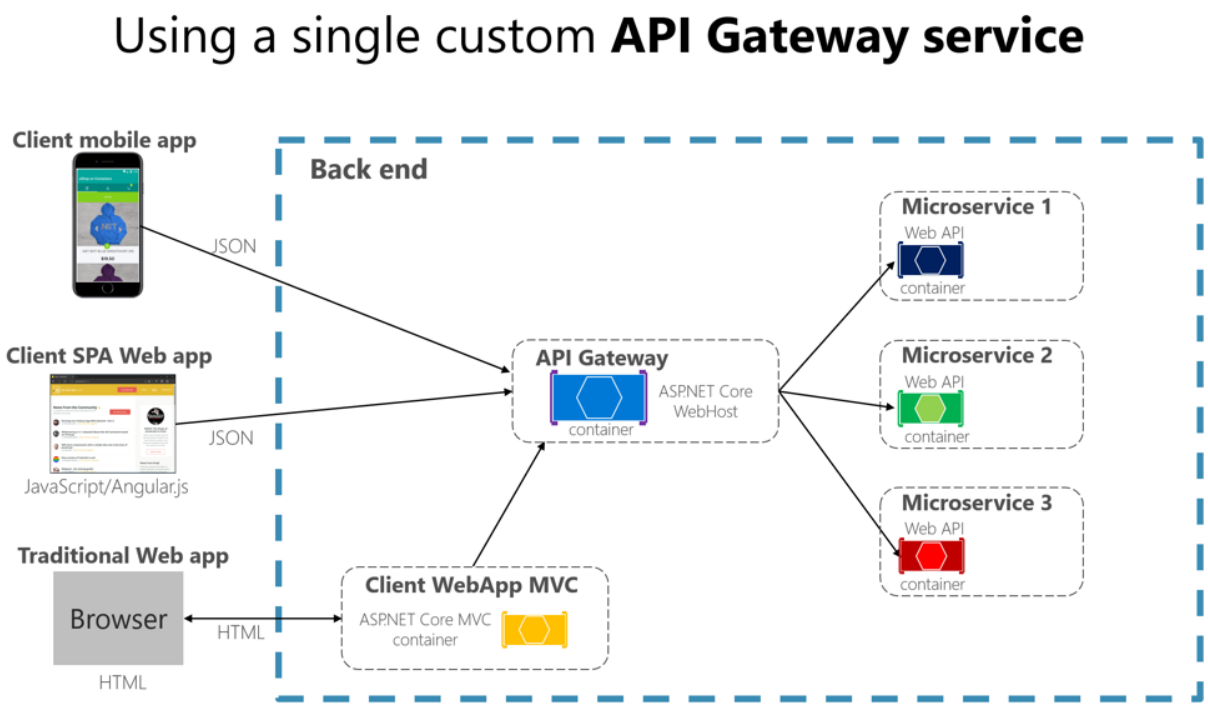
WEBMTS Mikroservis Dönüşümü Çalışması

1. Giriş

WEBMTS, Western Union'ın Türkiye'deki asıl acentesi olan "BPN Danismanlik A.S." şirketi tarafından kullanılan kapsamlı bir para transfer ve finans işlemleri yönetim sistemidir. Bu web tabanlı uygulama, acente çalışanlarının müşterilerle etkileşimlerini ve çeşitli finansal işlemleri yönetmelerini sağlar.

Şu anda bu sistem monolitik bir yapıda çalışmakta olup, tüm işlevler tek bir kod tabanında bulunmaktadır. Bu durum ölçeklendirme, bakım ve geliştirme süreçlerinde zorluklar yaratmaktadır. WEBMTS dönüşüm projesinin amacı, mevcut monolitik web uygulamasını modern **mikroservis mimarisine** dönüştürmektir.

Bu dönüşüm ile uygulamanın ölçeklenebilirliği, sürdürülebilirliği ve bağımsız geliştirilebilirliği artırılacaktır. Proje kapsamında, adaydan mevcut sistemdeki ana işlevleri ayrıştırarak farklı mikroservisler şeklinde yeniden geliştirmesi beklenmektedir. Her bir mikroservis, belirli bir iş sorumluluğunu üstlenecek ve diğer servislerle iyi tanımlanmış arabirimler üzerinden iletişim kuracaktır.



Şekil: API Gateway kullanan basit bir mikroservis mimarisi örneği. Bu diyagram, istemcilerin (mobil, web uygulaması vb.) tek bir giriş noktasından (API Gateway) geçerek arka plandaki farklı mikroservislere nasıl yönlendirildiğini göstermektedir. Mikroservislerin her biri ayrı bir konteyner içinde çalışmakta ve belirli bir işi gerçekleştirmektedir.

Yukarıdaki şekilde görüldüğü gibi mikroservis mimarisinde istemciler doğrudan arka uçtaki servislere erişmek yerine bir API Gateway veya benzeri bir düzenleyici katmandan geçer. Bu sayede istemci uygulamalar birden çok servisin uç noktasını bilmek zorunda kalmaz ve istekler tek bir noktada toplanarak ilgili mikroservislere yönlendirilir. Her mikroservis kendi veritabanına sahip olup bağımsız olarak ölçeklenebilir ve dağıtılabilir. Adaydan, bu prensipler doğrultusunda WEBMTS'nin temel bileşenlerini mikroservislere ayırması ve gerekli entegrasyonu sağlaması beklenmektedir.

Adaydan Beklenenler: Bu proje kapsamında aday, mikroservis mimarisinin tasarımı ve gerçekleştirilmesinden sorumlu olacaktır. Beklenen; mevcut sistemi analiz ederek uygun servis sınırlarını (bounded context) belirlemesi, her bir mikroservisi ilgili işlevleriyle geliştirmesi, servislerin birbiriyle iletişimini kurması ve tüm bunları çalışır halde sunabilmesidir. Dönüşüm sonucunda sistemin mevcut işlevselliğinin korunması ve mümkünse iyileştirilmesi hedeflenmektedir.

2. İşlevsel Gereksinimler

Bu bölüm, adayın geliştirmesi gereken mikroservisleri ve her birinin sağlaması gereken ana işlevleri tanımlamaktadır. WEBMTS'nin iş alanları belirli alt parçalara ayrılarak her parça için bağımsız servisler oluşturulacaktır. Her bir mikroservis kendi RESTful API'sini sunacak ve sadece kendi sorumluluk alanıyla ilgili işlemleri yürütecektir. Aşağıda, geliştirilmesi öngörülen mikroservisler ve temel işlevleri listelenmiştir:

- **Kullanıcı Yönetimi Servisi:** Sistem kullanıcılarının kimlik doğrulama ve yetkilendirme işlemlerini yönetir. Yeni kullanıcı oluşturma, giriş yapma (authentication), oturum yönetimi ve rol bazlı yetkilendirme kontrollerini sağlar. Örnek API uç noktaları: /api/users (CRUD işlemleri), /api/auth/login (giriş) ve /api/auth/validate (token doğrulama).
- **Müşteri/Kayıt Servisi:** WEBMTS'nin temel domain nesnelerini yönetir. Temel müşteri bilgilerini tutacak, oluşturma/güncelleme/silme ve sorgulama işlemlerini sağlayacaktır. Temel API uç noktaları: /api/customers altında POST (yeni ekleme), GET (listeleme ve detay), PUT (güncelleme), DELETE (silme) işlemleri.
- **İşlem/Servis A (Domain Servisi):** Monolitik sistemdeki belirli bir iş sürecini üstlenecek mikroservistir. (Örneğin para transferi, fatura ödeme vb). Kendi içinde gerekli iş mantığını barındırır ve ilgili veritabanı tablolarına erişimi vardır. Dışarıya REST API aracılığıyla hizmet verir. Örnek API uç noktaları: /api/sendmoney veya /api/requests gibi, ilgili işlem türüne göre kayıt oluşturma, durum güncelleme, listeleme vb. fonksiyonlar.
- **Bildirim Servisi:** Sistem içi bildirim ve haberleşme işlevlerini üstlenir. Farklı işlemler gerçekleştiğinde (örneğin yeni müşteri eklendiğinde veya bir işlemin durumu değiştiğinde)

bildirim servisi e-posta, SMS veya sistem içi bildirim oluşturabilir. Bu servis, diğer mikroservislerden gelen bildirim tetikleyici olayları dinleyebilir (Opsiyonel olarak bir mesaj kuyruğu üzerinden) ve uygun aksiyonu alır. Tipik API uç noktaları: /api/notifications/send (manuel tetikleme için) vb. Ayrıca event-driven mimari kullanılırsa, servis bir mesaj kuyruğuna abonelikle çalışabilir.

- **API Gateway (veya BFF) (İsteğe bağlı):** İstemcilerin tüm mikroservislere tek bir noktadan erişebilmesi için bir API Gateway veya *Backend for Frontend* katmanı öngörülür. Bu bileşen, dış dünyaya tek bir REST API sunar ve gelen isteklere göre ilgili mikroservise çağrı yapar. Ayrıca ortak konular (kimlik doğrulama, rate limiting, yük dengeleme) burada ele alınabilir. Aday bu katmanı kendi uygulaması içinde basitçe geliştirebilir veya mevcut bir API Gateway çözümü kullanabilir. API Gateway, örneğin /api/* yoluna gelen çağrıları ilgili servislere yönlendirecektir (örn. /api/customers/* -> Müşteri Servisi, /api/sendmoney/* -> İşlem Servisi vb.).

Her bir mikroservis kendi görev alanıyla sınırlı kalmalıdır. Mikroservisler arası veri paylaşımı gerekiyorsa, bu paylaşım servislerin API'leri üzerinden yapılacak çağrılarla veya gerekliyse bir mesajlaşma altyapısıyla (örneğin RabbitMQ, Kafka) gerçekleştirilmelidir. İşlevsel gereksinimler doğrultusunda, her servis için gereken tüm senaryolar (normal akışlar, hata durumları) aday tarafından ele alınmalıdır. Adayın, servislerin sorumluluklarını net çizgilerle ayırması ve gereksiz bağımlılıkları önlemesi beklenir.

3. Teknik Gereksinimler

Bu kısımda, projenin gerçekleştirilmesi sırasında uyulması beklenen teknik koşullar ve kullanılacak teknolojilere dair beklentiler listelenmiştir. Aday, aşağıdaki teknik gereksinimleri karşılayarak çözümünü geliştirmelidir:

- **Dil ve Çatılar:** C#, .NET Core Web API

- **Veritabanı ve Veri Erişimi:** Her mikroservisin kendine ait bir veritabanı veya veri deposu olmalıdır (örneğin her servis için ayrı bir **SQL** veritabanı şeması veya ayrı bir **NoSQL** koleksiyonu). Servisler doğrudan diğerinin veritabanına erişmemeli, veri paylaşımı API'ler veya mesajlar aracılığıyla olmalıdır. Veri erişim katmanı için ORM araçları (Entity Framework) veya uygun query mekanizmaları kullanılabilir. İlişkisel veritabanı (PostgreSQL, MySQL vb.) veya ihtiyaca göre NoSQL (MongoDB) tercih edilebilir. Aday, veritabanı bağlantılarını güvenli şekilde yönetmeli ve gerekli migrasyon/script dosyalarını sunmalıdır.

- **API İletişimi:** Mikroservisler arası iletişim öncelikli olarak **RESTful API** çağrıları ile gerçekleşmelidir. Tüm servisler, iyi tanımlanmış JSON tabanlı REST API'ler sunacak, HTTP status kodlarını doğru şekilde kullanacaktır. Gerekli durumlarda (özellikle uzun süren işlemler veya düşük sıkılıkta entegrasyonlar için) **asenkron iletişim** yöntemleri düşünülebilir. Bu kapsamda bir mesaj kuyruğu veya event bus (örneğin **RabbitMQ**, **Apache Kafka**) kullanılabilir. Asenkron iletişim, servislerin gevşek bağlanmasını sağlar ve sistemin daha ölçeklenebilir olmasına katkı sunar. Adaydan, kritik iletişimlerde uygun yöntemi (senkron vs. asenkron) seçmesi beklenir.

- **Güvenlik:** Tüm API çağrılarında uygun kimlik doğrulama ve yetkilendirme mekanizmaları uygulanmalıdır. Önerilen yöntem, merkezi **Auth** servisi veya API Gateway üzerinden **JWT (JSON Web Token)** tabanlı kimlik doğrulama yapısının kurulmasıdır. Kullanıcı girişinde üretilen bir JWT token, diğer servis çağrılarında doğrulanarak güvenli erişim sağlanmalıdır. Ayrıca servisler arası iletişimde gizli anahtarlar veya token'lar kullanılarak sadece yetkili servislerin haberleşmesi temin edilmelidir. Rollere göre yetki kontrolü (RBAC) uygulanmalı ve hassas işlemlerde gerekli kontroller yapılmalıdır. Güvenlik açısından şifrelerin/hash'lerin doğru yönetimi, girdi validasyonu (SQL injection, XSS gibi saldırılara karşı) ve gerekiyorsa rate limiting konuları da ele alınmalıdır.

- **Dokümantasyon ve API Sözleşmeleri:** Her bir mikroservis için API dokümantasyonu sağlanmalıdır. Tercihen **OpenAPI (Swagger)** spesifikasyonu kullanılarak her servisin uç noktaları, istek/cevap formatları belgelenmelidir. Böylece servislerin arayüzleri net bir şekilde tanımlanacak ve hem geliştirme sırasında hem de sonrasında diğer geliştiriciler tarafından anlaşılması kolay olacaktır.

- **Konfigürasyon ve Ortamlar:** Uygulama konfigürasyonları (veritabanı bağlantı adresleri, portlar, kimlik doğrulama anahtarları vb.), kod içerisine gömülmek yerine harici konfigürasyon dosyalarından veya ortam değişkenlerinden okunmalıdır. Geliştirme, test ve üretim ortamları için ayrı ayarlar öngörülmeli (**12-Factor App** prensiplerine uygunluk). Aday, projesini farklı ortamlarda çalıştırmaya uygun şekilde yapılandırmalıdır. Örneğin, Docker ortam değişkenleriyle yapılandırma veya .env dosyaları kullanımı beklenir.

- **Hata Yönetimi ve Loglama:** Her servis, oluşabilecek hataları yönetmeli ve anlamlı hata mesajları dönmelidir. Kritik işlemlerde uygun **try-catch** blokları ya da global hata yakalayıcılar kullanılmalı; API cevapları standart bir hata yapısında olmalıdır (ör. hata kodu ve mesajı ile). Ayrıca, servislerin aktivitelerini izlemek için loglama entegre edilmelidir. Her mikroservis için konsol veya dosya bazlı log tutulmalı, tercihen yapılandırılmış log formatı (JSON log gibi) kullanılmalıdır. Eğer mümkünse, dağıtık bir izleme sistemi (ELK stack, Grafana, etc.) entegrasyonu ekstra değer katar.

- **Performans ve Ölçeklenebilirlik:** Mikroservislerin konteynerize edilerek yatayda ölçeklenebilir olması beklenir. Aday, servislerini **Docker** imajları olarak paketlemeli ve böylece gerekirse birden çok instance çalıştırılabilir yapıda sunmalıdır. Servislerin bellek ve CPU gibi kaynak kullanımları makul seviyede olmalı; gerekliyse basit önbellekleme stratejileri uygulanmalıdır. Ayrıca, birbirini çağıran servisler arasındaki gecikme (latency) ve zaman aşımları (timeout) ele alınmalı, uzun süren işlemler asenkronlaştırılarak sistemin tepki süresi iyileştirilmelidir.

Özetle, adayın teknik açıdan güncel ve endüstri standartlarına uygun bir mikroservis mimarisi kurması beklenmektedir. Seçilen teknolojiler ve uygulanış biçimi, gereksinimlerin tamamını karşılamalı ve gelecekte sistemin genişletilebilir olmasını sağlamalıdır.

4. Teslim Edilecekler

Aday, proje sonunda aşağıdaki çıktıları teslim edecektir. Bu teslimatlar, adayın çalışmasını değerlendirmek ve projenin eksiksiz olduğundan emin olmak için kullanılacaktır:

- **Kaynak Kod:** Tüm mikroservislerin kaynak kodları, tercihen bir **versiyon kontrol sistemi** (ör. Git) kullanılarak sunulmalıdır. Kod yapısı, her servis için ayrı modüller/projeler olacak şekilde düzenlenmiş olmalıdır. Repository içinde her mikroservis ayrı bir klasör veya repository olarak ayarlanabilir. Kod içerisinde gerekli açıklama satırları (comment) eklenmeli ve kod standartlara uygun biçimde formatlanmış olmalıdır.

- **Derleme ve Çalıştırma Talimatları (README):** Projeyi derlemek, ortamı kurmak ve çalıştırmak için gereken adımları detaylı anlatan bir **README** dokümanı sağlanmalıdır. Bu doküman, bağımlılıkların nasıl yükleneceği, veritabanının nasıl oluşturulacağı (örn. gerekli SQL scriptleri veya ORM migrasyon adımları), konfigürasyonların nasıl yapılacağı ve uygulamanın nasıl başlatılacağı konularını içermelidir. Ayrıca, eğer özel bir yapılandırma veya ön koşul varsa (örneğin bir API anahtarı, üçüncü parti servis erişimi gibi), bunlar da belirtilmelidir.

- **API Dokümantasyonu:** Her bir servis için API uç noktalarını ve kullanım örneklerini açıklayan bir dokümantasyon sunulmalıdır. Bu, Swagger arayüzü şeklinde uygulanmış olabilir (tercihen proje çalıştığında /swagger veya benzeri bir URL'den erişilebilen), veya ayrı bir doküman (örn. PDF veya Markdown dosyası) olarak verilebilir. Dokümantasyonda her uç nokta için isteğin URL'i, yöntemi (GET, POST vb.), beklenen/gönderilen veri formatı ve olası cevaplar (200, 400, 500 durumları vb.) belirtilmelidir.

- **Test Kodları ve Raporları:** Birim testler ve varsa entegrasyon testleri, proje ile birlikte teslim edilmelidir. Her mikroservisin kritik işlevleri için birim test yazılması beklenmektedir. Teslim sırasında testlerin nasıl çalıştırılacağı README'de açıklanmalı ve testlerin geçtiğini gösteren bir çıktı veya rapor sunulmalıdır. Mümkünse kod kapsamı (code coverage) ölçümü yapılarak önemli bölümlerin ne oranda test edildiği belirtilebilir.

- **Çalışan Uygulama (Opsiyonel – Docker):** Projenin daha kolay değerlendirilebilmesi için, aday isterse uygulamanın çalışan bir versiyonunu Docker imajları olarak hazırlayabilir. Bu durumda her servis için bir Docker imajı ve kolay kurulum için bir **Docker Compose** dosyası sağlanmalıdır (bkz. Ekstralar). Bu sayede değerlendiriciler tek komutla tüm sistemi ayağa kaldırabilir ve test edebilir.

- **Diyagramlar ve Ek Dokümanlar:** Sistem mimarisi diyagramları (ör. genel mimari şeması, servis entegrasyon diyagramı, veritabanı ER diyagramı gibi) ve gerekliyse zaman çizelgesi, görev dağılımı vb. ek materyaller teslimata dahil edilmelidir. Diyagramlar anlaşılır ve okunaklı olmalı, tercihen yaygın kullanılan bir modelleme diliyle (UML, C4 modeli vb.) hazırlanmış olmalıdır. Özellikle mimariyi anlatan bir konteks diyagramı ve servislerin nasıl etkileştiğini gösteren bir akış diyagramı sunulması beklenir.

Tüm teslim edileceklerin derli toplu bir şekilde sunulması, proje değerlendirmesini kolaylaştıracaktır. Adayın, teslim ettiği materyallerin eksiksiz ve tutarlı olmasına özen göstermesi gerekmektedir.

5. Değerlendirme Kriterleri

Adayın çalışması aşağıdaki kriterler temelinde değerlendirilecektir. Bu kriterler, projenin hem işlevsel gereksinimleri karşılama düzeyini hem de teknik kalitesini ölçmeye yöneliktir:

- **Fonksiyonel Uyum:** Geliştirilen mikroservislerin yukarıda listelenen tüm işlevsel gereksinimleri karşılaması beklenir. Her bir özellik doğru şekilde uygulanmış mı, tüm ana senaryolar (ve önemli hata durumları) ele alınmış mı? Mevcut monolitik sistemin işlevselliği mikroservis versiyonunda eksiksiz yer alıyor mu? Bu hususlar kontrol edilecektir.

- **Kod Kalitesi ve Temiz Kod İlkeleri:** Kodun okunabilir, anlaşılır ve bakımı kolay olması önemlidir. İyi isimlendirmeler, uygun kod yapısı (paketleme/modülerlik), tekrarın minimize edilmesi (DRY prensibi) gibi unsurlar değerlendirilecektir. Ayrıca SOLID prensiplerine uyum, katmanlı mimari kullanımı ve genel temiz kod pratiklerinin uygulanıp uygulanmadığı incelenecektir.

- **Mimari Tasarım ve Mikroservis Prensiplerine Uyum:** Sistemin doğru bir şekilde mikroservislere bölünmesi, servislerin gevşek bağlı ve yüksek bağımlılık içermeyen şekilde tasarlanmış olması kritiktir. Her servis kendi sorumluluk alanına odaklanıyor mu? Aralarındaki iletişim doğru yöntemlerle mi yapılmış? Tekrar eden işlevler ortak bir şekilde ele alınmış mı (örneğin her servise kopyalamak yerine gerekirse ortak bir kütüphane veya servis kullanımı)? API Gateway ve olası BFF uygulaması verimli mi? Bu gibi mimari kararlar değerlendirilecek.

- **Performans ve Ölçeklenebilirlik:** Uygulamanın tasarımı ve gerçekleştirilme biçimi, artan yük altında ölçeklenmeye uygun mu? Servisler gerekli gördüğünde yatayda ölçeklenebilir (state-less) şekilde mi tasarlandı? Cevap süreleri, veritabanı sorgu performansları makul düzeyde mi? Gerekli yerlerde optimize çözümler kullanıldı mı? Değerlendirmede bu konulara bakılacaktır.

- **Güvenlik Uygulamaları:** Kimlik doğrulama ve yetkilendirme düzgün şekilde entegre edilmiş mi? Hassas bilgiler güvende mi (örn. şifreler hash'lenmiş, config'de gizli anahtar tutulmamış vb.)? Servisler arası iletişimde güvenlik (sertifika, token) kullanılmış mı? Temel güvenlik açıklarına (SQL enjeksiyonu, XSS, CSRF vs.) karşı önlemler alınmış mı? Bu başlıkta adayın güvenlik konusundaki dikkati değerlendirilecektir.

- **Test Kapsamı ve Kalitesi:** Otomatik testlerin kapsamı ve etkinliği önemli bir kriterdir. Kritik işlevler için birim testler yazılmış olması, olası hata durumlarının testlerle yakalanması beklenir. Testler hem pozitif (beklenen davranış) hem negatif (hata durumları) senaryoları kapsıyor mu? Kod kapsamı yeterli düzeyde mi? Ayrıca, test kodunun anlaşılırlığı ve düzeni de göz önüne alınacaktır.

- **Dokümantasyon ve Anlaşılabilirlik:** Adayın sunduğu dokümantasyonun kalitesi değerlendirilecektir. README ve API dökümantasyonları ne kadar net ve takip edilebilir? Sistemin mimarisi ve servislerin sorumlulukları dokümanlarda anlaşılır şekilde açıklanmış mı? Diyagramlar

doğru ve güncel mi? İyi bir dokümantasyon, projenin anlaşılmasını ve bakımını kolaylaştıracağı için önemli bir değerlendirme kriteridir.

- **İnovasyon ve İyileştirmeler:** Aday, ister dokümanında belirtilmeyen fakat projeye değer katan ekstra iyileştirmeler yapmış olabilir. Örneğin, ekstra bir güvenlik önlemi, daha gelişmiş bir mimari desen kullanımı (CQRS, Event Sourcing vb.), veya genel kullanıcı deneyimini artıracak bir detay. Bu tür olumlu yöndeki inisiyatifler değerlendirme sırasında artı puan getirecektir.

Değerlendirme sürecinde, yukarıdaki kriterlerin yanı sıra projenin genel bütünlüğü ve adayın sergilediği problem çözme yaklaşımı da göz önünde bulundurulacaktır. Her bir kriter belirli bir puan ağırlığına sahip olabilir ve sonuçta adayın başarısı bu puanların toplamıyla belirlenecektir.

6. Ekstralar (Opsiyonel + Puan Getiren Unsurlar)

Aşağıdaki unsurlar **zorunlu olmamakla birlikte** aday tarafından gerçekleştirilirse projeye artı puan kazandıracaktır. Bu ekstralar, adayın konuyu derinlemesine ele aldığını ve endüstri standartlarını yakından takip ettiğini gösterir:

- **Docker ve Docker Compose Kullanımı:** Tüm mikroservislerin Docker imajları hazırlanmışsa ve proje bir Docker Compose dosyası ile kolayca ayağa kaldırılabiliyorsa, bu önemli bir artıdır. Docker Compose ile veritabanı, servisler, belki mesaj kuyruğu gibi bileşenlerin tek komutla çalıştırılması sağlanabilir. Bu, projenin taşınabilirliğini ve kurulumu kolaylaştırdığı için takdir edilecektir.

- **CI/CD Boru Hattı (Pipeline) Entegrasyonu:** Aday, projesine bir sürekli entegrasyon/sürekli dağıtım (CI/CD) işlemi ekleyebilirse bu ekstra puan kazandırır. Örneğin GitHub Actions, GitLab CI veya Jenkins gibi araçlarla kodun her push'unda testlerin çalıştırılması, imajların oluşturulması ve belki otomatik deploy senaryosu hazırlanması profesyonel bir yaklaşımdır. **Basit bir CI pipeline yaml dosyası bile, bu konuda adım atıldığını gösterir.**

- **Event-Driven Mimari ve Mesajlaşma:** Projede istek/cevap modelinin yanı sıra event-driven (olay güdümlü) yaklaşımlar kullanılması beklenenin ötesinde bir katkıdır. Örneğin, bir servis belirli bir olayı (ör. yeni kayıt oluşturuldu) bir mesaj kuyruğuna yayınlar, başka bir servis bu mesajı yakalayıp işlemler yapar (ör. bildirim gönderir). Böyle bir yapı, servislerin daha gevşek bağlı olmasını sağlar ve ölçeklenebilirliği artırır. Adayın RabbitMQ, Kafka gibi bir altyapı kurup entegre etmesi ve servisleri buna uygun tasarlaması opsiyonel ancak değerli bir beceri göstergesidir.

- **Yük Dengeli ve Sağlamlık (Resilience) Özellikleri:** Mikroservis ortamında **circuit breaker**, **retry mekanizmaları**, **health check** uç noktaları ve **yük dengeleme** kullanımı gibi konuların ele alınması ekstra puan getirecektir. Örneğin, bir servis çağrısında başarısızlık olursa otomatik tekrar denemesi veya servis ayakta değilse devre kesici ile hızlıca hata dönülmesi gibi Netflix OSS (Hystrix) benzeri yaklaşımlar uygulanmışsa bu ileri seviye bir çalışmaya işaret eder.

- **Servis Keşfi ve Konfigürasyon Yönetimi:** Çok sayıda mikroservisin olduğu ortamlarda **service discovery** (Eureka, Consul vb.) ve merkezi konfigürasyon yönetimi (Spring Cloud Config gibi) kullanımı faydalıdır. Aday bu seviyede bir çözüm entegre etmişse, yani servislerin dinamik olarak

birbirini bulması veya konfigürasyonlarını merkezi olarak alması gibi gelişmiş konuları adreslediyse bu da not edilir.

- **Özelleştirilmiş Logging/Monitoring:** Uygulamanın takibi için merkezi bir log yönetim sistemi (ELK, Grafana Loki vb.) veya dağıtık tracing (Jaeger, Zipkin) entegrasyonu yapılması, projenin prodüksiyon kalitesine yakın olduğunu gösterir ve pozitif etki yapar. Aynı şekilde, servis metriklerinin toplanması (Prometheus + Grafana gibi) ve basit de olsa dashboard oluşturulması da ekstra olarak değerlendirilebilir.

- **Ön Yüz (Basit GUI) veya Ek İstemci:** Her ne kadar proje bir backend dönüşüm projesi olsa da, aday mikroservisleri test etmek veya göstermek amacıyla basit bir arayüz veya istemci uygulama sunarsa bu da artı sayılabilir. Örneğin, servisleri çağıran küçük bir web arayüzü veya Postman koleksiyonu hazırlanması, sistemin kullanımını kolaylaştıran ekstralardır.

Bu ekstralar, adayın beklentilerin ötesine geçerek projeyi olgunlaştırdığını ve yetkinliklerini sergilediğini gösterir. Aday, zaman ve imkân bulabildiği ölçüde bu opsiyonel alanlara da değinebilir. Her yapılan ekstra için değerlendirmede ek puan veya olumlu yorumlar alması muhtemeldir.

7. Sunum İçeriği

Aday, proje tamamlandıktan sonra bir teknik sunum gerçekleştirecektir. Bu sunumda aşağıdaki konuların ve içeriklerin yer alması beklenmektedir:

- **Mimari Genel Bakış:** Aday, öncelikle sistemin genel mimari tasarımını sunmalıdır. Hangi mikroservislerin geliştirildiği, bunların birbirleriyle nasıl etkileştiği, varsa API Gateway veya mesajlaşma altyapısının nasıl konumlandığı gibi konular net bir diyagram eşliğinde anlatılmalıdır. Bu kısımda, adayın çizmiş olduğu mimari diyagram(lar) üzerinden sistemin kuşbakışı görünümü açıklanacaktır.

- **Mimari Kararlar ve Gerekçeleri:** Sunumda, adayın önemli tasarım kararlarını ve bunların ardındaki nedenleri açıklaması beklenir. Örneğin, servis sınırları neden bu şekilde belirlendi, neden belirli bir teknoloji veya pattern (ör. event-driven, ya da belirli bir veri tabanı) seçildi, API Gateway kullanmanın getirdiği faydalar neler, gibi soruların cevapları sunulmalıdır. Her kararın artıları, eksileri değerlendirilmiş mi, alternatifler düşünülmüş mü? Bu noktalar, adayın düşünce sürecini ortaya koyacaktır.

- **Detaylı Bileşen İncelemesi:** Her bir mikroservis kısaca tanıtılmalı, ne iş yaptığı ve iç yapısı (katmanları, önemli sınıfları veya modülleri) yüksek seviyede açıklanmalıdır. Gerekirse bir sınıf diyagramı veya akış diyagramı ile servis içindeki işlemlerin sırası gösterilebilir. Örneğin, bir istek geldiğinde servis içinde nasıl işlendiği (controller -> service -> repository akışı gibi) anlatılabilir. Ayrıca servislerin dış dünyaya sunduğu ana API uç noktaları ve bu uç noktaların ne yaptığı belirtilmelidir.

- **Entegrasyon ve İletişim:** Mikroservisler arası iletişimin nasıl gerçekleştirildiği sunumda açık bir şekilde ele alınmalıdır. Senkron iletişimde hangi servis kiminle konuşuyor, asenkron iletişim varsa

hangi olaylar yayınlanıyor/ dinleniyor bunlar örneklerle açıklanmalı. Gerekirse bir **sequence diagram** (zaman dizini diyagramı) kullanılarak örnek bir iş akışında (use-case) servislerin birbirleriyle haberleşme sırası gösterilmelidir. Bu, değerlendiricilere sistemin dinamiğini anlama konusunda yardımcı olacaktır (örn. "Yeni bir müşteri eklendiğinde, Müşteri Servisi veriyi kaydeder ve bir 'CustomerCreated' olayı yayınlar, Bildirim Servisi bu olayı dinleyip e-posta gönderir" gibi bir akış).

- **Teknoloji Stack ve Araçlar:** Aday, kullandığı teknolojileri ve araçları sunumda belirtmeli ve kısaca neden bu araçları seçtiğini açıklamalıdır. Örneğin, "Backend için .NET Core seçildi çünkü ekip tecrübesi buna uygundu ve cross-platform mikroservis geliştirme imkanı sağladı; veritabanı olarak PostgreSQL kullanıldı çünkü ilişkisel bir yapı gerekiyordu; mesajlaşma için RabbitMQ tercih edildi, vb." şeklinde. Ayrıca Docker, Kubernetes, CI/CD gibi devops araçları kullanıldıysa bunların entegrasyonu ve faydaları anlatılmalıdır.

- **Karşılaşılan Zorluklar ve Çözümler:** Proje sırasında adayın karşılaştığı önemli teknik sorunlar veya mimari zorluklar varsa, bunlar ve nasıl aşıldıkları sunumda yer almalıdır. Örneğin, "X servisinde döngüsel bağımlılık sorununu çözmek için Y deseni uygulandı" veya "Veri tutarlılığı için iki-fazlı onay mekanizması denendi" gibi tecrübeler paylaşılabilir. Bu kısım, adayın problem çözme becerisini ve proje sırasında öğrendiklerini aktarması için bir fırsattır.

- **Canlı Demo (Opsiyonel):** Sunum esnasında, eğer mümkünse, aday kısa bir canlı demo ile sistemin çalıştığını gösterebilir. Örneğin Postman ile birkaç API çağrısı yaparak servislerin cevap verdiğini, veya Docker Compose ile tüm servislerin ayağa kalktığını hızlıca göstermek etkili olabilir. Canlı demo opsiyoneldir ancak yapıldığı takdirde projenin gerçekten çalışır ve bütünleşik olduğunu kanıtlar.

- **Sonuç ve İleri Adımlar:** Sunumun sonunda, aday projenin genel bir özetini yapmalı ve ileriye dönük neler yapılabileceğine değinmelidir. Mikroservis dönüşümünün getirdiği kazanımlar özetlenebilir (ölçeklenebilirlik, esneklik vb.). Ayrıca zaman kısıtından dolayı yapılamayan ama yapılabilecek geliştirmeler varsa (ör. daha fazla servis ekleme, gelişmiş monitoring, auto-scaling senaryoları) bunlardan bahsedilebilir. Bu, adayın vizyonunu ve sistemi daha da geliştirebilme fikirlerini ortaya koyacaktır.

8. Tamamlama Süresi Hakkında Bilgilendirme

Proje içindeki gereksinimlerin temel bir **mikroservis prototip** düzeyinde tamamlanması ve kısa bir **teknik sunum** hazırlanması için ortalama **3 ila 5 günlük** bir süre öngörülmektedir.

Not:

- Bu süre zarfı, projenin temel fonksiyonlarını (mikroservislerin ayrılması, basit veritabanı erişimi, kimlik doğrulama, temel testler ve dokümantasyon) içeren bir prototip seviyesini hedeflemektedir.
- Projeye eklenecek mesajlaşma altyapısı, Docker Compose entegrasyonu, kapsamlı testler ve daha gelişmiş güvenlik özellikleri gibi ek işlevler olması durumunda, teslim süresi adayın takdirine göre biraz daha uzayabilir.

Aday, belirtilen süre içinde tamamlayamadığı veya ek geliştirme ihtiyacı duyduğu noktaları kendi değerlendirmesi doğrultusunda açıkça belirtmelidir. Önemli olan, **fonksiyonel ve anlaşılır bir prototip** sunarak proje yaklaşımı ve teknik becerilerini gösterebilmektir.

Başarılar…