|  |  |
| --- | --- |
| **Doküman Adı** | PetVerse Mimari Tasarım Dokümanı |
| **Doküman Kodu** | YGD- PetVerse -MT-Mimari Tasarım |
| **Süreç Grubu** | Tasarım Kodlama Ürün Entegrasyon Süreci |
| **Süreç** | Tasarım Kodlama Ürün Entegrasyon Süreci |
| **Revizyon** | V1.0 |
| **Tarih** | 3 Temmuz 2025 |
| **Süreç Sahibi** | Yağmur BAŞOĞLU , Sedanur AYHAN ,Tunahan ALTINTOP , Fatih BOZKOYUN |

|  |  |  |
| --- | --- | --- |
| **ÜNVAN** | **TARİH** | **İMZA** |
| **HAZIRLAYAN** | 3 Temmuz 2025 | Yağmur BAŞOĞLU , Sedanur AYHAN |
| **GÖZDEN GEÇİREN** | 3 Temmuz 2025 | Yağmur BAŞOĞLU , Sedanur AYHAN  Tunahan ALTINTOP  Fatih BOZKOYUN |
| **ONAYLAYAN** | 3 Temmuz 2025 | Tunahan ALTINTOP |

# Değişiklik Tarihçesi

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Yayım No** | **Tarih** | **Değişen Sayfa/Tablo/Şekil** | **\*E,**  **S, D** | **Değişikliğin Sahibi** | **Değişiklik Kaynağı** |
| **V1.0** | 3 Temmuz 2025 | Döküman hazırlandı | E | Yağmur BAŞOĞLU , Sedanur AYHAN |  |
| **V2.0** | 25 Temmuz 2025 | Güncelleme yapıldı | D | Yağmur BAŞOĞLU |  |

\*E: Ekleme S: Silme D: Değiştirme

# Amaç ve Kapsam

## Amaç

PetVerse, kullanıcıların sanal ortamda evcil hayvanlara ait bilgileri yönetmesini, kullanıcı kayıt işlemlerini gerçekleştirmesini, aktiviteleri takip etmesini ve bildirimleri almasını sağlayan, mikroservis mimarisiyle geliştirilmiş bir yazılım ürünüdür.Bu sistem, kullanıcıların hayvan bilgilerini kaydetmesini, aktiviteleri takip etmesini ve gerektiğinde hatırlatma almasını sağlar. Projede Java 21'in modern yazılım geliştirme olanakları (Record, Virtual Thread, Pattern Matching vb.) aktif olarak kullanılmaktadır. Sistem, temel olarak Java ile geliştirilen servisler ve React tabanlı bir kullanıcı arayüzünden oluşmaktadır. Servis keşfi için Eureka, yönlendirme için APIGateway, mesajlaşma için RabbitMQ ve veri yönetimi için PostgreSQL tercih edilmiştir.

## Kapsam

Bu doküman, PetVerse sisteminin mimari tasarımına ilişkin donanım ve yazılım bileşenlerinin genel görünümünü, servisler arası iletişim yapısını ve sistemin iç ve dış arayüzlerini kapsamaktadır. Mimari tasarım, kullanıcı ve sistem gereksinimlerine dayalı olarak mikroservis mimarisi çerçevesinde yapılandırılmıştır ve yazılım gereksinimlerine tamamlayıcı niteliktedir. PetVerse’in kapsamı; kullanıcı yönetimi, hayvan kayıtları, aktivite takibi ve bildirim hizmetlerinin her biri için bağımsız ama koordineli çalışan mikroservislerin tanımlanmasını ve bu servislerin servis keşfi, yönlendirme, mesajlaşma ve kalıcı veri yönetimi sağlayacak şekilde entegrasyonunu içermektedir.

* 1. Mimari Paydaşları

|  |  |  |
| --- | --- | --- |
| **Paydaş(Stakeholder) Rol ve İlgili olduğu mimari görünümler**  **açıklamalar** | | |
| **Tunahan Altıntop** | Türksat Yazılım Mimari Müdürlüğü | Genel Sistem Yapısı, Yazılım Ürünleri Görünümü, Modül Yapısal  Görünümü, Kurulum Görünümü, Yazılım Geliştirme Görünümü |
| **Fatih Bozkoyun** | Türksat Yazılım Mimari Müdürlüğü | Genel Sistem Yapısı, Yazılım Ürünleri Görünümü, Modül Yapısal  Görünümü, Kurulum Görünümü, Yazılım Geliştirme Görünümü |
| **Yağmur Başoğlu** | Türksat Yazılım Mimari Müdürlüğü | Genel Sistem Yapısı, Yazılım Ürünleri Görünümü, Modül Yapısal  Görünümü, Kurulum Görünümü, Yazılım Geliştirme Görünümü |
| **Sedanur Ayhan** | Türksat Yazılım Mimari Müdürlüğü | Genel Sistem Yapısı, Yazılım Ürünleri Görünümü, Modül Yapısal  Görünümü, Kurulum Görünümü, Yazılım Geliştirme Görünümü |

* 1. Referans Dokümanlar

|  |  |  |  |
| --- | --- | --- | --- |
| **Referans Numarası** | **Revizyon Numarası** | **Revizyon Tarihi** | **Referans Adı** |
| **YM-Karar Analiz-2025** | v1.0 | 16 Temmuz 2025 | |  | | --- | |  |  |  | | --- | | [Karar Analizi Dökümanı](https://d.docs.live.net/1ad0e738fe361cd4/Masaüstü/STAJ/TURKSAT/Proje/Karar%20Analiz%20Dökümanları/KararAnalizi%20Geliştirme%20Ortamı.docx) | |
| **YM-Gözlemlenebilirlik** | v1.0 | 22 Temmuz 2025 | [Gözlemlenebilirlik Kullanım Kılavuzu](https://d.docs.live.net/1ad0e738fe361cd4/Masaüstü/STAJ/TURKSAT/Proje/Gözlemlenebilirlik%20Kurulum%20Kılavuzu/Gözlemlenebilirlik%20Kullanım%20Kılavuzu.docx) |

# Tanımlar ve Kısaltmalar

* 1. Tanımlar

|  |  |
| --- | --- |
| **İfade Tanım** | |
| **Mikroservis** | Her biri bağımsız olarak dağıtılabilen, küçük ve tek bir işlevi yerine getiren servislerden oluşan mimari yaklaşımdır. |
| **Eureka** | Servis keşfi sağlayan bir Netflix Spring Cloud bileşenidir. Mikroservislerin birbirini bulmasını sağlar. |
| **API Gateway** | Mikroservislere tek giriş noktası sağlayan yapı. Kimlik doğrulama, rate limiting, yönlendirme gibi işlemleri üstlenir. |
| **JWT** | JSON Web Token, kullanıcı kimlik doğrulaması ve yetkilendirmesi için kullanılan bir açık standarttır. |
| **RabbitMQ** | Mikroservisler arası mesajlaşmayı sağlayan açık kaynaklı mesaj kuyruğu sistemidir. |
| **Config Server** | Merkezi konfigürasyon yönetimi sağlayan Spring Cloud bileşenidir. Tüm mikroservislerin yapılandırma dosyalarını merkezi bir yerden okumasını sağlar. |
| **Prometheus** | Mikroservislerden metrik toplayan ve zaman serisi verisi olarak saklayan bir gözlemlenebilirlik aracıdır. |
| **Grafana** | Prometheus gibi kaynaklardan veri çekerek görselleştirme sağlayan dashboard aracıdır. |
| **Circuit Breaker** | Servisler arası hatalı istekleri yönetmek için kullanılan dayanıklılık (resilience) desenidir. Resilience4j ile uygulanabilir. |
| **Swagger / OpenAPI** | REST API'leri belgeleyen, test edilebilir arayüz sunan araçlardır. Geliştiricilerin API’leri kolayca anlamasını sağlar. |
| **Docker / Docker Compose** | Uygulamaların bağımlılıklarıyla birlikte konteynerize edilmesini ve servislerin birlikte çalıştırılmasını sağlar. |
| **Spring Boot** | Spring tabanlı uygulamaların hızlı geliştirilmesini sağlayan Java framework'üdür. |
| **Spring**  **Security** | Kimlik doğrulama ve yetkilendirme işlemleri için kullanılan güvenlik framework'üdür. |
| **PostgreSQL** | Açık kaynaklı, güçlü bir ilişkisel veritabanı sistemidir. Mikroservislerde veri yönetimi için kullanılır. |

* 1. Kısaltmalar

|  |  |
| --- | --- |
| **Kısaltma Açıklama** | |
| **API** | Application Programming Interface |
| **JWT** | JSON Web Token |
| **JPA** | Java Persistence API |
| **REST** | Representational State Transfer |
| **ORM** | Object Relational Mapping |
| **HTTP** | Hypertext Transfer Protocol |
| **DBMS** | Database Management System |
| **CRUD** | Create, Read, Update, Delete |
| **DTO** | Data Transfer Object |
| **CI/CD** | Continuous Integration / Continuous Deployment |
| **MQ** | Message Queue |
| **POM** | Project Object Model (Maven yapılandırma dosyası) |

# Mimari Tasarım Kriterleri ve Kararları

PetVerse projesi, mikroservis mimarisi yaklaşımı ile geliştirilmektedir. Mimarinin temel tasarım kriterleri, projenin gereksinimlerine ve Yazılım Mimari Müdürlüğü tarafından belirlenen hedeflere göre aşağıdaki şekilde şekillendirilmiştir:

**Katmanlı Mimari:** Uygulama üç katmanlı mimari (controller–service–repository) yapısında geliştirilmekte olup, iş mantığı ve veri erişimi katmanları ayrıştırılarak sürdürülebilirlik ve test edilebilirlik sağlanmaktadır.

**Teknoloji Tercihleri:**

* Backend tarafında Java 21 ve Spring Boot 3.x kullanılmaktadır. Modern Java özellikleri (record, sealed class, virtual threads) mimari içinde uygun yerlerde tercih edilmiştir.
* Veritabanı olarak PostgreSQL seçilmiş, her mikroservis için ayrı veritabanı prensibi benimsenmiştir.
* Mesajlaşma için RabbitMQ, servisler arası asenkron iletişim sağlamak üzere kullanılmaktadır.
* Servis keşfi ve yönlendirme için sırasıyla Eureka Server ve Spring Cloud Gateway yapılandırılmıştır.
* Güvenlik için Spring Security ve JWT (JSON Web Token) altyapısı kullanılmaktadır.

**Konfigürasyon:** Spring Cloud Config Server kullanılarak, servis yapılandırmaları merkezi bir Git repo üzerinden yönetilmektedir.

**Dağıtım:** Docker ile container yapısı kurulmuş, Docker Compose ile servis orkestrasyonu sağlanmıştır. CI/CD için GitHub Actions kullanılmaktadır.

**Servis İletişimi:** Tüm mikroservisler, dış sistemlerle ve frontend arayüzü ile haberleşebilmek adına **REST API** yapısında geliştirilmiştir. JSON veri formatı standart olarak benimsenmiştir.

Bu mimari yapı, dağıtık sistemlerde veri tutarlılığı, hata toleransı ve sistem izlenebilirliği gibi konular göz önünde bulundurularak tasarlanmıştır.

**Loglama:** Logback üzerinden özelleştirilmiş loglama yapısı kurulmuştur. Test ortamı ve normal ortam için ayrı log profilleri tanımlanmıştır (logback-spring.xml).

Her servisin logları ayrı dosyalarda tutulur, üretim ortamında /app/logs, testte ./logs/test dizinleri kullanılır.

**Gözlemlenebilirlik:** Prometheus ve Grafana ile metrik toplama ve görselleştirme yapılandırılmıştır.

Her servise Prometheus endpoint’i (/actuator/prometheus) eklenmiştir.

Grafana panelleri ile sistem performansı ve sağlık durumu izlenmektedir.

**Circuit Breaker / Fault Tolerance** : Servisler arası iletişimde hata toleransı için Resilience4j Circuit Breaker kullanılmaktadır.

ActivityService, PetService ile olan iletişiminde devre kesici (circuit breaker) mekanizması kullanır.

**Test ve Kalite Araçları**

JUnit 5 ve Testcontainers kullanılarak birim ve entegrasyon testleri yazılmıştır.

Kod kalitesi takibi için SonarQube entegre edilmiştir, GitHub Actions üzerinden analiz edilir.

# Genel Sistem Yapısı

PetVerse, kullanıcıların sanal ortamda evcil hayvan besleyip bakımını yapabileceği bir yazılım ürünüdür. Uygulama, hayvanların beslenme, oyun, aktivite ve bildirim süreçlerini yönetmeye odaklanmıştır. Sistem, kullanıcıların kendi sanal hayvanlarını ekleyip takip edebileceği, aktiviteler atayabileceği ve bu aktivitelerle ilgili hatırlatmalar alabileceği bir ortam sağlar.

PetVerse, mikroservis mimarisiyle geliştirilmiş olup, her bir işlevsel modül bağımsız servisler olarak yapılandırılmıştır. Servisler Java 21 ile geliştirilmiş olup, modern dil özelliklerinden (Record, Pattern Matching, Virtual Thread vb.) yararlanılmaktadır.

React tabanlı kullanıcı arayüzü, API Gateway üzerinden servislerle haberleşir. Servis keşfi için Eureka, mesajlaşma altyapısı için RabbitMQ, merkezi yapılandırma için Config Server, güvenlik için JWT ve Spring Security, veri saklama için PostgreSQL kullanılmıştır. Servislerin davranışlarını izleyebilmek ve sistem sağlığını takip edebilmek amacıyla Prometheus ile metrikler toplanmakta, Grafana panoları ile görselleştirilmektedir. Log yönetimi için dosya tabanlı loglama sistemi uygulanmaktadır. Tüm servisler GitHub Actions kullanılarak CI/CD süreciyle otomatik olarak derlenip test edilmekte ve Docker imajları üretilebilmektedir. Her mikroservis için birim testler (JUnit) ve entegrasyon testleri (Testcontainers) ile güvenilirlik artırılmıştır.

Sistem, kullanıcı ihtiyaçlarına göre genişletilebilir, modüler ve ölçeklenebilir bir yapıdadır. Güncel ihtiyaçlara uyum sağlayabilmek adına, geliştirme sürecinde çevik (agile) metodoloji benimsenmiştir.

## Sistem İşleyiş Senaryoları

Bu bölümde PetVerse sisteminin ana işleyiş senaryoları sunulmuştur. Sistemin temel hedefi, kullanıcıların evcil hayvan bilgilerini, aktivitelerini ve bildirimlerini mikroservis mimarisi üzerinden yönetmesini sağlamak; kullanıcıların sisteme kaydolması, oturum açması, evcil hayvan eklemesi ve sistemden bildirim alması gibi akışların gerçekleştirilmesidir.

### PetVerse Talep Oluşturma Senaryosu

Aşağıdaki UML Sırası çizimi (sequence diagram), PetVerse projemizin talep oluşturması ile ilgili ana akışlarını göstermektedir. A diagram of a project

AI-generated content may be incorrect.

# Mimari Görünümler

## Yazılım Ürünleri Görünümü

PetVerse yazılımını oluşturan ürünler (artifact), mikroservis mimarisi esas alınarak geliştirilmiş olup üç gruba ayrılmıştır:

* Proje kapsamında geliştirilen bileşenler: Şemada beyaz renkle gösterilen servislerdir. Bunlar UserService, PetService, ActivityService, NotificationService, WeatherService, API Gateway, Eureka Server ve Config Server gibi projeye özgü mikroservis bileşenleridir.
* Hazır ürün olarak temin edilen parçalar: Örneğin PostgreSQL, RabbitMQ gibi bileşenler projede harici olarak kullanılmıştır. Bu bileşenler diyagramda gösterilmemiş olsa da proje içinde hizmet vermektedir.
* A screenshot of a computer

  AI-generated content may be incorrect.Entegrasyon ürünleri: Spring Boot, Spring Security, Eureka, JWT, gibi uygulama içi entegrasyonu sağlayan teknolojilerdir. Bu teknolojiler mikroservislerin içerisinde bağımlılık olarak kullanılmıştır.

Şekilde her bir artifact, Java platformunda .jar tipinde paketlenmiş yazılım bileşenine denk gelmektedir. Görünüm dili olarak UML 2.5 tercih edilmiştir.

**Not:** Tüm mikroservisler Spring Boot tabanlı olup bağımsız olarak CI/CD süreciyle derlenir ve Docker konteyneri olarak deploy edilir.

### Geliştirilen Ürünler

PetVerse projesi kapsamında aşağıdaki yazılım bileşenleri geliştirilmiştir. Her biri mikroservis mimarisinin bir parçası olarak bağımsız çalışmaktadır. Bileşenlerin açıklamaları, dizin yapıları ve teknik bilgileri aşağıda verilmiştir:.

* + - 1. *userservice*
         * **Modül yapısı**: jar (Java)
         * **Dizin/Dosya Adı**: userservice
         * **Kurulum**: Docker container olarak yapılandırılmıştır.
         * **Erişim**: http://localhost:8082
         * **Açıklamalar**: Kullanıcı kayıt, giriş ve kimlik doğrulama işlemlerini yürütür. JWT tabanlı Authentication ve Authorization işlemleri içerir.
      2. *apigateway*
         * **Modül yapısı:** jar (Java)
         * **Dizin/Dosya Adı:** apigateway
         * **Kurulum**: Docker container olarak yapılandırılmıştır.
         * **Erişim:** http://localhost:8080
         * **Açıklamalar:** Tüm istemci isteklerini karşılayan merkezi ağ geçididir. Servislere yönlendirme (routing), JWT kontrolü ve rate limiting işlemlerini uygular.
      3. *configserver*
         * **Modül yapısı:** jar (Java)
         * **Dizin/Dosya Adı:** configserver
         * **Kurulum**: Docker üzerinden çalıştırılır.
         * **Erişim:** http://localhost:8888
         * **Açıklamalar:** Tüm mikroservislerin yapılandırma dosyalarını merkezi olarak yönetir. Spring Cloud Config altyapısı kullanılmıştır.
      4. *eurekaserver*
         * **Modül yapısı:** jar (Java)
         * **Dizin/Dosya Adı:** eurekaserver
         * **Kurulum**: Docker üzerinden çalıştırılır.
         * **Erişim:** http://localhost:8761
         * **Açıklamalar:** Servis keşfi sağlar. Mikroservislerin birbirini bulmasını ve haberleşmesini mümkün kılar.
      5. *petservice*
         * **Modül yapısı**: jar (Java)
         * **Dizin/Dosya Adı**: petservice
         * **Kurulum**: Docker container olarak yapılandırılmıştır.
         * **Erişim**: http://localhost:8083
         * **Açıklamalar**: Kullanıcıların evcil hayvan bilgilerini yönetmesini sağlar. REST API üzerinden CRUD işlemleri sunar. JPA ve Java record yapısı kullanılmıştır.
      6. *activityservice*
         * **Modül yapısı**: jar (Java)
         * **Dizin/Dosya Adı**: activityservice
         * **Kurulum**: Docker container olarak yapılandırılmıştır.
         * **Erişim**: http://localhost:8084
         * **Açıklamalar**: Evcil hayvanlara ait günlük aktivitelerin (beslenme, oyun, yürüyüş vb.) kaydını tutar. Event-driven mimari kullanılarak RabbitMQ üzerinden NotificationService ile haberleşir. Yapay zeka desteğiyle evcil hayvanın ruh hali (PetMood) tahmini yapılır ve mood verisi aktiviteye eklenir. Aktivite verileri doğrulama kurallarına göre kontrol edilir.
      7. *notificationservice*
         * **Modül yapısı**: jar (Java)
         * **Dizin/Dosya Adı**: notificationservice
         * **Kurulum**: Docker container olarak yapılandırılmıştır.
         * **Erişim**: http://localhost:8085
         * **Açıklamalar**: Bu servis, sistemde gerçekleşen olaylara (örneğin bir aktivite oluşturulması) karşılık olarak kullanıcılara bildirim göndermek amacıyla çalışır. RabbitMQ mesaj kuyruğu kullanılarak asenkron şekilde çalışır. ActivityService tarafından yayınlanan mesajları dinler ve ilgili kullanıcıya bildirim gönderilmesi işlemini üstlenir. Hatırlatma bildirimleri RabbitMQ ile basit mesaj gönderimi.
      8. *weatherservice*
* **Modül yapısı:** jar (Java)
* **Dizin/Dosya Adı:** weatherservice
* **Kurulum:** Docker container olarak yapılandırılmıştır.
* **Erişim:** http://localhost:8086
* **Açıklamalar:** Gerçek zamanlı hava durumu verilerini dış API’ler üzerinden alır ve sistemdeki aktiviteleri hava koşullarına göre yönlendirir. Özellikle yürüyüş gibi dış ortam aktivitelerinde hava durumuna dayalı karar verme mekanizması sağlar.

### Hazır Ürünler

Aşağıdaki yazılım ürünleri, PetVerse sistemi tarafından hazır ürün olarak kullanılmaktadır. Bu ürünler açık kaynaklı kütüphaneler olup, sistemde çeşitli mikroservislerin geliştirilmesinde ve entegrasyonunda kullanılmaktadır.

* **Spring Boot 3.2.x**: REST servisleri geliştirmek için temel yapı sunar.  
  *Lisans:* Açık kaynak
* **Spring Security 6.x**: Kullanıcı doğrulama ve yetkilendirme (authentication & authorization) işlemleri için kullanılır.  
  *Lisans:* Açık kaynak
* **Lombok 1.18.x**: Java sınıflarındaki getter, setter, constructor gibi metotların otomatik üretilmesini sağlar.  
  *Lisans:* Açık kaynak
* **Spring Boot Mail**: E-posta göndermek isteyen servislerde (örneğin NotificationService) kullanılabilir.  
  *Lisans:* Açık kaynak
* **Spring Cloud Gateway:** API Gateway işlevini üstlenir, dinamik yönlendirme ve filtreleme sağlar. Lisans: Açık kaynak
* **Spring Cloud Netflix Eureka:** Servis keşfi (service discovery) için kullanılır. Lisans: Açık kaynak
* **Spring Cloud Config:** Mikroservislerin merkezi konfigürasyon yönetimini sağlar.Lisans: Açık kaynak
* **JSON Web Token (JWT)**: Kullanıcı oturumu ve güvenli API erişimi için token üretimi ve doğrulaması yapılır.  
  *Lisans:* Açık kaynak
* **Spring Boot Data JPA 3.x**: Veritabanı işlemleri için ORM sağlar.  
  *Lisans:* Açık kaynak
* **PostgreSQL 15.x**: Mikroservislerin veri saklama işlemlerinde kullanılan ilişkisel veritabanıdır.  
  *Lisans:* Açık kaynak
* **React 18.x**: Web arayüzünün geliştirilmesinde kullanılan modern JavaScript kütüphanesidir.  
  *Lisans:* Açık kaynak
* **RabbitMQ**: Mikroservisler arası mesajlaşmayı sağlar. Özellikle NotificationService’in mesaj alması için kullanılır.  
  *Lisans:* Açık kaynak
* **Docker**: Uygulama bileşenlerinin konteynerler halinde dağıtılmasını sağlar.  
  *Lisans:* Açık kaynak
* **Prometheus**: Mikroservislerin metrik verilerini toplayan gözlemlenebilirlik aracıdır. *Lisans*: Açık kaynak
* **Grafana**: Prometheus metriklerini görselleştiren dashboard aracı. *Lisans*: Açık kaynak
* **TestContainers**: Entegrasyon testlerinde bağımlı servislerin konteyner ortamında çalıştırılmasını sağlar. *Lisans*: Açık kaynak
* **SonarQube**: Kod kalitesi analizi ve teknik borç takibi için kullanılır.*Lisans*: Açık kaynak
* **Open-Meteo API:** Hava durumu verisi sağlamak için kullanılan harici servis. Lisans: Açık veri / ücretsiz API
* **HuggingFace Inference API:** Yapay zeka tabanlı duygu analizi modeli (j-hartmann/emotion-english-distilroberta-base) kullanılarak evcil hayvan aktivitelerinden duygu tahmini yapılır. Lisans: Ücretsiz kullanım sınırlarıyla açık API (HuggingFace lisansı geçerlidir)
* **WebClient (Spring WebFlux):** REST API çağrıları yapmak için reaktif istemci. PetMoodService HuggingFace API’yi çağırmak için kullanır. Lisans: Açık kaynak

### Modül Bağımlılıkları

PetVerse mikroservis mimarisine sahip bir sistemdir ve her bir servis kendi bağımsız çalışma ortamına sahiptir. Bu sebeple servisler arası sıkı bir bağ bulunmamakla birlikte, Config Server, Eureka Server ve RabbitMQ gibi servisler belirli mikroservislerin çalışması için bağımlılık oluşturmaktadır. Örnek olarak UserService ve PetService → Eureka & Config Server'a bağımlı, NotificationService → RabbitMQ’ya bağımlı

## Modül Yapısal Görünümü

**PetVerse – Sanal Evcil Hayvan Yönetim Sistemi**, mikroservis mimarisi prensipleri doğrultusunda geliştirilmiş, oyunlaştırılmış sanal evcil hayvan bakım uygulamasıdır. Bu kapsamda geliştirilen yazılım bileşenleri, kullanıcıların evcil hayvan oluşturma, bakım yapma, aktivite gerçekleştirme, gelişim sürecini izleme gibi etkileşimleri sağlayacak şekilde modüler yapıda tasarlanmıştır.

Bu bölümde, PetVerse yazılım bileşenlerinin geliştirilen ürün içerisindeki **yapısal görünüm diyagramı** ve bu bileşenlere ait kısa tanıtımlar yer almaktadır. Amaç, her bir ürünün hangi mikroservis bileşenlerini barındırdığını ve bu bileşenlerin genel görevlerini **üst seviye bakış açısıyla** sunmaktır.

Detaylı yazılım gereksinimleri ve fonksiyonel açıklamalar, ilgili **analiz ve tasarım dokümanlarında** yer almaktadır.

A screenshot of a computer screen

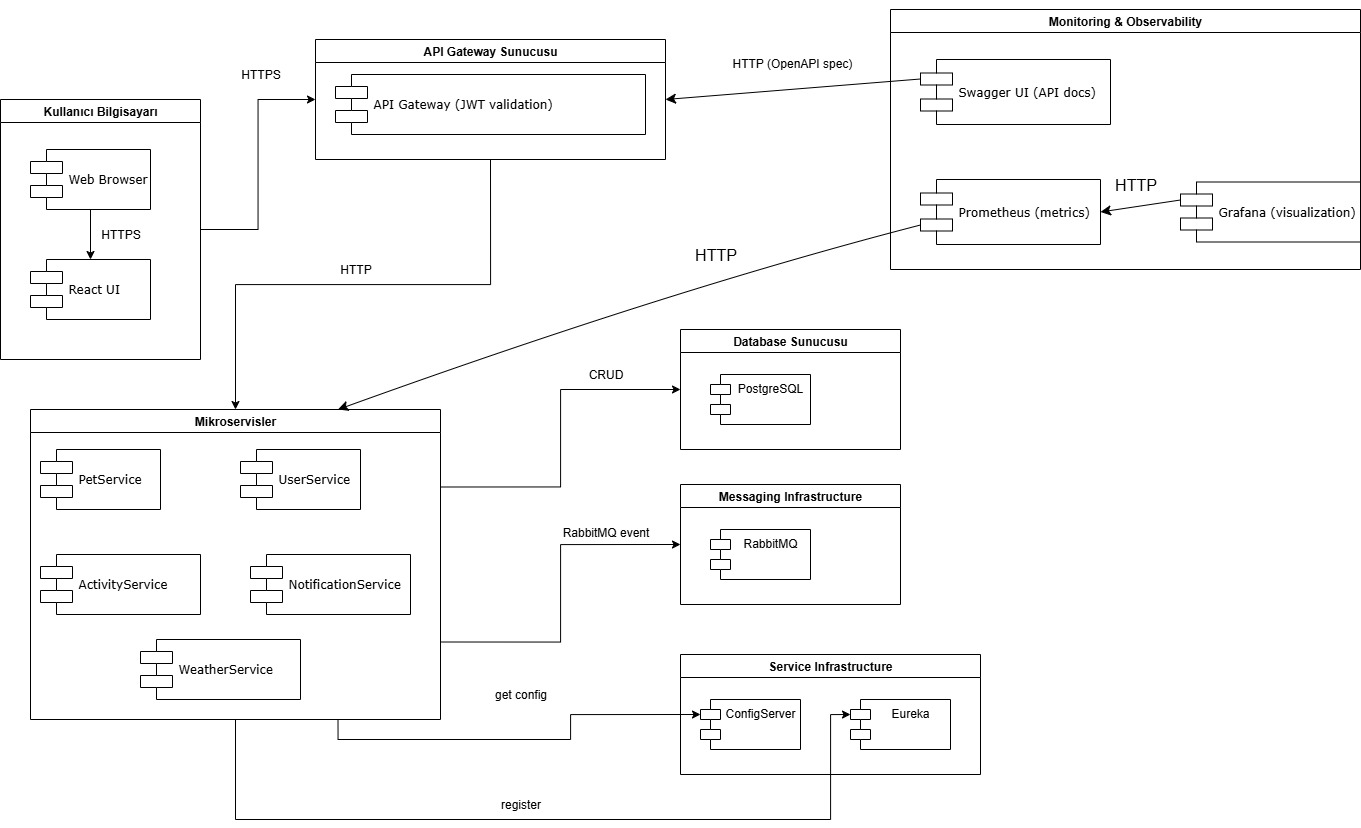
AI-generated content may be incorrect.

## Kurulum Görünümü

Kurulum görünümünde temel olarak PetVerse sisteminin dağıtık yapısı ve bileşenler arası etkileşimleri sunulmuştur. Bu görünüm, sistemin yazılım bileşenlerinin birbirleriyle nasıl haberleştiğini ve hangi altyapı sistemlerinden faydalandığını göstermektedir:

* Şema dili olarak UML 2.5 kullanılmıştır.
* Sistem mikroservis mimarisiyle geliştirilmiş olup bileşenler servis tabanlı yapıdadır.
* Fiziksel sunucu katmanları yerine bileşenler mantıksal gruplarla temsil edilmiştir.
* Her mikroservis kendi görevine odaklanmış, bağımsız olarak ölçeklenebilir ve deploy edilebilir şekilde yapılandırılmıştır.

.



### Mimari İç Elemanlar

### *API Gateway Sunucusu*

* + - * + **Açıklama:** Dış dünyadan gelen istekleri karşılayan, JWT doğrulaması yaparak mikroservislere yönlendiren giriş noktasıdır.
        + **Tip:** Sunucu
        + **İşlev:** Kimlik doğrulama (JWT), yönlendirme (routing)
        + **İletişim Protokolü** : HTTPS
      1. *Mikroservisler*
* **Açıklama:** PetVerse sistemi, özelleşmiş görevleri olan bağımsız mikroservislerden oluşmaktadır. Her bir servis kendi işlevini yerine getirir ve REST/AMQP protokolleri üzerinden iletişim kurar.
* **Servisler ve Görevleri:**
  + **UserService:** Kullanıcı kayıt, giriş ve JWT üretimi işlemlerini gerçekleştirir. Kimlik doğrulama (authentication) ve yetkilendirme (authorization) süreçlerinden sorumludur.
  + **PetService:** Kullanıcıya ait evcil hayvanların oluşturulması, görüntülenmesi ve silinmesi gibi temel CRUD işlemlerini sağlar.
  + **ActivityService:** Evcil hayvanlara ait aktivitelerin (ör. oyun, beslenme, yürüyüş) kaydını yapar. Bu aktiviteleri NotificationService için RabbitMQ üzerinden yayınlar. Ayrıca yapay zeka desteğiyle aktiviteye karşılık gelen evcil hayvan ruh hali (mood) tahmini de yapılır.
  + **NotificationService:** RabbitMQ’dan gelen aktiviteleri dinler ve ilgili kullanıcıya sistem içi ya da e-posta bildirimi oluşturur.
  + **WeatherService:** Harici hava durumu API’lerinden anlık verileri alır. Yürüyüş gibi dış mekan aktiviteleri için uygun hava koşullarını değerlendirerek yönlendirme sağlar.
* **İşletim Sistemi:** Linux (Docker container ortamında çalışmaktadır)
* **İletişim Protokolleri**: HTTP (REST), AMQP (RabbitMQ)
  + - 1. *Veritabanı Sunucusu*
         * **Açıklama:** Mikroservislerin verilerini kalıcı olarak saklamak için kullanılan veritabanı sunucusudur.
         * **Veritabanı:** PostgreSQL
         * **İşlev:** Veri saklama, okuma ve güncelleme işlemlerini gerçekleştirme
         * **İşletim Sistemi:** Linux
         * **Tip:** Altyapı bileşeni
      2. *Mesajlaşma Altyapısı*
         * **Açıklama:** Mikroservisler arası mesajlaşma altyapısıdır. NotificationService gibi servisler event üretir.
         * **Bileşen:** RabbitMQ
         * **İşlev:** Event yayınlama ve tüketme
         * **Protokol:** AMQP
         * **Tip:** Altyapı bileşeni
      3. *Gözlemlenebilirlik Bileşenleri*
* **Açıklama**: Servislerin performans, sağlık ve metrik verilerinin izlenmesini sağlar.
* **Bileşenler**:

Prometheus: Servislerden metrik verileri toplar.

Grafana: Toplanan metrikleri panolar (dashboard) şeklinde görselleştirir.

* **Tip**: İzleme ve analiz altyapı bileşeni
  + - 1. *Test Altyapısı*
* **Açıklama**: PetVerse sisteminde yazılım kalitesini güvence altına almak amacıyla birim testler (unit test), entegrasyon testleri ve sistem testleri uygulanmaktadır. Test süreçleri otomasyon pipeline’ı içerisinde sürekli entegrasyon (CI) kapsamında çalıştırılmaktadır.
* **Bileşenler**:

JUnit 5: Java tabanlı birim test framework’üdür. Servis katmanlarının doğru çalışmasını doğrulamak için kullanılır.

Testcontainers: PostgreSQL gibi bağımlılıkların test ortamında izole olarak başlatılmasını sağlar. Gerçek veritabanı ortamında entegrasyon testleri çalıştırılır.

Mockito: Bağımlılıkları taklit ederek izole birim testler gerçekleştirilmesini sağlar.

CI/CD Entegrasyonu: GitHub Actions üzerinden test adımları otomatik olarak çalıştırılmakta, hatalı kodlar merge edilmeden engellenmektedir.

* **Tip**: Test altyapısı bileşeni
  + - 1. *Kod Kalitesi Analiz Altyapısı(SonarQube)*
* **Açıklama:** PetVerse sisteminde yazılımın okunabilirliğini, güvenliğini ve sürdürülebilirliğini sağlamak amacıyla SonarQube kullanılmıştır. Kod kalitesi analizleri CI/CD süreçlerine entegre edilmiştir.
* **Bileşen:**

**SonarQube (Community Edition):** Java, Spring Boot tabanlı mikroservislerde kod hatalarını, karmaşıklığı ve güvenlik açıklarını analiz eder.

* **İşlev:**

Her push işleminden sonra kod analizleri otomatik çalışır.

Kodun teknik borç oranı, güvenlik açıkları ve code smell’ler görselleştirilir.

* **CI/CD Entegrasyonu:**

GitHub Actions ile sonar-maven-plugin üzerinden otomatik tarama yapılmaktadır.

* **Tip:** Kod kalitesi altyapı bileşeni
  + - 1. *Servis Altyapısı*
* **Açıklama:** Mikroservislerin yapılandırma ve servis keşif işlemlerini yöneten altyapı bileşenleridir.
* **Bileşenler:** Config Server, Eureka Server
* **İşlevler:** Yapılandırma yönetimi, servis kayıt ve keşfi
* **Protokol:** HTTP
* **Tip:** Altyapı sunucusu

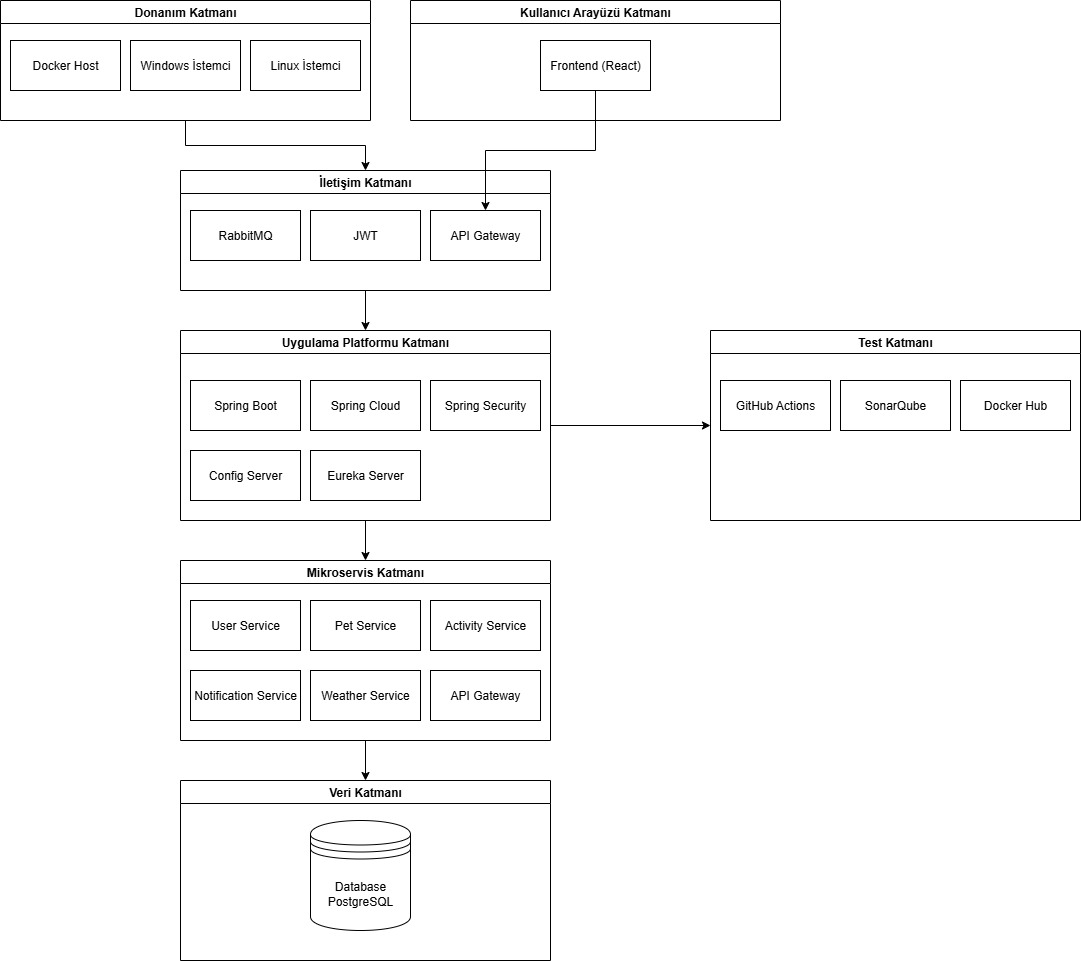
### Mimari Dış Elemanları

### *Kullanıcı Bilgisayarı*

* + - * + **Açıklama**: HTML5 uyumlu web tarayıcısı çalıştırabilen herhangi bir istemci cihazdır. Kullanıcı, bu cihaz üzerinden React tabanlı kullanıcı arayüzüne erişir ve API Gateway aracılığıyla sistemle iletişim kurar. Web tarayıcı, istemci taraflı uygulama olarak HTTPS protokolü üzerinden çalışır.
        + **Tip**: Bilgisayar (İstemci cihaz) / Web tarayıcı (İstemci uygulama)
        + **İşlev**: Kullanıcının arayüz üzerinden sisteme erişmesini ve işlemleri başlatmasını sağlar
        + **İletişim Protokolleri**: TCP/IP, HTTP, HTTP

## Mimari Katman Görünümü

Bu bölümde, PetVerse sistemi kapsamında geliştirilen yazılım bileşenlerinin mimari katmanlardaki yerleşimi sunulmuştur. PetVerse mimarisi, 3 fiziksel ve 4 mantıksal katmandan oluşacak şekilde modellenmiştir. Her katman, sistemin belirli sorumluluklarını üstlenen bileşenleri barındırmakta ve bu yapı, sistemin modüler, yönetilebilir ve genişletilebilir olmasını sağlamaktadır. Aşağıda sunulan mimari katman görünüm diyagramı, sistemin güncel mimari yapısını temsil etmektedir. Gelecekte yapılacak iyileştirme veya modül ekleme/silme işlemleri bu diyagrama da yansıtılacaktır. Katmanlara ait açıklamalar, ilgili bileşenler ile birlikte detaylı olarak sunulmuştur.



**Donanım Katmanı**

* **Açıklama:** PetVerse sisteminin çalıştığı fiziksel ve sanal altyapıyı temsil eder. Docker container ortamı üzerinde çalışan servislerin işletim sistemleri ile olan ilişkisi bu katmanda ele alınmıştır. Sistem Windows işletim sistemi üzerinde WSL (Linux alt sistemi) kullanılarak Docker host üzerinden çalıştırılmaktadır.
* **Tipi:** Fiziksel Katman

**Kullanıcı Arayüzü Katmanı**

* **Açıklama:** Kullanıcıların PetVerse sistemine erişim sağladığı kullanıcı arayüzüdür. React.js tabanlı geliştirilmiş olan frontend uygulaması, API Gateway üzerinden mikroservislerle iletişime geçer. Bu katman, kullanıcı ile sistem arasındaki görsel ve fonksiyonel etkileşimi sağlar.
* **Tipi:** Mantıksal Katman

**İletişim Katmanı**

* **Açıklama:** PetVerse mikroservisleri arasındaki veri iletimi bu katmanda gerçekleşir. API Gateway HTTP isteklerini ilgili servislere yönlendirirken; JWT ile güvenli iletişim sağlanmakta ve RabbitMQ üzerinden event-driven haberleşme yürütülmektedir. Katman, sistemin dışa açık servislerini de kapsar.
* **Tipi:** Mantıksal Katman

**Uygulama Platformu Katmanı**

* **Açıklama:** PetVerse sisteminde kullanılan altyapı teknolojilerini temsil eder. Spring Boot uygulamaları bu katman içinde çalışmakta, Config Server ile konfigürasyon yönetimi yapılmakta ve Eureka Server ile servis keşfi sağlanmaktadır. Spring Security, servislerin güvenliğini sağlayan mekanizmaları içerir.
* **Tipi:** Mantıksal Katman

**Mikroservis Katmanı**

* **Açıklama:** PetVerse projesinin iş mantığını gerçekleştiren mikroservisleri içerir. User Service kullanıcı yönetiminden, Pet Service evcil hayvan bilgilerinden, Activity Service aktivite takibinden, Notification Service bildirimlerden, Weather Service ise hava durumu verilerinin toplanmasından ve aktivitelerle ilişkilendirilmesinden sorumludur. API Gateway, kullanıcı isteklerini bu servislere yönlendirir.
* **Tipi:** Mantıksal Katman

**Veri Katmanı**

* **Açıklama:**   
  Veri Katmanı, PetVerse mikroservislerinin kalıcı veri ihtiyaçlarını karşılayan fiziksel katmandır. Her mikroservis, kendi veritabanı bağlantısı üzerinden veri işlemlerini gerçekleştirir. Bu katmanda, PostgreSQL veritabanı yönetim sistemi kullanılmıştır.
* **Tipi:**Fiziksel-Katman

**Test Katmanı**

* **Açıklama**:  
  Bu katman, sistemin yazılım geliştirme sürecinde **kod kalitesinin korunmasını**, **birim testlerin otomatik çalıştırılmasını** ve **devamlı entegrasyon (CI)** işlemlerinin yönetilmesini sağlar.  
  Bu sayede geliştirilen her bileşen belirli kalite standartlarına uygun şekilde derlenir, test edilir ve güvenle dağıtıma hazırlanır.
* **Bileşenler**:  
  GitHub Actions, SonarQube, Docker Hub
* **İşlevler**:
  + Her kod güncellemesinde test senaryolarının otomatik olarak çalıştırılması
  + Kodun statik analizi ve kalitesinin SonarQube üzerinden izlenmesi
  + Docker image’larının oluşturulup Docker Hub’a gönderilmesi
* **Tipi**: Mantıksal Katman

## Yazılım Geliştirme Görünümü

### Yazılım Geliştirme Ortamı

PetVerse sistemi kapsamında geliştirilen ürünlerin hangi yazılım geliştirme araçları kullanılarak geliştirildiği aşağıda servis bazında belirtilmiştir. Geliştirilen her mikroservis farklı görevleri yerine getirmekte olup, birbirinden bağımsız olarak geliştirilmektedir. Tüm servisler Docker üzerinden PostgreSQL, RabbitMQ gibi altyapı servislerine bağlanmakta ve merkezi olarak Eureka üzerinden keşfedilmektedir.

PetVerse sistemi kapsamında geliştirilen mikroservislerde kullanılan yazılım geliştirme araçları aşağıda belirtilmiştir. Tüm geliştirme süreçleri Visual Studio Code üzerinde yürütülmekte, kaynak kodlar GitHub üzerinden yönetilmektedir.

*UserService, PetService, ActivityService, NotificationService, WeatherService için kullanılan araçlar*

* **IDE**: IntelliJ – Visual Studio Code
* **Uygulama Sunucusu/Application Container:** Docker
* **Veritabanı**: PostgreSQL (Docker container ile)
* **Repo:** GitHub – [PetVerse](https://github.com/yagmurbasoglu/PetVerse)
* **Proje Yapılandırma:** Apache Maven (Lisanssız)
* **Konfigürasyon Yönetimi:** Spring Cloud Config Server
* **Servis Keşfi:** Eureka Server
* **Mesajlaşma:** RabbitMQ (Bildirimler için event bazlı kullanım planlanmıştır)
* **HTTP İstemcisi:** WebClient (özellikle ActivityService için HuggingFace API erişiminde)
* **Yapay Zeka API Entegrasyonu:** HuggingFace Inference API
* **Hava Durumu Entegrasyonu:** Open-Meteo API (WeatherService aracılığıyla)
  + - 1. *API Gateway için kullanılan araçlar*
* **IDE**: IntelliJ - Visual Studio Code
* **Repo:** GitHub – [PetVerse](https://github.com/yagmurbasoglu/PetVerse)
* **Proje Yapılandırma:** Apache Maven (Lisanssız)
* **Gateway Teknolojisi**: Spring Cloud Gateway
* **Güvenlik:** JWT filtreleme, rate limiting (Redis destekli)
  + - 1. *Test ve CI/CD için kullanılan araçlar*
* **IDE/Test Ortamı**: Visual Studio Code, IntelliJ IDEA
* **Test Çatısı**: JUnit 5, Mockito
* **Entegrasyon Testi**: Testcontainers (Docker tabanlı test veritabanı ile)
* **Kod Kalitesi Analizi**: SonarQube (Community Edition)
* **CI/CD**: GitHub Actions (Her develop push/pull request'inde otomatik test ve kalite analizi)
* **Container Registry**: Docker Hub

### Test Ortamı

PetVerse sistemi kapsamında geliştirilen servislerin ve kullanıcı arayüzünün test süreçlerinde kullanılan araçlar aşağıda belirtilmiştir. Geliştirilen servisler Java ve Spring Boot temelli olup, test işlemleri yerel geliştirme ortamlarında gerçekleştirilecektir..

* + - 1. *Backend (Java Spring Boot) Servisleri için Test Araçları*
* **Birim Testleri ve Debug için IDE**: Visual Studio Code, IntelliJ IDEA
* **Web Servis Test Araçları:** IntelliJ HTTP Client (IDE üzerinden) veya Postman (Açık Kaynak)
* **Test Otomasyon Aracı:** JUnit 5 , Mockito
* **Veritabanı Erişim:** PgAdmin (Açık Kaynak)
* **Statik Kod Analizi:** Kodların analizi periyodik olarak SonarQube analiz aracı ile düzenli olarak standart Java ve FrontEnd kuralları ile kontrol edilmektedir. İhtiyaca göre özelleşmiş kural setleri kullanılabilecektir

### Entegrasyon ve Teslimat Süreçleri

PetVerse projesinde, entegrasyon ve teslimat süreçleri Git versiyon kontrol sistemi kullanılarak yürütülmektedir. Geliştirme süreci Git Flow modeline uygun olarak sürdürülmekte, her bir özellik dalı (feature/) geliştirme tamamlandıktan sonra develop dalına merge edilmektedir.

Proje kapsamında CI/CD (Continuous Integration / Continuous Delivery) süreçleri **tamamen otomatik hale getirilmiştir**. Bu kapsamda:

* **GitHub Actions** kullanılarak her push veya pull request sonrasında:
  + Birim testleri (JUnit, Testcontainers) otomatik çalıştırılmakta,
  + Kod kalitesi **SonarQube** ile analiz edilmekte,
  + Docker imajları oluşturularak **DockerHub** üzerine yüklenmektedir.

Bu sayede kod kalitesi güvence altına alınmakta ve dağıtıma hazır Docker imajları her versiyonda otomatik olarak oluşturulmaktadır.

* + - 1. *Git Flow Yapısı*

PetVerse projesinde Git tabanlı sürüm kontrolü uygulanmakta olup, aşağıdaki Git Flow yaklaşımı izlenmektedir:

* Ana akış main branch üzerinden yürütülmektedir. Yayınlanmaya hazır kodlar burada tutulur.
* Günlük geliştirme faaliyetleri dev branch üzerinde sürdürülmektedir.
* Her mikroservisin kendi içinde bağımsız olarak geliştirme yapılabilmesi için gerekirse servis bazlı feature branch yapısı (örn. feature/pet-add) açılmaktadır.
* Servis tamamlandığında pull request ile development branch’e merge edilmektedir.
* Merge sonrası **GitHub Actions CI/CD süreci otomatik tetiklenir**. Bu süreçte:  
  Birim testleri (JUnit/Testcontainers) çalıştırılır  
  Kod kalitesi SonarQube ile analiz edilir  
  Docker imajı build edilerek DockerHub’a gönderilir
* Her versiyon veya teslimat için main branch’e merge işlemi yapılır ve sürüm etiketi (tag) oluşturularak yayınlanır.
* Docker container’lar ile deployment işlemi yürütülmekte, servis konfigürasyonları configserver altında merkezi olarak tutulmaktadır.
* Her servis ayrı pom.xml dosyasına sahip bağımsız build edilen modüllerdir.
  + - 1. *Entegrasyon ve Teslimat Stratejisi*

**PetVerse** sistemi kapsamında geliştirilen ürünlerde **Git Flow** sürüm kontrol modeli uygulanmakta olup, her mikroservis aynı test adımlarından geçerek **GitHub Actions** tabanlı otomatik CI/CD süreçleri ile **Docker Compose** ortamında test edilip deploy edilmektedir.

* + - * 1. Sürekli Entegrasyon Stratejisi CI/CD

**PetVerse** projesinde otomatik entegrasyon ve teslimat stratejisi aktif olarak uygulanmaktadır. Tüm mikroservislerde gerçekleştirilen her kod değişikliği, **GitHub Actions** pipeline'ı tarafından test edilmekte ve otomatik olarak Docker imajı oluşturularak **deploy** edilmektedir.

Planlanan CI/CD iş akışı aşağıdaki gibidir:

Geliştirme işlemleri feature/ branch'lerinde yapılır.

**GitHub Actions**, push veya pull request tetiklenince CI sürecini başlatır.

Kodlar **unit test** ve **integration test** (JUnit & Testcontainers) süreçlerinden geçer.

Eğer birim testlerinde hata varsa, otomatik olarak bildirim gönderilir (örneğin **GitHub Issues** veya **Slack**).

Testler başarılı olursa:

* **Docker imajı** otomatik olarak oluşturulur,
* Docker Hub'a push edilir ve
* **docker-compose** ortamında ilgili servise **deploy** edilir

Not: CI/CD süreci tüm servislerde standart hale getirilmiş olup, sistemin otomatize çalışması sağlanmıştır. Gelecekte bu süreçlerin kalite analiz aşamaları (örneğin **SonarQube**) ve ortam geçişleri (staging → prod) daha da detaylandırılacaktır.

* + - * 1. Manuel Entegrasyon Stratejisi

**PetVerse** sisteminde CI/CD altyapısı tam olarak devreye alınmadan önce, manuel entegrasyon stratejisi uygulanmaktaydı. Entegrasyon süreci **BOTTOM-UP** (aşağıdan yukarıya) yaklaşımı ile yürütülmüştür.

**Bu yaklaşımda:**

* Proje mikroservis mimarisine sahip olduğu için, bağımsız geliştirilen bileşenler (örneğin UserService, PetService, ActivityService, NotificationService) öncelikle ayrı ayrı test edilerek manuel olarak deploy edilmiştir.
* Ortak yapı taşı olan API Gateway, en son aşamada test edilip devreye alınmıştır.
* Her servis, geliştirici ortamında **Visual Studio Code** kullanılarak build ve run komutları ile çalıştırılmıştır.
* PostgreSQL, RabbitMQ, Eureka gibi altyapı bileşenleri **Docker Compose** ile ayağa kaldırılmıştır.
* Entegrasyon testleri tamamlandıktan sonra tüm bileşenler **staging** ortamında birlikte çalıştırılmıştır.

**Bu yöntemde dikkat edilen hususlar:**

* Her servisin yapılandırması ve veritabanı bağlantıları application.yml dosyası üzerinden manuel olarak kontrol edilmiştir.
* NotificationService üzerinden test mesajları gönderilerek bildirim akışı manuel olarak doğrulanmıştır.
* Frontend bileşeni henüz geliştirilmediği için, API Gateway üzerinden yapılan testlerle backend servislerinin doğru çalıştığı kontrol edilmiştir.

.

### Uygulanan Teknolojiler

**Java 21**

**Spring Boot Framework**

**Spring Security**

**Spring Cloud Gateway**

**Spring Cloud Netflix Eureka**

**Spring Cloud Config**

**Apache Maven**

**JUnit**

**PostgreSQL**

**RabbitMQ**

**Docker & Docker Compose**

**JWT (JSON Web Token)**

**GitHub Actions**

**SonarQube (Community Edition)**

# Mimari Riskler

## Atıf Verilen Dokümanlar

PetVerse mimari tasarım dokümanında, sistemin geliştirilmesine temel teşkil eden çeşitli referans dokümanlara atıf yapılmıştır. Bu kapsamda iki temel mimari risk değerlendirilmiştir:

1. Atıf yapılan dokümanların içeriklerinin bu dokümana doğrudan kopyalanması durumunda, ilgili dokümanlar güncellendiğinde bu dokümanın eski bilgi içermesi riski ortaya çıkmaktadır. Atıf yapılan dokümanların içeriklerinin bu dokümana konulmayıp yalnızca referans verilmesi durumunda ise erişim engelleri (yetkisizlik, dosya kaybı vb.) nedeniyle bilgiye erişilememe riski oluşmaktadır.

Bu riskler, ilgili içeriklerin mümkün olduğunca özetlenerek bu dokümana aktarılması ya da erişim imkânı olmayanlar için alternatif bilgi kaynaklarıyla desteklenmesi yoluyla azaltılmaya çalışılmıştır.

## Hazır Bileşenlerin Farklı Bölümlerde Yer Alması

PetVerse sistemi kapsamında geliştirilen mikroservislerde birçok hazır bileşen (Spring Boot, Spring Security, JWT, RabbitMQ, PostgreSQL vb.) kullanılmaktadır. Mimari tasarım dokümanının 5.1 Yazılım Ürünleri Görünümü, 5.2 Modül Bağımlılıkları, 5.3 Kurulum Görünümü ve 5.4 Uygulanan Teknolojiler bölümlerinde bu bileşenler tekrar eden biçimlerde sunulmuştur.

Bu sunumlar kimi zaman açıklayıcı diyagramlar, kimi zaman ise metinsel özetler şeklinde yer almaktadır. Hazır bileşenlerde yapılacak güncellemelerin her bölüme eş zamanlı olarak yansıtılmaması; dokümanda bilgi tutarsızlığına ve bazı güncellemelerin gözden kaçmasına neden olabileceği için risk teşkil etmektedir.

Bu nedenle, güncellenebilir bileşenlerin merkezi olarak yönetilmesi ve ilgili tüm bölümlere dikkatlice yansıtılması gerekmektedir.

## Microservis Mimarisi ve Dağıtık Sistem Riskleri

PetVerse sistemi; kullanıcı, evcil hayvan, aktivite ve bildirim gibi işlevleri kapsayan bir yazılım ürünü olarak, monolitik mimari yerine mikroservis mimarisi kullanılarak geliştirilmiştir. Bu tercih ile her işlevsel bileşen (örneğin UserService, PetService, ActivityService, NotificationService) birbirinden bağımsız servisler olarak ele alınmıştır. Monolitik bir yapının tercih edilmesi durumunda, tüm işlevlerin tek bir uygulama altında toplanması; ölçeklenebilirlik, bakım, güncelleme ve hata ayıklama süreçlerinde ciddi zorluklar doğurabilir, bir bileşendeki hata tüm sistemi etkileyebilir ve küçük değişiklikler dahi tüm sistemin yeniden dağıtılmasını gerektirebilirdi.

Bu nedenlerle PetVerse projesinde bağımsız geliştirme, hızlı dağıtım, hata izolasyonu ve esnek ölçeklenebilirlik gibi avantajlar sunan mikroservis mimarisi tercih edilmiştir. Bu yapı sayesinde her servis, kendi yaşam döngüsüne sahip şekilde yönetilebilmekte, teknolojik olarak özelleştirilebilmekte ve gerektiğinde bağımsız olarak devreye alınabilmektedir.

Ancak, mikroservis mimarisinin sunduğu bu avantajların yanında bazı riskleri de beraberinde getirdiği unutulmamalıdır. Öncelikle, hizmetler arası iletişimin HTTP ya da mesajlaşma altyapıları (örneğin RabbitMQ) üzerinden sağlanması sistemde gecikmelere, bağlantı problemlerine ve hata yönetimi karmaşasına yol açabilir. Ayrıca, her mikroservisin kendi veritabanını kullanması nedeniyle, sistem genelinde veri tutarlılığı garanti edilemez; bu da geçici tutarsızlık (eventual consistency) gibi durumların ortaya çıkmasına neden olabilir. Servislerin birbirine olan bağımlılığı, versiyonlama ve geri uyumluluk gibi konularda ek zorluklar doğururken, sistemin genel sağlığını izlemek için merkezi loglama, dağıtık izleme (distributed tracing) ve metrik takibi gibi ek araçların kurulması zorunlu hale gelmektedir. Tüm bu yapı ağ tabanlı çalıştığı için, mikroservislerin taşıdığı riskler arasında ağ gecikmeleri, zaman aşımı sorunları ve güvenlik açıkları da önemli bir yer tutmaktadır.

Sonuç olarak, mikroservis mimarisi PetVerse sistemi için esneklik, modülerlik ve bağımsızlık gibi önemli avantajlar sağlamaktadır. Ancak, dağıtık mimarinin getirdiği teknik zorlukların ve operasyonel karmaşıklıkların göz önünde bulundurulması, sistemin sürdürülebilirliği açısından kritik öneme sahiptir. Bu bağlamda, PetVerse projesinde gözlemlenebilirlik, hata yönetimi ve güvenlik gibi konulara özel önem verilmekte ve ilgili altyapılar geliştirilmektedir.