

USER SERVICE

Servis Mimari Dökümanı

E-Commerce Platform — Kimlik & Profil Yönetimi

Go · Fiber v2 · GORM · PostgreSQL · Redis · Keycloak · JWT

1. Genel Bakış

user-service, e-commerce platformunun kimlik doğrulama ve kullanıcı profil yönetiminden sorumlu mikroservisidir. Keycloak ile entegre çalışır; token doğrulama, profil yönetimi ve Nginx auth_request mekanizması için /internal/auth/validate endpoint'ini sağlar.

1.1 Servis Bilgileri

Özellik	Değer
Servis Adı	user-service
Port	8081
Veritabanı	PostgreSQL — users_db
Cache	Redis
Identity Provider	Keycloak 24
Framework	Go Fiber v2
ORM	GORM
Modül Adı	github.com/yasinbozat/ecommerce-platform/services/user-service

1.2 Sorumluluklar

- Keycloak üzerinden gelen JWT token'larını doğrulamak (/internal/auth/validate)
- Kullanıcı profil bilgilerini yönetmek (ad, soyad, telefon)
- Kullanıcı adres bilgilerini yönetmek
- Nginx auth_request mekanizması için token introspection yapmak
- X-User-ID ve X-User-Role header'larını Nginx'e döndürmek

1.3 Sorumluluk Dışı

- Kullanıcı kaydı ve girişi — Keycloak yönetir
- Şifre yönetimi — Keycloak yönetir
- Token üretimi — Keycloak yönetir
- Sipariş, ürün, ödeme işlemleri — ilgili servisler yönetir

2. Klasör Yapısı

user-service, Clean Architecture prensiplerine göre katmanlı bir yapıda organize edilmiştir. Her katmanın tek bir sorumluluğu vardır ve bağımlılık yönü her zaman dıştan içe doğrudur.

```
user-service/
├── cmd/
│   └── main.go                                # Uygulamanın giriş noktası
├── internal/
│   ├── domain/                               # Katman 1: İş Alanı
│   │   ├── user.go                          # User, Address entity'leri
│   │   ├── dto.go                          # Request/Response struct'ları
│   │   └── errors.go                        # Servis hata tipleri
│   ├── repository/                          # Katman 2: Veri Erişimi
│   │   ├── user_repository.go              # IUserRepository interface
│   │   ├── address_repository.go          # IAddressRepository interface
│   │   └── postgres/                      # PostgreSQL implementasyonları
│   │       ├── user_postgres.go
│   │       └── address_postgres.go
│   ├── service/                             # Katman 3: İş Mantığı
│   │   ├── user_service.go                # IUserService interface + impl
│   │   └── auth_service.go                # IAuthService interface + impl
│   ├── handler/                             # Katman 4: HTTP
│   │   ├── user_handler.go                # Profil endpoint'leri
│   │   ├── auth_handler.go                # /internal/auth/validate
│   │   └── address_handler.go              # Adres endpoint'leri
│   ├── middleware/                          # Fiber middleware'leri
│   │   ├── keycloak.go                    # Token doğrulama middleware
│   │   └── role.go                        # Role bazlı yetkilendirme
│   └── config/                              # Konfigürasyon
│       ├── config.go                      # Config struct
│       └── database.go                    # GORM bağlantısı
├── migrations/                              # SQL migration dosyaları
│   ├── 000001_create_users.up.sql
│   ├── 000001_create_users.down.sql
│   ├── 000002_create_addresses.up.sql
│   └── 000002_create_addresses.down.sql
├── .env.example                             # Örnek environment değişkenleri
├── Dockerfile                              # Multi-stage build
├── go.mod
└── go.sum
```

2.1 Katman Bağımlılık Kuralı

Bağımlılıklar her zaman dıştan içe doğrudur. İçteki katmanlar dıştaki katmanları bilmez:

Katman	Bağımlı Olduğu	Bağımlı Olmadığı
domain/	Hiçbir şey (saf Go)	Fiber, GORM, Redis, Keycloak
repository/	domain/	Fiber, service/
service/	domain/, repository/	Fiber, handler/
handler/	domain/, service/	repository/, GORM

3. Veritabanı Şeması

user-service, PostgreSQL'de users_db veritabanını kullanır. GORM AutoMigrate yerine golang-migrate ile yönetilen SQL migration dosyaları tercih edilir. Bu sayede migration geçmişi izlenebilir ve rollback yapılabilir.

3.1 users Tablosu

Kolon	Tip	Kısıt	Açıklama
id	UUID	PK, DEFAULT gen_random_uuid()	Birincil anahtar
email	VARCHAR(255)	UNIQUE, NOT NULL	Kullanıcı email adresi
full_name	VARCHAR(255)	NOT NULL	Ad Soyad
phone	VARCHAR(20)	NULL	Telefon numarası
role	VARCHAR(20)	DEFAULT 'customer'	customer veya admin
keycloak_id	VARCHAR(255)	UNIQUE, NOT NULL	Keycloak user ID (sub claim)
is_active	BOOLEAN	DEFAULT true	Hesap aktif mi
created_at	TIMESTAMPTZ	NOT NULL	Kayıt tarihi
updated_at	TIMESTAMPTZ	NOT NULL	Son güncelleme

3.2 addresses Tablosu

Kolon	Tip	Kısıt	Açıklama
id	UUID	PK, DEFAULT gen_random_uuid()	Birincil anahtar
user_id	UUID	FK → users.id, NOT NULL	Kullanıcı referansı
title	VARCHAR(100)	NOT NULL	Adres başlığı (Ev, İş vb.)
full_name	VARCHAR(255)	NOT NULL	Alıcı adı
phone	VARCHAR(20)	NOT NULL	Alıcı telefonu
street	TEXT	NOT NULL	Cadde / sokak / bina
district	VARCHAR(100)	NOT NULL	İlçe
city	VARCHAR(100)	NOT NULL	Şehir
zip_code	VARCHAR(10)	NULL	Posta kodu
is_default	BOOLEAN	DEFAULT false	Varsayılan adres
created_at	TIMESTAMPTZ	NOT NULL	Oluşturma tarihi
updated_at	TIMESTAMPTZ	NOT NULL	Son güncelleme

3.3 Migration Dosya Yapısı

golang-migrate aracı kullanılır. Her migration için bir up (ileri) ve bir down (geri) dosyası bulunur. Dosya isimlendirme: {versiyon}_{açıklama}.{up|down}.sql

Dosya	İçerik
000001_create_users.up.sql	users tablosunu oluşturur, indexleri ekler
000001_create_users.down.sql	users tablosunu siler
000002_create_addresses.up.sql	addresses tablosunu oluşturur, FK ekler
000002_create_addresses.down.sql	addresses tablosunu siler

4. Domain Katmanı

Domain katmanı, servisin temel veri yapılarını barındırır. Hiçbir harici kütüphaneye bağımlı değildir. Tüm diğer katmanlar bu katmandaki tipleri kullanır.

4.1 Entity'ler (domain/user.go)

Struct	Alan	Tip	Açıklama
User	ID	uuid.UUID	Birincil anahtar
User	Email	string	Kullanıcı emaili
User	FullName	string	Ad Soyad
User	Phone	string	Telefon
User	Role	Role (string)	customer veya admin
User	KeycloakID	string	Keycloak sub claim
User	IsActive	bool	Hesap aktif mi
User	CreatedAt	time.Time	Kayıt tarihi
User	UpdatedAt	time.Time	Güncelleme tarihi
Address	ID	uuid.UUID	Birincil anahtar
Address	UserID	uuid.UUID	FK → User
Address	Title	string	Adres başlığı
Address	FullName	string	Alıcı adı
Address	Phone	string	Alıcı telefonu
Address	Street	string	Cadde/sokak
Address	District	string	İlçe
Address	City	string	Şehir
Address	ZipCode	string	Posta kodu
Address	IsDefault	bool	Varsayılan mı

4.2 DTO'lar (domain/dto.go)

Struct	Kullanım Yeri	Alanlar
UpdateProfileRequest	PUT /users/me	full_name, phone
UserResponse	GET /users/me response	id, email, full_name, phone, role
CreateAddressRequest	POST /users/me/addresses	title, full_name, phone, street, district, city, zip_code, is_default
UpdateAddressRequest	PUT /users/me/addresses/:id	Aynı alanlar, hepsi opsiyonel

Struct	Kullanım Yeri	Alanlar
AddressResponse	Adres endpoint response'ları	Tüm address alanları
ValidateTokenResponse	GET /internal/auth/validate	user_id, email, role, keycloak_id

4.3 Hata Tipleri (domain/errors.go)

Hata	Açıklama	HTTP Karşılığı
ErrUserNotFound	Kullanıcı bulunamadı	404
ErrAddressNotFound	Adres bulunamadı	404
ErrAddressNotOwned	Adres bu kullanıcıya ait değil	403
ErrInvalidToken	Token geçersiz veya eksik	401
ErrTokenExpired	Token süresi dolmuş	401
ErrKeycloakUnreachable	Keycloak'a ulaşılamıyor	503
ErrInvalidInput	Validation hatası	400

5. Repository Katmanı

Repository katmanı, veritabanı erişimini soyutlar. Interface tanımları repository/ altında, PostgreSQL implementasyonları repository/postgres/ altında bulunur. Bu yapı sayesinde service katmanı test edilirken mock repository kullanılabilir.

5.1 IUserRepository Interface

Metod	Parametreler	Dönüş	Açıklama
FindByID	ctx, id uuid.UUID	*User, error	ID ile kullanıcı bul
FindByKeycloakID	ctx, keycloakID string	*User, error	Keycloak ID ile kullanıcı bul
FindByEmail	ctx, email string	*User, error	Email ile kullanıcı bul
Create	ctx, user *User	error	Yeni kullanıcı oluştur
Update	ctx, user *User	error	Kullanıcı güncelle
Delete	ctx, id uuid.UUID	error	Kullanıcı sil (soft delete)

5.2 IAddressRepository Interface

Metod	Parametreler	Dönüş	Açıklama
FindByID	ctx, id uuid.UUID	*Address, error	ID ile adres bul
FindAllByUserID	ctx, userID uuid.UUID	[]*Address, error	Kullanıcının tüm adresleri
Create	ctx, address *Address	error	Yeni adres oluştur
Update	ctx, address *Address	error	Adres güncelle
Delete	ctx, id uuid.UUID	error	Adres sil
SetDefault	ctx, id, userID uuid.UUID	error	Varsayılan adres belirle

6. Service Katmanı

Service katmanı, tüm iş mantığını barındırır. Repository interface'lerini kullanır, Fiber veya GORM'dan habersizdir. İki ayrı servis dosyasına bölünmüştür.

6.1 IUserService

Metod	Parametreler	Dönüş	Açıklama
GetProfile	ctx, userID uuid.UUID	*UserResponse, error	Profil bilgilerini getir
UpdateProfile	ctx, userID uuid.UUID, req *UpdateProfileRequest	*UserResponse, error	Profil güncelle
GetAddresses	ctx, userID uuid.UUID	[]*AddressResponse, error	Tüm adresleri listele
CreateAddress	ctx, userID uuid.UUID, req *CreateAddressRequest	*AddressResponse, error	Yeni adres ekle
UpdateAddress	ctx, userID, addressID uuid.UUID, req *UpdateAddressRequest	*AddressResponse, error	Adres güncelle
DeleteAddress	ctx, userID, addressID uuid.UUID	error	Adres sil
SetDefaultAddress	ctx, userID, addressID uuid.UUID	error	Varsayılan adres belirle

6.2 IAuthService

Metod	Parametreler	Dönüş	Açıklama
ValidateToken	ctx context.Context, token string	*ValidateTokenResponse, error	Keycloak token introspect et, kullanıcı bilgilerini döndür
SyncUser	ctx context.Context, keycloakID string	*User, error	Keycloak'tan gelen user'ı DB ile senkronize et, yoksa oluştur

6.3 Auth Service İş Akışı

ValidateToken metodu şu adımları izler:

- Authorization header'dan Bearer token'ı ayıklar
- Keycloak /realms/ecommerce/protocol/openid-connect/token/introspect endpoint'ine istek atar
- Token aktif değilse ErrInvalidToken döner
- Token geçerliyse sub claim'den keycloak_id'yi alır
- users_db'de bu keycloak_id'ye sahip kullanıcıyı arar
- Kullanıcı yoksa otomatik oluşturur (SyncUser)
- X-User-ID ve X-User-Role header değerlerini döndürür

7. Handler Katmanı

Handler katmanı, HTTP isteklerini karşılar, input validasyonu yapar, service katmanını çağırır ve response döner. Fiber framework kullanılır.

7.1 Route Tanımları

Method	Path	Handler	Auth	Açıklama
GET	/internal/auth/validate	AuthHandler.Validate	Internal	Nginx auth_request hedefi
GET	/api/v1/users/me	UserHandler.GetProfile	Keycloak JWT	Profil bilgisi
PUT	/api/v1/users/me	UserHandler.UpdateProfile	Keycloak JWT	Profil güncelle
GET	/api/v1/users/me/addresses	AddressHandler.List	Keycloak JWT	Adres listesi
POST	/api/v1/users/me/addresses	AddressHandler.Create	Keycloak JWT	Yeni adres
PUT	/api/v1/users/me/addresses/:id	AddressHandler.Update	Keycloak JWT	Adres güncelle
DELETE	/api/v1/users/me/addresses/:id	AddressHandler.Delete	Keycloak JWT	Adres sil
PATCH	/api/v1/users/me/addresses/:id/default	AddressHandler.SetDefault	Keycloak JWT	Varsayılan yap
GET	/health	HealthHandler	Public	Health check
GET	/metrics	PrometheusHandler	Public	Prometheus metrikleri

7.2 /internal/auth/validate Detayı

Bu endpoint sadece Nginx tarafından çağrılır, doğrudan dışarıya açık değildir. Nginx'in auth_request direktifi her korumalı isteğe önce bu endpoint'e gider.

	Detay
Method	GET
Path	/internal/auth/validate
Girdi	Authorization: Bearer <token> header
Başarı Response	200 OK
Başarı Headers	X-User-ID: uuid, X-User-Role: customer admin, X-User-Email: email

	Detay
Hata Response	401 Unauthorized (token geçersiz veya süresi dolmuş)
Hata Response	503 Service Unavailable (Keycloak'a ulaşamıyor)

7.3 Handler Akış Şeması

Her handler aşağıdaki akışı izler:

- İstek gelir
 - Fiber context'ten X-User-ID header'ını al (Nginx tarafından set edilmiş)
- Input validation
 - Request body parse et
 - go-playground/validator ile validate et
 - Hata varsa 400 döndür
- Service katmanını çağır
 - İş mantığı hatalarını HTTP kodlarına çevir
- Response döndür
 - Başarı: 200/201 + JSON body
 - Hata: ilgili HTTP status + hata mesajı

8. Middleware

8.1 Keycloak Middleware (middleware/keycloak.go)

Nginx auth_request mekanizması sayesinde bu servisin kendi JWT doğrulama middleware'ine ihtiyacı yoktur. Nginx zaten /internal/auth/validate üzerinden doğrulama yaptırır ve X-User-ID, X-User-Role header'larını ekler.

Bu middleware'in tek görevi şudur: Fiber context'ten Nginx'in eklediği X-User-ID header'ını okuyarak locals'a yazmak, böylece handler'lar kolayca erişebilir.

8.2 Role Middleware (middleware/role.go)

Admin gerektiren endpoint'lerde kullanılır. X-User-Role header'ını kontrol eder, admin değilse 403 döndürür.

Kullanım	Açıklama
RequireRole("admin")	Sadece admin rolündeki kullanıcılar geçer
RequireRole("customer", "admin")	Her iki rol de geçer

9. Konfigürasyon

9.1 Environment Variables (.env.example)

Değişken	Örnek Değer	Açıklama
APP_PORT	8081	Servis portu
APP_ENV	development	development production
DB_HOST	localhost	PostgreSQL host
DB_PORT	5432	PostgreSQL port
DB_NAME	users_db	Veritabanı adı
DB_USER	ecommerce	Veritabanı kullanıcısı
DB_PASSWORD	ecommerce123	Veritabanı şifresi
DB_SSLMODE	disable	disable require
REDIS_ADDR	localhost:6379	Redis adresi
REDIS_PASSWORD	redis123	Redis şifresi
REDIS_DB	0	Redis DB index
KEYCLOAK_URL	http://localhost:8180	Keycloak base URL
KEYCLOAK_REALM	ecommerce	Keycloak realm adı
KEYCLOAK_CLIENT_ID	ecommerce-service	Backend client ID
KEYCLOAK_CLIENT_SECRET	service-secret	Backend client secret
JAEGGER_ENDPOINT	http://localhost:4318/v1/traces	OTLP HTTP endpoint

9.2 Config Struct Yapısı

Alan	Alt Alanlar	Açıklama
AppConfig	Port, Env	Uygulama ayarları
DatabaseConfig	Host, Port, Name, User, Password, SSLMode	PostgreSQL bağlantısı
RedisConfig	Addr, Password, DB	Redis bağlantısı
KeycloakConfig	URL, Realm, ClientID, ClientSecret	Keycloak bağlantısı
JaegerConfig	Endpoint	Tracing ayarları

10. Go Bağımlılıkları

Paket	Versiyon	Kullanım
github.com/gofiber/fiber/v2	v2.52+	HTTP framework
gorm.io/gorm	v1.25+	ORM
gorm.io/driver/postgres	v1.5+	GORM PostgreSQL driver
github.com/google/uuid	v1.6+	UUID üretimi
github.com/joho/godotenv	v1.5+	Env dosyası yükleme
github.com/go-playground/validator/v10	v10+	Input validasyonu
github.com/redis/go-redis/v9	v9+	Redis client (token cache)
go.uber.org/zap	v1.27+	Structured logging
github.com/golang-migrate/migrate/v4	v4+	DB migration
go.opentelemetry.io/otel	v1.24+	Distributed tracing
go.opentelemetry.io/otel/exporters/otlp/otlptrace/otlptracehttp	v1.24+	Jaeger OTLP exporter
github.com/prometheus/client_golang	v1.19+	Prometheus metrics
github.com/stretchr/testify	v1.9+	Test assertion
github.com/stretchr/mock	v1.6+	Mock üretimi (test)

11. Test Stratejisi

11.1 Unit Test

Service katmanı unit test ile test edilir. Repository interface'leri mock'lanır, gerçek veritabanı kullanılmaz.

Test Dosyası	Test Edilen	Mock'lanan
service/user_service_test.go	GetProfile, UpdateProfile, adres metodları	IUserRepository, IAddressRepository
service/auth_service_test.go	ValidateToken, SyncUser	IUserRepository, Keycloak HTTP client
handler/user_handler_test.go	HTTP response kodları, body	IUserService
handler/auth_handler_test.go	Validate endpoint response header'ları	IAuthService

11.2 Integration Test

Repository katmanı integration test ile test edilir. testcontainers-go ile gerçek PostgreSQL container'ı ayağa kaldırılır.

- Test başında PostgreSQL container başlatılır
- Migration'lar çalıştırılır
- CRUD operasyonları gerçek DB'ye karşı test edilir
- Test sonunda container durdurulur

11.3 Test Klasör Yapısı

```
user-service/  
├── internal/  
│   ├── service/  
│   │   ├── user_service.go  
│   │   └── user_service_test.go      # Unit test  
│   ├── handler/  
│   │   ├── user_handler.go  
│   │   └── user_handler_test.go      # Unit test  
│   └── repository/  
│       └── postgres/  
│           ├── user_postgres.go  
│           └── user_postgres_test.go # Integration test  
└── mocks/  
    ├── IUserRepository.go  
    ├── IAddressRepository.go  
    └── IUserService.go
```

12. Dockerfile

Multi-stage build kullanılır. Builder stage'de binary derlenir, final stage'de sadece binary çalıştırılır. Final image boyutu minimumda tutulur.

12.1 Stage'ler

Stage	Base Image	Görev
builder	golang:1.22-alpine	go mod download, go build
final	alpine:3.19	Sadece binary + ca-certificates

12.2 Build Argümanları

Argüman	Değer	Açıklama
CGO_ENABLED	0	CGO kapalı (statik binary)
GOOS	linux	Linux için derleme
GOARCH	amd64	64-bit mimari

12.3 Güvenlik Notları

- Final image'da root kullanıcı kullanılmaz, ayrı bir user oluşturulur
- .env dosyası image'a kopyalanmaz, runtime'da environment variable olarak geçilir
- Sadece gerekli portlar expose edilir (8081)

13. Geliştirme Sırası

user-service'i sıfırdan geliştirirken önerilen adım sırası:

Adım	Görev	Çıktı
1	go.mod ve bağımlılıklar	Tüm paketler yüklü
2	config/ katmanı	Env'den config okuma çalışıyor
3	domain/ katmanı	Entity, DTO, error struct'ları hazır
4	Migration dosyaları	users ve addresses tabloları oluşuyor
5	repository/ interface'leri	IUserRepository, IAddressRepository tanımlı
6	repository/postgres/ implementasyonları	GORM ile CRUD çalışıyor
7	Integration testler	Repository testcontainers ile test edildi
8	service/ katmanı	İş mantığı yazıldı
9	Unit testler	Service katmanı mock ile test edildi
10	handler/ katmanı	Fiber route'ları ve handler'lar yazıldı
11	/internal/auth/validate	Keycloak introspection çalışıyor
12	middleware/ katmanı	Keycloak ve role middleware hazır
13	main.go	Tüm katmanlar bağlandı, servis ayağa kalkıyor
14	Dockerfile	Multi-stage build çalışıyor
15	docker-compose entegrasyonu	Servis altyapı ile birlikte çalışıyor