به نام خدا

### \*\*1. جدول کاربران (Users)\*\*

```csharp

public class AppUser : IdentityUser<Guid>

{

public string FirstName { get; set; }

public string LastName { get; set; }

public DateTime CreatedAt { get; set; }

public DateTime? LastLoginDate { get; set; }

public bool IsVerified { get; set; }

public bool IsActive { get; set; }

public int MaxAllowedDevices { get; set; }

public int ActiveDeviceCount { get; set; }

public string PhoneNumber { get; set; }

public bool IsDeleted { get; set; }

public DateTime? UpdatedAt { get; set; }

// Relationships

public ICollection<DeviceEntity> Devices { get; set; }

public ICollection<RefreshTokenEntity> RefreshTokens { get; set; }

public ICollection<SubscriptionEntity> Subscriptions { get; set; }

}

```

**جدول انواع دستگاه‌ها (DeviceTypes)**

این جدول برای مدیریت انواع مختلف دستگاه‌ها طراحی شده است.

public class DeviceTypeEntity

{

public Guid Id { get; set; } // شناسه یکتا

public string Name { get; set; } // نام نوع دستگاه (مانند GPS Tracker, Smart Watch)

public string Description { get; set; } // توضیحات

public bool IsActive { get; set; } // وضعیت فعال بودن

public bool IsDeleted { get; set; } // حذف منطقی

public DateTime CreatedAt { get; set; } // تاریخ ایجاد

public DateTime? UpdatedAt { get; set; } // زمان آخرین به‌روزرسانی

// Relationships

public ICollection<DeviceEntity> Devices { get; set; } // ارتباط با دستگاه‌ها

}

---

### \*\*2. جدول تغییرات کاربر (UserChanges)\*\*

```csharp

public class UserChangeEntity

{

public Guid Id { get; set; }

public Guid UserId { get; set; }

public string FieldChanged { get; set; }

public string OldValue { get; set; }

public string NewValue { get; set; }

public string ChangeReason { get; set; }

public DateTime ChangedAt { get; set; }

}

```

---

### \*\*3. جدول نقش‌ها (Roles)\*\*

```csharp public class PermissionEntity

{

public Guid Id { get; set; } // شناسه یکتا

public string Name { get; set; } // نام مجوز

public string Description { get; set; } // توضیحات مجوز

public bool IsDeleted { get; set; } // حذف منطقی

public DateTime? UpdatedAt { get; set; } // زمان آخرین به‌روزرسانی

// Relationships

public ICollection<RolePermissionEntity> RolePermissions { get; set; } // ارتباط با جدول واسط

}```

**3. جدول واسط نقش‌ها و مجوزها (RolePermissions)**

این جدول برای پیاده‌سازی رابطه Many-to-Many بین **RoleEntity** و **PermissionEntity** استفاده می‌شود.

public class RolePermissionEntity

{

public Guid RoleId { get; set; } // شناسه نقش

public RoleEntity Role { get; set; } // رابطه با نقش

public Guid PermissionId { get; set; } // شناسه مجوز

public PermissionEntity Permission { get; set; } // رابطه با مجوز

public bool IsDeleted { get; set; } // حذف منطقی

public DateTime? UpdatedAt { get; set; } // زمان آخرین به‌روزرسانی

}

---

### \*\*4. جدول مجوزها (Permissions)\*\*

```csharp

public class PermissionEntity

{

public Guid Id { get; set; } // شناسه یکتا

public string Name { get; set; } // نام مجوز

public string Description { get; set; } // توضیحات مجوز

public bool IsDeleted { get; set; } // حذف منطقی

public DateTime? UpdatedAt { get; set; } // زمان آخرین به‌روزرسانی

// Relationships

public ICollection<RolePermissionEntity> RolePermissions { get; set; } // ارتباط با جدول واسط

}

### \*\*5. جدول دستگاه‌ها (Devices)\*\*

```csharp

public class DeviceEntity

{

public Guid Id { get; set; }

public Guid UserId { get; set; }

public AppUser User { get; set; }

public string SerialNumber { get; set; }

public string Name { get; set; }

public DeviceStatus Status { get; set; }

public DateTime CreatedAt { get; set; }

public DateTime? LastStatusChangeAt { get; set; }

public DeviceStatus PreviousStatus { get; set; }

public DateTime? LastSeen { get; set; }

public int? BatteryLevel { get; set; }

public string FirmwareVersion { get; set; }

public bool IsDeleted { get; set; }

public DateTime? UpdatedAt { get; set; }

// Relationships

public ICollection<LocationEntity> Locations { get; set; }

public ICollection<CommandEntity> Commands { get; set; }

public Guid SubscriptionId { get; set; }

public SubscriptionEntity Subscription { get; set; }

}

```

---

### \*\*6. جدول موقعیت‌ها (Locations)\*\*

```csharp

public class LocationEntity

{

public Guid Id { get; set; }

public Guid DeviceId { get; set; }

public DeviceEntity Device { get; set; }

public double Latitude { get; set; }

public double Longitude { get; set; }

public double? Altitude { get; set; }

public double? Speed { get; set; }

public double? Direction { get; set; }

public DateTime Timestamp { get; set; }

public bool IsDeleted { get; set; }

public DateTime? UpdatedAt { get; set; }

}

```

---

### \*\*7. جدول دستورات (Commands)\*\*

```csharp

public class CommandEntity

{

public Guid Id { get; set; }

public Guid DeviceId { get; set; }

public DeviceEntity Device { get; set; }

public CommandType Type { get; set; }

public CommandStatus Status { get; set; }

public DateTime CreatedAt { get; set; }

public DateTime? ExecutedAt { get; set; }

public string ErrorCode { get; set; }

public string ErrorMessage { get; set; }

public bool IsDeleted { get; set; }

public DateTime? UpdatedAt { get; set; }

}

```

---

### \*\*8. جدول اشتراک‌ها (Subscriptions)\*\*

```csharp

public class SubscriptionEntity

{

public Guid Id { get; set; }

public Guid UserId { get; set; }

public AppUser User { get; set; }

public SubscriptionType Type { get; set; }

public DateTime StartDate { get; set; }

public DateTime EndDate { get; set; }

public decimal Amount { get; set; }

public PaymentStatus Status { get; set; }

public string InvoiceId { get; set; }

public string PaymentMethod { get; set; }

public bool AutoRenewal { get; set; }

public bool IsDeleted { get; set; }

public DateTime? UpdatedAt { get; set; }

// Relationships

public ICollection<DeviceEntity> Devices { get; set; }

}

```

---

### \*\*9. جدول تاریخچه اشتراک‌ها (SubscriptionHistory)\*\*

```csharp

public class SubscriptionHistoryEntity

{

public Guid Id { get; set; }

public Guid SubscriptionId { get; set; }

public SubscriptionType PreviousType { get; set; }

public DateTime PreviousStartDate { get; set; }

public DateTime PreviousEndDate { get; set; }

public decimal PreviousAmount { get; set; }

public DateTime ChangedAt { get; set; }

public bool IsDeleted { get; set; }

}

```

---

### \*\*10. جدول اعلان‌ها (Notifications)\*\*

```csharp

public class NotificationEntity

{

public Guid Id { get; set; }

public Guid UserId { get; set; }

public AppUser User { get; set; }

public string Message { get; set; }

public NotificationType Type { get; set; }

public NotificationStatus Status { get; set; }

public bool IsRead { get; set; }

public DateTime CreatedAt { get; set; }

public bool IsDeleted { get; set; }

public DateTime? UpdatedAt { get; set; }

}

```

---

### \*\*11. جدول پیام‌های پشتیبانی (SupportMessages)\*\*

```csharp

public class SupportMessageEntity

{

public Guid Id { get; set; }

public Guid UserId { get; set; }

public AppUser User { get; set; }

public string Subject { get; set; }

public string Message { get; set; }

public DateTime CreatedAt { get; set; }

public bool IsReadByAdmin { get; set; }

public bool IsDeleted { get; set; }

}

```

---

### \*\*12. جدول توکن‌های Refresh (RefreshTokens)\*\*

```csharp

public class RefreshTokenEntity

{

public int Id { get; set; }

public Guid UserId { get; set; }

public AppUser User { get; set; }

public string Token { get; set; }

public DateTime ExpiresAt { get; set; }

public bool IsRevoked { get; set; }

public DateTime CreatedAt { get; set; }

public bool IsDeleted { get; set; }

public DateTime? UpdatedAt { get; set; }

}

```

---

### \*\*13. جدول لاگ‌ها (RequestLogs)\*\*

```csharp

public class RequestLog

{

public int Id { get; set; }

public string RequestMethod { get; set; }

public string RequestPath { get; set; }

public int StatusCode { get; set; }

public DateTime Timestamp { get; set; }

public string UserId { get; set; }

public string IpAddress { get; set; }

public string UserAgent { get; set; }

public string ErrorMessage { get; set; }

public bool IsDeleted { get; set; }

}

```

---

### \*\*14. جدول آلارم‌ها (Alarms)\*\*

```csharp

public class AlarmEntity

{

public Guid Id { get; set; }

public Guid DeviceId { get; set; }

public DeviceEntity Device { get; set; }

public string Message { get; set; }

public DateTime TriggeredAt { get; set; }

public bool Acknowledged { get; set; }

public Guid? AcknowledgedByUserId { get; set; }

public bool IsDeleted { get; set; }

public DateTime? UpdatedAt { get; set; }

}

```

---

### \*\*15. جدول تنظیمات سیستم (SystemSettings)\*\*

```csharp

public class SystemSettingEntity

{

public Guid Id { get; set; }

public string Key { get; set; }

public string Value { get; set; }

public string Description { get; set; }

public DateTime CreatedAt { get; set; }

public DateTime? UpdatedAt { get; set; }

}

```

---

### \*\*16. جدول سلامت سیستم (SystemHealthStatus)\*\*

```csharp

public class SystemHealthStatus

{

public int Id { get; set; }

public double CpuUsage { get; set; }

public double MemoryUsage { get; set; }

public double DiskUsage { get; set; }

public string NetworkStatus { get; set; }

public DateTime LastChecked { get; set; }

public bool IsHealthy { get; set; }

}

public class DeviceEntity

{

public Guid Id { get; set; } // شناسه یکتا دستگاه

public Guid UserId { get; set; } // شناسه کاربر مالک دستگاه

public AppUser User { get; set; } // رابطه با کاربر

public Guid DeviceTypeId { get; set; } // شناسه نوع دستگاه

public DeviceTypeEntity DeviceType { get; set; } // رابطه با انواع دستگاه

public string SerialNumber { get; set; } // شماره سریال دستگاه

public string Name { get; set; } // نام دستگاه

public DeviceStatus Status { get; set; } // وضعیت دستگاه

public DateTime CreatedAt { get; set; } // تاریخ ثبت

public DateTime? LastSeen { get; set; } // آخرین زمان مشاهده دستگاه

public int? BatteryLevel { get; set; } // سطح شارژ دستگاه (اختیاری)

public string FirmwareVersion { get; set; } // نسخه فرم‌ور دستگاه

public bool IsDeleted { get; set; } // حذف منطقی

public DateTime? UpdatedAt { get; set; } // زمان آخرین به‌روزرسانی

// Relationships

public ICollection<LocationEntity> Locations { get; set; } // ارتباط با موقعیت‌ها

public ICollection<CommandEntity> Commands { get; set; } // ارتباط با دستورات

}

|  |
| --- |
| AuditEntity |

|  |
| --- |
| لاگ تغییرات سیستمی دقیق‌تر (برای دستگاه/اشتراک/نقش‌ها و...) |

**ساختار کامل پروژه مدیریت دستگاه‌های GPS با ظرفیت پشتیبانی از 10 میلیون کاربر و بیشتر**

در ادامه، ساختار کامل پوشه‌بندی پروژه به همراه تمام ماژول‌ها، فایل‌ها، توضیحات جامع، وابستگی‌ها، و نیازمندی‌های خاص ارائه شده است. این ساختار شامل تمام ویژگی‌های درخواستی مانند MQTT ، SignalR ، RateLimiter.Redis ، IdentityServer/OpenIddict ، ElasticSearch + Kibana ، Circuit Breaker و Retry با Polly ، Prometheus + Grafana ، Swagger ، و CQRS + Event Sourcing است.

/src

│

├── ApiGateway

│ ├── Controllers

│ │ └── GatewayController.cs

│ ├── Middleware

│ │ ├── AuthenticationMiddleware.cs

│ │ ├── RateLimitingMiddleware.cs

│ │ └── DeviceLimitMiddleware.cs

│ ├── Extensions

│ │ └── ServiceCollectionExtensions.cs

│ ├── Configurations

│ │ ├── OcelotConfiguration.json

│ │ └── RateLimiterConfiguration.cs

│ ├── appsettings.json

│ └── Program.cs

│

├── LocationTrackingService

│ ├── Controllers

│ │ └── LocationsController.cs

│ ├── Services

│ │ ├── LocationService.cs

│ │ └── ILocationService.cs

│ ├── Repositories

│ │ ├── LocationRepository.cs

│ │ └── ILocationRepository.cs

│ ├── Models

│ │ ├── LocationDto.cs

│ │ └── LocationEntity.cs

│ ├── Messaging

│ │ └── KafkaLocationPublisher.cs

│ ├── BackgroundServices

│ │ └── MqttBackgroundService.cs

│ ├── Persistence

│ │ ├── LocationDbContext.cs

│ │ └── EntityConfigurations

│ │ └── LocationConfiguration.cs

│ ├── Logging

│ │ └── RequestLoggingMiddleware.cs

│ ├── Caching

│ │ └── RedisCacheService.cs

│ ├── RealTimeCommunication

│ │ └── SignalRHub.cs

│ ├── appsettings.json

│ └── Program.cs

│

├── DeviceManagementService

│ ├── Controllers

│ │ └── DevicesController.cs

│ ├── Services

│ │ ├── DeviceService.cs

│ │ └── IDeviceService.cs

│ ├── Repositories

│ │ ├── DeviceRepository.cs

│ │ └── IDeviceRepository.cs

│ ├── Models

│ │ ├── DeviceDto.cs

│ │ └── DeviceEntity.cs

│ ├── Enums

│ │ └── DeviceStatus.cs

│ ├── ValueObjects

│ │ └── DeviceId.cs

│ ├── Persistence

│ │ ├── DeviceDbContext.cs

│ │ └── EntityConfigurations

│ │ └── DeviceConfiguration.cs

│ ├── RealTimeCommunication

│ │ └── MqttIntegration.cs

│ ├── appsettings.json

│ └── Program.cs

│

├── CommandService

│ ├── Controllers

│ │ └── CommandsController.cs

│ ├── Services

│ │ ├── CommandService.cs

│ │ └── ICommandService.cs

│ ├── Repositories

│ │ ├── CommandRepository.cs

│ │ └── ICommandRepository.cs

│ ├── Models

│ │ ├── CommandDto.cs

│ │ └── CommandEntity.cs

│ ├── Persistence

│ │ ├── CommandDbContext.cs

│ │ └── EntityConfigurations

│ │ └── CommandConfiguration.cs

│ ├── Messaging

│ │ └── MqttCommandPublisher.cs

│ ├── RealTimeCommunication

│ │ └── SignalRCommandHub.cs

│ ├── appsettings.json

│ └── Program.cs

│

├── NotificationService

│ ├── Controllers

│ │ └── NotificationsController.cs

│ ├── Services

│ │ ├── NotificationService.cs

│ │ └── INotificationService.cs

│ ├── Repositories

│ │ ├── NotificationRepository.cs

│ │ └── INotificationRepository.cs

│ ├── Models

│ │ ├── NotificationDto.cs

│ │ └── NotificationEntity.cs

│ ├── Persistence

│ │ ├── NotificationDbContext.cs

│ │ └── EntityConfigurations

│ │ └── NotificationConfiguration.cs

│ ├── RealTimeCommunication

│ │ └── SignalRNotificationHub.cs

│ ├── appsettings.json

│ └── Program.cs

│

├── AuthService

│ ├── Controllers

│ │ └── AuthController.cs

│ ├── Services

│ │ ├── AuthService.cs

│ │ └── IAuthService.cs

│ ├── Models

│ │ ├── UserDto.cs

│ │ └── UserEntity.cs

│ ├── Enums

│ │ └── UserRole.cs

│ ├── Persistence

│ │ ├── AuthDbContext.cs

│ │ └── EntityConfigurations

│ │ └── UserConfiguration.cs

│ ├── Token

│ │ └── JwtTokenGenerator.cs

│ ├── OpenIddict

│ │ └── OpenIddictConfiguration.cs

│ ├── appsettings.json

│ └── Program.cs

│

├── SubscriptionService

│ ├── Controllers

│ │ └── SubscriptionsController.cs

│ ├── Services

│ │ ├── SubscriptionService.cs

│ │ └── ISubscriptionService.cs

│ ├── Repositories

│ │ ├── SubscriptionRepository.cs

│ │ └── ISubscriptionRepository.cs

│ ├── Models

│ │ ├── SubscriptionDto.cs

│ │ └── SubscriptionEntity.cs

│ ├── Persistence

│ │ ├── SubscriptionDbContext.cs

│ │ └── EntityConfigurations

│ │ └── SubscriptionConfiguration.cs

│ ├── appsettings.json

│ └── Program.cs

│

├── BillingService

│ ├── Controllers

│ │ └── BillingController.cs

│ ├── Services

│ │ ├── BillingService.cs

│ │ └── IBillingService.cs

│ ├── Repositories

│ │ ├── BillingRepository.cs

│ │ └── IBillingRepository.cs

│ ├── Models

│ │ ├── BillingDto.cs

│ │ └── BillingEntity.cs

│ ├── Persistence

│ │ ├── BillingDbContext.cs

│ │ └── EntityConfigurations

│ │ └── BillingConfiguration.cs

│ ├── appsettings.json

│ └── Program.cs

│

├── EventBus

│ ├── IEventPublisher.cs

│ ├── IEventSubscriber.cs

│ ├── KafkaEventBus.cs

│ ├── Outbox

│ │ ├── OutboxMessageEntity.cs

│ │ └── OutboxPublisherService.cs

│ ├── appsettings.json

│ └── Program.cs

│

├── EventStoreService

│ ├── Controllers

│ │ └── EventsController.cs

│ ├── Services

│ │ ├── EventStoreService.cs

│ │ └── IEventStoreService.cs

│ ├── Repositories

│ │ ├── EventRepository.cs

│ │ └── IEventRepository.cs

│ ├── Models

│ │ ├── EventDto.cs

│ │ └── EventEntity.cs

│ ├── Persistence

│ │ ├── EventDbContext.cs

│ │ └── EntityConfigurations

│ │ └── EventConfiguration.cs

│ ├── Streaming

│ │ └── StreamProcessor.cs

│ ├── appsettings.json

│ └── Program.cs

│

├── SystemMonitoringService

│ ├── Controllers

│ │ └── HealthCheckController.cs

│ ├── Services

│ │ ├── MonitoringService.cs

│ │ └── IMonitoringService.cs

│ ├── Models

│ │ └── HealthCheckResult.cs

│ ├── Persistence

│ │ ├── MonitoringDbContext.cs

│ │ └── EntityConfigurations

│ │ └── MonitoringConfiguration.cs

│ ├── PrometheusMetrics

│ │ └── MetricsCollector.cs

│ ├── appsettings.json

│ └── Program.cs

│

├── JobSchedulerService

│ ├── QuartzJobService.cs

│ ├── IScheduledTask.cs

│ ├── Jobs

│ │ ├── RenewSubscriptionJob.cs

│ │ └── SendNotificationJob.cs

│ ├── appsettings.json

│ └── Program.cs

│

├── SharedKernel

│ ├── Exceptions

│ │ └── CustomException.cs

│ ├── Interfaces

│ │ └── IRepository.cs

│ ├── Extensions

│ │ └── ServiceCollectionExtensions.cs

│ ├── Constants

│ │ └── AppConstants.cs

│ ├── Audit

│ │ └── AuditLogService.cs

│ ├── appsettings.json

│ └── Program.cs

│

├── Infrastructure

│ ├── Persistence

│ │ └── BaseDbContext.cs

│ ├── Messaging

│ │ ├── MqttConfiguration.cs

│ │ └── SignalRConfiguration.cs

│ ├── Logging

│ │ └── SerilogLogger.cs

│ ├── Caching

│ │ └── RedisConfiguration.cs

│ ├── Security

│ │ └── EncryptionHelper.cs

│ ├── HealthChecks

│ │ └── HealthCheckExtensions.cs

│ ├── appsettings.json

│ └── Program.cs

│

└── Tests

├── UnitTests

│ ├── LocationServiceTests.cs

│ ├── DeviceServiceTests.cs

│ ├── CommandServiceTests.cs

│ ├── NotificationServiceTests.cs

│ ├── AuthServiceTests.cs

│ └── BillingServiceTests.cs

└── IntegrationTests

├── LocationTrackingIntegrationTests.cs

├── DeviceManagementIntegrationTests.cs

├── CommandIntegrationTests.cs

└── NotificationIntegrationTests.cs

### \*\*توضیح کامل ماژول ApiGateway\*\*

ماژول \*\*ApiGateway\*\* یکی از اجزای کلیدی در معماری میکروسرویس‌ها است و نقش آن به‌عنوان \*\*نقطه ورودی واحد\*\* برای تمام API‌هاست. این ماژول درخواست‌ها را دریافت کرده و آن‌ها را به میکروسرویس‌های مختلف توزیع می‌کند. در ادامه، تمام جنبه‌های این ماژول به‌صورت کامل توضیح داده شده است.

---

## \*\*1. هدف ماژول ApiGateway\*\*

- \*\*هدف اصلی\*\*: مدیریت توزیع درخواست‌ها به میکروسرویس‌های مختلف.

- \*\*کاربرد\*\*:

- توزیع درخواست‌ها به میکروسرویس‌های مختلف (Routing).

- مدیریت محدودیت‌های نرخ (Rate Limiting).

- اعتبارسنجی توکن‌ها (Authentication).

- لاگ‌گیری درخواست‌ها (Logging).

- مدیریت خطاهای سراسری (Global Error Handling).

---

## \*\*2. ساختار پوشه‌بندی ماژول ApiGateway\*\*

```plaintext

/ApiGateway

│

├── Controllers

│ └── GatewayController.cs

│

├── Middleware

│ ├── AuthenticationMiddleware.cs

│ ├── RateLimitingMiddleware.cs

│ └── GlobalExceptionMiddleware.cs

│

├── Configurations

│ ├── OcelotConfiguration.json

│ └── RateLimiterConfiguration.cs

│

├── Extensions

│ └── ServiceCollectionExtensions.cs

│

├── appsettings.json

└── Program.cs

```

---

3. توضیحات جامع برای هر بخش\*\*

3.1. Controllers\*\*

- \*\*هدف\*\*: مدیریت درخواست‌های HTTP و ارائه API‌های عمومی.

- \*\*فایل‌ها\*\*:

- \*\*GatewayController.cs\*\*:

- کنترلر اصلی برای مدیریت عملیات عمومی API Gateway.

- مثال: بررسی وضعیت سلامت سیستم (`GET /api/gateway/status`).

```csharp

[ApiController]

[Route("api/gateway")]

public class GatewayController : ControllerBase

{

[HttpGet("status")]

public IActionResult GetStatus()

{

return Ok(new { Message = "API Gateway is running." });

}

}

```

---

3.2. Middleware\*\*

- \*\*هدف\*\*: مدیریت درخواست‌ها و پاسخ‌ها در لایه میانی.

- \*\*فایل‌ها\*\*:

- \*\*AuthenticationMiddleware.cs\*\*: اعتبارسنجی توکن JWT.

- \*\*RateLimitingMiddleware.cs\*\*: محدود کردن تعداد درخواست‌ها از هر IP.

- \*\*GlobalExceptionMiddleware.cs\*\*: مدیریت خطاهای ناخواسته.

مثال: RateLimitingMiddleware.cs\*\*

```csharp

public class RateLimitingMiddleware

{

private readonly RequestDelegate \_next;

private readonly Dictionary<string, int> \_requestCounts = new();

private const int MaxRequestsPerMinute = 100;

public RateLimitingMiddleware(RequestDelegate next)

{

\_next = next;

}

public async Task InvokeAsync(HttpContext context)

{

var ipAddress = context.Connection.RemoteIpAddress?.ToString();

if (!\_requestCounts.ContainsKey(ipAddress))

{

\_requestCounts[ipAddress] = 0;

}

\_requestCounts[ipAddress]++;

if (\_requestCounts[ipAddress] > MaxRequestsPerMinute)

{

context.Response.StatusCode = StatusCodes.Status429TooManyRequests;

await context.Response.WriteAsync("Too many requests. Please try again later.");

return;

}

await \_next(context);

}

}

```

---

3.3. Configurations\*\*

- \*\*هدف\*\*: تنظیمات مسیریابی و توزیع درخواست‌ها.

- \*\*فایل‌ها\*\*:

- \*\*OcelotConfiguration.json\*\*: تنظیمات Routing برای توزیع درخواست‌ها به میکروسرویس‌ها.

```json

{

"Routes": [

{

"DownstreamPathTemplate": "/api/locations/{everything}",

"DownstreamScheme": "http",

"DownstreamHostAndPorts": [

{ "Host": "localhost", "Port": 5001 }

],

"UpstreamPathTemplate": "/locations/{everything}",

"UpstreamHttpMethod": ["GET", "POST", "PUT", "DELETE"]

},

{

"DownstreamPathTemplate": "/api/devices/{everything}",

"DownstreamScheme": "http",

"DownstreamHostAndPorts": [

{ "Host": "localhost", "Port": 5002 }

],

"UpstreamPathTemplate": "/devices/{everything}",

"UpstreamHttpMethod": ["GET", "POST", "PUT", "DELETE"]

}

],

"GlobalConfiguration": {

"BaseUrl": "http://localhost:5000"

}

}

```

---

3.4. Extensions\*\*

- \*\*هدف\*\*: تنظیمات Dependency Injection و سرویس‌های جانبی.

- \*\*فایل‌ها\*\*:

- \*\*ServiceCollectionExtensions.cs\*\*: ثبت سرویس‌ها و میان‌افزارها.

```csharp

public static class ServiceCollectionExtensions

{

public static IServiceCollection AddCustomServices(this IServiceCollection services)

{

services.AddTransient<RequestLoggingMiddleware>();

services.AddTransient<AuthenticationMiddleware>();

services.AddTransient<RateLimitingMiddleware>();

return services;

}

}

```

---

3.5. appsettings.json\*\*

- \*\*هدف\*\*: ذخیره تنظیمات سراسری.

- \*\*مثال\*\*:

```json

{

"ConnectionStrings": {

"DefaultConnection": "Server=localhost;Database=ApiGatewayDb;User Id=sa;Password=YourPassword;"

},

"JwtSettings": {

"SecretKey": "YourSecretKeyForJwtToken",

"Issuer": "YourIssuer",

"Audience": "YourAudience"

}

}

3.6. Program.cs\*\*

- \*\*هدف\*\*: راه‌اندازی برنامه و تنظیم خط لول درخواست‌ها.

```csharp

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

// Add custom services and middlewares

builder.Services.AddCustomServices();

builder.Services.AddOcelot(builder.Configuration);

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseMiddleware<RequestLoggingMiddleware>();

app.UseMiddleware<AuthenticationMiddleware>();

app.UseMiddleware<RateLimitingMiddleware>();

await app.UseOcelot();

app.MapControllers();

app.Run();

```

---

4. ویژگی‌ها و قابلیت‌ها\*\*

4.1. Rate Limiting\*\*

- \*\*هدف\*\*: محدود کردن تعداد درخواست‌ها از هر IP در دقیقه.

- \*\*پیاده‌سازی\*\*: استفاده از \*\*Redis\*\* یا \*\*MemoryCache\*\* برای ذخیره‌سازی تعداد درخواست‌ها.

```csharp

services.AddStackExchangeRedisCache(options =>

{

options.Configuration = "redis:6379";

});

```

---

4.2. Global Exception Handling\*\*

- \*\*هدف\*\*: مدیریت خطاهای ناخواسته و ارائه پاسخ‌های استاندارد.

- \*\*پیاده‌سازی\*\*: استفاده از Middleware یا `UseExceptionHandler`.

```csharp

app.UseExceptionHandler(errorApp =>

{

errorApp.Run(async context =>

{

context.Response.StatusCode = 500;

context.Response.ContentType = "application/json";

await context.Response.WriteAsync(JsonSerializer.Serialize(new

{

error = "Internal Server Error"

}));

});

});

```

---

4.3. Circuit Breaker\*\*

- \*\*هدف\*\*: جلوگیری از overload میکروسرویس‌ها با استفاده از \*\*Polly\*\*.

پیاده‌سازی\*\*: تنظیمات QoS در فایل `OcelotConfiguration.json`.

```json

"QoSOptions": {

"ExceptionsAllowedBeforeBreaking": 3,

"DurationOfBreak": 10000,

"TimeoutValue": 5000

}

```

---

4.4. Health Checks\*\*

- \*\*هدف\*\*: بررسی وضعیت سلامت میکروسرویس‌ها.

- \*\*پیاده‌سازی\*\*: استفاده از `AspNetCore.HealthChecks`.

```csharp

builder.Services.AddHealthChecks()

.AddCheck("self", () => HealthCheckResult.Healthy())

.AddUrlGroup(new Uri("http://identityserver:5000/health"), name: "IdentityServer");

4.5. IP Whitelisting/Blacklisting\*\*

- \*\*هدف\*\*: محدود کردن دسترسی بر اساس IP.

- \*\*پیاده‌سازی\*\*: استفاده از Middleware.

```csharp

public class IpWhitelistMiddleware

{

private readonly HashSet<string> \_allowedIps;

public IpWhitelistMiddleware(RequestDelegate next, IConfiguration configuration)

{

\_allowedIps = new HashSet<string>(configuration.GetSection("AllowedIps").Get<string[]>());

}

public async Task InvokeAsync(HttpContext context)

{

var ipAddress = context.Connection.RemoteIpAddress?.ToString();

if (!\_allowedIps.Contains(ipAddress))

{

context.Response.StatusCode = StatusCodes.Status403Forbidden;

await context.Response.WriteAsync("Access Denied");

return;

}

await \_next(context);

}

}

4.6. JWT Token Validation\*\*

- \*\*هدف\*\*: اعتبارسنجی توکن‌ها با استفاده از \*\*JwtBearerHandler\*\*.

```csharp

builder.Services.AddAuthentication("Bearer")

.AddJwtBearer("Bearer", options =>

{

options.Authority = "https://identityserver.local";

options.Audience = "gateway\_api";

5. نتیجه‌گیری

ماژول \*\*ApiGateway\*\* به‌عنوان نقطه ورودی سیستم، نقش حیاتی در مدیریت درخواست‌ها و توزیع آن‌ها به میکروسرویس‌های مختلف دارد. این ماژول با استفاده از ابزارهایی مانند \*\*Ocelot\*\*، \*\*Redis\*\*، و \*\*JWT\*\*، قابلیت‌هایی مانند محدودیت نرخ، اعتبارسنجی، و لاگ‌گیری را ارائه می‌دهد.

### \*\*توضیح کامل ماژول LocationTrackingService\*\*

ماژول \*\*LocationTrackingService\*\* یکی از اجزای کلیدی در سیستم مدیریت دستگاه‌های GPS است و نقش آن دریافت، پردازش، ذخیره‌سازی، و ارسال داده‌های مرتبط با موقعیت‌ها، وضعیت دستگاه‌ها، و دستورات به دستگاه‌ها است. این ماژول با استفاده از فناوری‌هایی مانند \*\*SignalR\*\*، \*\*Redis Cache\*\*، \*\*Kafka\*\*، و \*\*MQTT\*\* طراحی شده است تا عملکرد بلادرنگ و مقیاس‌پذیری را تضمین کند. در ادامه، تمام جنبه‌های این ماژول به‌صورت دقیق و کامل توضیح داده شده است.

---

## \*\*1. هدف ماژول LocationTrackingService\*\*

- \*\*هدف اصلی\*\*: مدیریت داده‌های مرتبط با موقعیت GPS، وضعیت دستگاه‌ها، و دستورات.

- \*\*کاربرد\*\*:

- دریافت داده‌های زنده (مانند موقعیت و وضعیت) از دستگاه‌ها از طریق پروتکل‌هایی مانند \*\*MQTT\*\*.

- پردازش و ذخیره‌سازی داده‌های موقعیت، وضعیت دستگاه، و دستورات در دیتابیس‌های زمان‌محور (مانند \*\*InfluxDB\*\* یا \*\*TimescaleDB\*\*).

- ارسال داده‌های زنده به کلاینت‌ها (مثل وب‌اپلیکیشن یا موبایل) از طریق \*\*SignalR\*\* یا \*\*WebSocket\*\*.

- مدیریت دسترسی به داده‌ها برای کاربران و دستگاه‌های مختلف.

- ارسال دستورات به دستگاه‌ها از طریق صف‌های رویداد مانند \*\*Kafka\*\*.

---

## \*\*2. ساختار کلی ماژول\*\*

```plaintext

/LocationTrackingService

│

├── Controllers

│ └── LocationsController.cs

│

├── Services

│ ├── LocationService.cs

│ ├── DeviceStatusService.cs

│ ├── CommandService.cs

│

├── Repositories

│ ├── LocationRepository.cs

│ ├── DeviceStatusRepository.cs

│ ├── CommandRepository.cs

│

├── Models

│ ├── LocationDto.cs

│ ├── DeviceStatusDto.cs

│ ├── CommandDto.cs

│ ├── LocationEntity.cs

│ ├── DeviceStatusEntity.cs

│ └── CommandEntity.cs

│

├── Infrastructure

│ ├── Persistence

│ │ └── LocationDbContext.cs

│ └── Messaging

│ ├── MqttBackgroundService.cs

│ └── KafkaEventProducer.cs

│

├── Extensions

│ └── AutofacModule.cs

│

└── Program.cs

```

---

## \*\*3. توضیحات جامع برای هر بخش\*\*

### \*\*3.1. Controllers (کنترلرها)\*\*

#### \*\*هدف:\*\*

کنترلرها واسط بین کلاینت‌ها (مثل وب‌اپلیکیشن یا موبایل) و سرویس‌های داخلی هستند. آن‌ها درخواست‌های HTTP را دریافت کرده و پاسخ مناسب ارسال می‌کنند.

#### \*\*مثال: LocationsController.cs\*\*

```csharp

[ApiController]

[Route("api/[controller]")]

public class LocationsController : ControllerBase

{

private readonly ILocationService \_locationService;

public LocationsController(ILocationService locationService)

{

\_locationService = locationService;

}

[HttpPost]

public async Task<IActionResult> TrackLocation([FromBody] LocationDto locationDto)

{

await \_locationService.TrackLocationAsync(locationDto);

return Ok(new { Message = "موقعیت با موفقیت ثبت شد." });

}

[HttpGet("{deviceId}")]

public async Task<IActionResult> GetLatestLocation(string deviceId)

{

var location = await \_locationService.GetLatestLocationAsync(deviceId);

if (location == null)

return NotFound(new { Message = "موقعیت یافت نشد." });

return Ok(location);

}

}

```

- \*\*POST /api/locations\*\*: برای ثبت موقعیت جدید.

- \*\*GET /api/locations/{deviceId}\*\*: برای دریافت آخرین موقعیت یک دستگاه.

---

### \*\*3.2. Services (سرویس‌ها)\*\*

#### \*\*هدف:\*\*

سرویس‌ها شامل منطق کسب‌وکار برنامه هستند. آن‌ها کارهایی مثل پردازش داده‌ها، ارتباط با مخازن، و ارسال دستورات به دستگاه‌ها را انجام می‌دهند.

#### \*\*مثال: LocationService.cs\*\*

```csharp

public class LocationService : ILocationService

{

private readonly ILocationRepository \_locationRepository;

public LocationService(ILocationRepository locationRepository)

{

\_locationRepository = locationRepository;

}

public async Task TrackLocationAsync(LocationDto locationDto)

{

var locationEntity = new LocationEntity

{

DeviceId = locationDto.DeviceId,

Latitude = locationDto.Latitude,

Longitude = locationDto.Longitude,

Timestamp = locationDto.Timestamp

};

await \_locationRepository.SaveLocationAsync(locationEntity);

}

public async Task<LocationDto> GetLatestLocationAsync(string deviceId)

{

var locationEntity = await \_locationRepository.GetLatestLocationAsync(deviceId);

if (locationEntity == null)

return null;

return new LocationDto

{

DeviceId = locationEntity.DeviceId,

Latitude = locationEntity.Latitude,

Longitude = locationEntity.Longitude,

Timestamp = locationEntity.Timestamp

};

}

}

```

---

### \*\*3.3. Repositories (مخازن)\*\*

#### \*\*هدف:\*\*

مخازن وظیفه مدیریت دسترسی به دیتابیس را دارند. آن‌ها شامل عملیات CRUD (Create, Read, Update, Delete) برای داده‌ها هستند.

#### \*\*مثال: LocationRepository.cs\*\*

```csharp

public class LocationRepository : ILocationRepository

{

private readonly LocationDbContext \_context;

public LocationRepository(LocationDbContext context)

{

\_context = context;

}

public async Task SaveLocationAsync(LocationEntity location)

{

\_context.Locations.Add(location);

await \_context.SaveChangesAsync();

}

public async Task<LocationEntity> GetLatestLocationAsync(string deviceId)

{

return await \_context.Locations

.Where(l => l.DeviceId == deviceId)

.OrderByDescending(l => l.Timestamp)

.FirstOrDefaultAsync();

}

}

```

---

### \*\*3.4. Models (مدل‌ها)\*\*

هدف:\*\*

مدل‌ها شامل داده‌هایی هستند که در سراسر برنامه استفاده می‌شوند. آن‌ها به دو دسته تقسیم می‌شوند:

- \*\*DTO (Data Transfer Object)\*\*: برای انتقال داده‌ها بین لایه‌ها.

- \*\*Entities\*\*: برای ذخیره‌سازی داده‌ها در دیتابیس.

مثال: LocationDto.cs و LocationEntity.cs\*\*

```csharp

// DTO برای انتقال داده‌ها

public class LocationDto

{

public string DeviceId { get; set; }

public double Latitude { get; set; }

public double Longitude { get; set; }

public DateTime Timestamp { get; set; }

}

// Entity برای ذخیره‌سازی در دیتابیس

public class LocationEntity

{

public int Id { get; set; }

public string DeviceId { get; set; }

public double Latitude { get; set; }

public double Longitude { get; set; }

public DateTime Timestamp { get; set; }

}

```

---

3.5. Infrastructure (زیرساخت)\*\*

هدف:\*\*

این بخش شامل ابزارهایی است که برای مدیریت ارتباطات، دیتابیس‌ها، و پیام‌رسانی استفاده می‌شوند.

مثال: MqttBackgroundService.cs\*\*

```csharp

public class MqttBackgroundService : BackgroundService

{

protected override async Task ExecuteAsync(CancellationToken stoppingToken)

{

while (!stoppingToken.IsCancellationRequested)

{

// دریافت داده‌های موقعیت از MQTT

var locationData = ReceiveLocationFromMqtt();

// پردازش و ذخیره داده‌ها

await ProcessLocationData(locationData);

await Task.Delay(5000, stoppingToken);

}

}

private LocationDto ReceiveLocationFromMqtt()

{

// دریافت داده‌ها از MQTT

return new LocationDto

{

DeviceId = "device123",

Latitude = 35.6895,

Longitude = 51.3890,

Timestamp = DateTime.UtcNow

};

}

private async Task ProcessLocationData(LocationDto locationData)

{

// ذخیره داده‌ها در دیتابیس

await \_locationService.TrackLocationAsync(locationData);

}

}

```

---

3.6. Extensions (افزونه‌ها)\*\*

هدف:\*\*

این بخش شامل تنظیمات Dependency Injection و سرویس‌های جانبی است.

#### \*\*مثال: AutofacModule.cs\*\*

```csharp

public class AutofacModule : Module

{

protected override void Load(ContainerBuilder builder)

{

builder.RegisterType<LocationDbContext>().AsSelf().InstancePerLifetimeScope();

builder.RegisterType<LocationRepository>().As<ILocationRepository>().InstancePerLifetimeScope();

builder.RegisterType<LocationService>().As<ILocationService>().InstancePerLifetimeScope();

builder.RegisterType<MqttBackgroundService>().AsSelf().SingleInstance();

}

}

```

---

3.7. Program.cs\*\*

هدف:\*\*

این فایل برنامه را راه‌اندازی می‌کند و تنظیمات اولیه را اعمال می‌کند.

```csharp

var builder = WebApplication.CreateBuilder(args);

// استفاده از Autofac برای DI

builder.Host.UseServiceProviderFactory(new AutofacServiceProviderFactory())

.ConfigureContainer<ContainerBuilder>(containerBuilder =>

{

containerBuilder.RegisterModule<AutofacModule>();

});

// افزودن سرویس‌های پیش‌فرض ASP.NET Core

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.MapControllers();

app.Run();

```

---

4. ویژگی‌ها و قابلیت‌ها\*\*

4.1. SignalR برای ارسال داده‌های زنده\*\*

- \*\*هدف:\*\* ارسال داده‌های موقعیت و وضعیت دستگاه به کلاینت‌ها به‌صورت بلادرنگ.

4.2. Redis Cache\*\*

- \*\*هدف:\*\* ذخیره موقت داده‌های پرتکرار مانند آخرین موقعیت دستگاه‌ها.

4.3. Time-Series Database\*\*

- \*\*هدف:\*\* ذخیره‌سازی داده‌های زمان‌محور (مانند موقعیت‌ها و وضعیت دستگاه‌ها).

4.4. Event Streaming با Kafka\*\*

- \*\*هدف:\*\* مدیریت دستورات و ارسال آن‌ها به دستگاه‌ها.

**4.4. Event Streaming با Kafka**

* **هدف:** استفاده از Kafka برای ارسال و دریافت رویدادها (مانند موقعیت‌ها، وضعیت‌ها، و دستورات) بین میکروسرویس‌ها. این ویژگی کمک می‌کند تا سیستم:
  + **مقیاس‌پذیر** و **ماژولار** باشد.
  + بتواند **داده‌ها را به صورت asynchronous** پردازش کند.
  + قابلیت **بازپخش (Replay)** رویدادها در صورت نیاز را داشته باشد.
  + امکان پیاده‌سازی **CQRS** یا **Event Sourcing** را فراهم کند.

بله، تمام مواردی که ارائه دادید به‌دقت در نظر گرفته شده‌اند و در توضیحات ماژول \*\*LocationTrackingService\*\* اعمال شده‌اند. برای اطمینان از پوشش کامل نیازها، در ادامه به بررسی دقیق‌تر مواردی که ذکر کردید می‌پردازم و نحوه پیاده‌سازی آن‌ها را توضیح می‌دهم:

1. امنیت و اعتبارسنجی پیام‌ها\*\*

1.1. اعتبارسنجی پیام‌ها با JWT یا HMAC\*\*

- \*\*هدف\*\*: اطمینان از صحت و امنیت داده‌های ارسالی از سمت دستگاه‌ها.

- \*\*پیاده‌سازی\*\*:

- استفاده از \*\*JWT\*\* برای اعتبارسنجی کاربران و دستگاه‌ها.

- استفاده از \*\*HMAC\*\* برای امضای پیام‌ها و بررسی صحت آن‌ها.

مثال: اعتبارسنجی با JWT\*\*

```csharp

public class JwtValidationMiddleware

{

private readonly RequestDelegate \_next;

public JwtValidationMiddleware(RequestDelegate next)

{

\_next = next;

}

public async Task InvokeAsync(HttpContext context, IJwtService jwtService)

{

var token = context.Request.Headers["Authorization"].ToString().Replace("Bearer ", "");

if (!jwtService.ValidateToken(token))

{

context.Response.StatusCode = StatusCodes.Status401Unauthorized;

await context.Response.WriteAsync("Unauthorized");

return;

}

await \_next(context);

}

}

```

1.2. بررسی صحت DeviceId\*\*

- \*\*هدف\*\*: اطمینان از اینکه دستگاه‌ها فعال و تأیید شده‌اند.

- \*\*پیاده‌سازی\*\*:

- اضافه کردن یک لیست سفید (Whitelist) برای `DeviceId`ها.

- بررسی وجود `DeviceId` در دیتابیس قبل از پردازش پیام.

مثال: بررسی DeviceId\*\*

```csharp

public class DeviceValidationService : IDeviceValidationService

{

private readonly IDeviceRepository \_deviceRepository;

public DeviceValidationService(IDeviceRepository deviceRepository)

{

\_deviceRepository = deviceRepository;

}

public async Task<bool> IsValidDevice(string deviceId)

{

var device = await \_deviceRepository.GetDeviceByIdAsync(deviceId);

return device != null && device.IsActive;

}

}

```

---

2. کاهش مصرف منابع\*\*

2.1. Rate Limiting\*\*

- \*\*هدف\*\*: جلوگیری از overload شدن سیستم هنگام ارسال بیش‌ازحد داده از سمت یک دستگاه.

- \*\*پیاده‌سازی\*\*:

- استفاده از Middleware برای محدود کردن تعداد درخواست‌ها.

مثال: Rate Limiting Middleware\*\*

```csharp

public class RateLimitingMiddleware

{

private readonly RequestDelegate \_next;

private static readonly ConcurrentDictionary<string, int> \_requestCounts = new();

public RateLimitingMiddleware(RequestDelegate next)

{

\_next = next;

}

public async Task InvokeAsync(HttpContext context)

{

var deviceId = context.Request.Headers["DeviceId"];

if (\_requestCounts.ContainsKey(deviceId) && \_requestCounts[deviceId] > 100)

{

context.Response.StatusCode = StatusCodes.Status429TooManyRequests;

await context.Response.WriteAsync("Rate limit exceeded.");

return;

}

\_requestCounts.AddOrUpdate(deviceId, 1, (key, oldValue) => oldValue + 1);

await \_next(context);

}

}

```

2.2. فشرده‌سازی پیام‌ها\*\*

- \*\*هدف\*\*: کاهش حجم داده‌ها در Kafka و MQTT.

- \*\*پیاده‌سازی\*\*:

- استفاده از الگوریتم‌های فشرده‌سازی مانند \*\*GZip\*\* یا \*\*Snappy\*\*.

مثال: فشرده‌سازی در Kafka\*\*

```csharp

var config = new ProducerConfig

{

BootstrapServers = "localhost:9092",

CompressionType = CompressionType.Snappy

};

```

---

3. بهبود کش Redis\*\*

3.1. Expiration Policy\*\*

- \*\*هدف\*\*: مدیریت منابع Redis با تنظیم زمان انقضا (TTL).

- \*\*پیاده‌سازی\*\*:

- اضافه کردن TTL برای داده‌های موقت.

مثال: تنظیم Expiration Policy\*\*

```csharp

public class LocationCacheService : ILocationCacheService

{

private readonly IDatabase \_redis;

public LocationCacheService(IConnectionMultiplexer redis)

{

\_redis = redis.GetDatabase();

}

public void CacheLocation(string deviceId, string locationData)

{

\_redis.StringSet($"location:{deviceId}", locationData, TimeSpan.FromMinutes(5));

}

}

```

3.2. الگوهای Cache Aside و Write-through\*\*

- \*\*Cache Aside\*\*:

- ابتدا داده‌ها از کش خوانده می‌شوند و در صورت عدم وجود، از دیتابیس خوانده می‌شوند.

- \*\*Write-through\*\*:

- داده‌ها همزمان در کش و دیتابیس ذخیره می‌شوند.

---

. تست و مانیتورینگ\*\*

4.1. مانیتورینگ با Prometheus و Grafana\*\*

- \*\*هدف\*\*: نظارت بر عملکرد سیستم.

- \*\*پیاده‌سازی\*\*:

- استفاده از Prometheus برای جمع‌آوری متریک‌ها.

- استفاده از Grafana برای نمایش داشبوردها.

مثال: ثبت متریک‌ها با Prometheus\*\*

```csharp

private static readonly Counter RequestCounter = Metrics.CreateCounter("total\_requests", "Total number of requests");

public async Task TrackLocationAsync(LocationDto locationDto)

{

RequestCounter.Inc();

// پردازش داده‌ها

}

```

4.2. تست‌های واحد و ادغام\*\*

- \*\*Unit Test\*\*:

- تست‌هایی برای بررسی عملکرد تک‌تک کلاس‌ها.

- \*\*Integration Test\*\*:

- تست‌هایی برای بررسی تعامل بین ماژول‌ها.

مثال: Unit Test برای LocationService\*\*

```csharp

public class LocationServiceTests

{

[Fact]

public async Task TrackLocationAsync\_ShouldSaveLocation()

{

// Arrange

var mockRepository = new Mock<ILocationRepository>();

var service = new LocationService(mockRepository.Object);

// Act

await service.TrackLocationAsync(new LocationDto { DeviceId = "123" });

// Assert

mockRepository.Verify(repo => repo.SaveLocationAsync(It.IsAny<LocationEntity>()), Times.Once);

}

}

```

---

5. بهبود BackgroundService\*\*

5.1. استفاده از Event-Based Callbacks\*\*

- \*\*هدف\*\*: حذف Delay غیرضروری و استفاده از رویدادها برای دریافت پیام‌ها.

- \*\*پیاده‌سازی\*\*:

- استفاده از Callback برای پردازش پیام‌های MQTT.

#### \*\*مثال: MqttBackgroundService به‌روش Event-Based\*\*

```csharp

protected override async Task ExecuteAsync(CancellationToken stoppingToken)

{

\_mqttClient.UseApplicationMessageReceivedHandler(async e =>

{

var message = Encoding.UTF8.GetString(e.ApplicationMessage.Payload);

await ProcessMessageAsync(message);

});

await \_mqttClient.SubscribeAsync("device/+/location", stoppingToken);

}

private async Task ProcessMessageAsync(string message)

{

// پردازش پیام

}

```

---

**6. توسعه آینده و نکات پیشنهادی**

* افزودن قابلیت **Geofencing** (محدوده‌ی مجاز مکانی برای دستگاه‌ها).
* طراحی سیستم **هشداردهی بلادرنگ** (مثلاً هنگام ورود به منطقه ممنوع یا توقف طولانی).
* پشتیبانی از چندین نوع پروتکل (مانند HTTP, TCP) در کنار MQTT.
* یکپارچه‌سازی با سیستم‌های تحلیلی (مانند Grafana یا Kibana برای مانیتورینگ داده‌ها).
* ذخیره‌سازی داده‌های تراکنشی مهم در پایگاه‌های داده SQL در کنار Time-Series DB برای گزارش‌های مدیریتی.

7. نتیجه‌گیری\*\*

با اعمال این موارد، ماژول \*\*LocationTrackingService\*\* به‌صورت قابل توجهی بهبود می‌یابد:

- \*\*امنیت\*\*: اعتبارسنجی پیام‌ها با JWT یا HMAC و بررسی صحت `DeviceId`.

- \*\*کاهش مصرف منابع\*\*: استفاده از Rate Limiting و فشرده‌سازی پیام‌ها.

- \*\*بهبود کش Redis\*\*: اضافه کردن Expiration Policy و استفاده از الگوهای Cache Aside/Write-through.

- \*\*تست و مانیتورینگ\*\*: اضافه کردن تست‌ها و مانیتورینگ با Prometheus و Grafana.

- \*\*بهبود BackgroundService\*\*: استفاده از رویدادها برای پردازش پیام‌ها.

ماژول \*\*LocationTrackingService\*\* یک ماژول کامل و مستقل است که می‌تواند به‌عنوان بخشی از یک سیستم ردیابی GPS استفاده شود. تمام بخش‌ها شامل کنترلرها، سرویس‌ها، مخازن، مدیریت دیتابیس، و ارسال دستورات از طریق Kafka به همراه توضیحات کامل نوشته شده‌اند

بله، متوجه شدم. اجازه دهید تمام موارد را به‌صورت کاملاً واضح و ساختارمند برای شما توضیح دهم. این بار تمام جزئیات را با دقت و بدون هیچ ابهامی بیان می‌کنم.

---

### \*\*توضیح کامل ماژول DeviceManagementService\*\*

ماژول \*\*DeviceManagementService\*\* یکی از اجزای کلیدی در سیستم مدیریت دستگاه‌های GPS است. این ماژول مسئولیت مدیریت دستگاه‌ها، شامل ثبت، به‌روزرسانی، حذف، و اختصاص دستگاه‌ها به کاربران را بر عهده دارد. همچنین، این ماژول قابلیت‌هایی مانند اعتبارسنجی شماره سریال دستگاه‌ها، مدیریت وضعیت دستگاه‌ها، و اتصال آن‌ها به حساب کاربری را فراهم می‌کند. در ادامه، تمام جنبه‌های این ماژول به‌صورت دقیق و ساختارمند توضیح داده شده است.

---

## \*\*1. هدف ماژول DeviceManagementService\*\*

- \*\*هدف اصلی\*\*: مدیریت دستگاه‌ها در سیستم.

- \*\*کاربرد\*\*:

- ثبت دستگاه‌های جدید.

- به‌روزرسانی اطلاعات دستگاه‌ها (مانند نام، شماره سریال، و وضعیت).

- حذف دستگاه‌ها.

- اختصاص دستگاه‌ها به کاربران.

- اعتبارسنجی شماره سریال دستگاه‌ها قبل از ثبت.

- مدیریت وضعیت دستگاه‌ها (فعال، غیرفعال، متصل، قطع شده).

---

## \*\*2. ساختار کلی ماژول\*\*

```plaintext

/DeviceManagementService

│

├── Controllers

│ └── DevicesController.cs

│

├── Services

│ ├── DeviceService.cs

│ └── IDeviceService.cs

│

├── Repositories

│ ├── DeviceRepository.cs

│ └── IDeviceRepository.cs

│

├── Models

│ ├── DeviceDto.cs

│ └── DeviceEntity.cs

│

├── Infrastructure

│ ├── Persistence

│ │ └── DeviceDbContext.cs

│ └── Messaging

│ └── MqttBackgroundService.cs

│

├── Extensions

│ └── ServiceCollectionExtensions.cs

│

└── Program.cs

```

---

## \*\*3. توضیحات جامع برای هر بخش\*\*

### \*\*3.1. Controllers (کنترلرها)\*\*

#### \*\*هدف:\*\*

کنترلرها واسط بین کلاینت‌ها (مثل وب‌اپلیکیشن یا موبایل) و سرویس‌های داخلی هستند. آن‌ها درخواست‌های HTTP را دریافت کرده و پاسخ مناسب ارسال می‌کنند.

#### \*\*مثال: DevicesController.cs\*\*

```csharp

[ApiController]

[Route("api/[controller]")]

public class DevicesController : ControllerBase

{

private readonly IDeviceService \_deviceService;

public DevicesController(IDeviceService deviceService)

{

\_deviceService = deviceService;

}

[HttpPost]

public async Task<IActionResult> RegisterDevice([FromBody] DeviceDto deviceDto)

{

var result = await \_deviceService.RegisterDeviceAsync(deviceDto);

return Ok(result);

}

[HttpGet("{deviceId}")]

public async Task<IActionResult> GetDevice(string deviceId)

{

var device = await \_deviceService.GetDeviceByIdAsync(deviceId);

if (device == null)

return NotFound(new { Message = "دستگاه یافت نشد." });

return Ok(device);

}

[HttpPut("{deviceId}")]

public async Task<IActionResult> UpdateDevice(string deviceId, [FromBody] DeviceDto deviceDto)

{

var result = await \_deviceService.UpdateDeviceAsync(deviceId, deviceDto);

return Ok(result);

}

[HttpDelete("{deviceId}")]

public async Task<IActionResult> DeleteDevice(string deviceId)

{

var result = await \_deviceService.DeleteDeviceAsync(deviceId);

return Ok(result);

}

}

```

---

### \*\*3.2. Services (سرویس‌ها)\*\*

#### \*\*هدف:\*\*

سرویس‌ها شامل منطق کسب‌وکار برنامه هستند. آن‌ها کارهایی مثل ثبت دستگاه‌ها، به‌روزرسانی اطلاعات، و اعتبارسنجی شماره سریال را انجام می‌دهند.

#### \*\*مثال: DeviceService.cs\*\*

```csharp

public class DeviceService : IDeviceService

{

private readonly IDeviceRepository \_deviceRepository;

public DeviceService(IDeviceRepository deviceRepository)

{

\_deviceRepository = deviceRepository;

}

public async Task<DeviceDto> RegisterDeviceAsync(DeviceDto deviceDto)

{

// اعتبارسنجی شماره سریال

if (!IsValidSerialNumber(deviceDto.SerialNumber))

throw new Exception("شماره سریال نامعتبر است.");

var deviceEntity = new DeviceEntity

{

DeviceId = Guid.NewGuid().ToString(),

Name = deviceDto.Name,

SerialNumber = deviceDto.SerialNumber,

Status = DeviceStatus.Inactive,

CreatedAt = DateTime.UtcNow

};

await \_deviceRepository.SaveDeviceAsync(deviceEntity);

return new DeviceDto

{

DeviceId = deviceEntity.DeviceId,

Name = deviceEntity.Name,

SerialNumber = deviceEntity.SerialNumber,

Status = deviceEntity.Status.ToString(),

CreatedAt = deviceEntity.CreatedAt

};

}

public async Task<DeviceDto> GetDeviceByIdAsync(string deviceId)

{

var deviceEntity = await \_deviceRepository.GetDeviceByIdAsync(deviceId);

if (deviceEntity == null)

return null;

return new DeviceDto

{

DeviceId = deviceEntity.DeviceId,

Name = deviceEntity.Name,

SerialNumber = deviceEntity.SerialNumber,

Status = deviceEntity.Status.ToString(),

CreatedAt = deviceEntity.CreatedAt

};

}

public async Task<DeviceDto> UpdateDeviceAsync(string deviceId, DeviceDto deviceDto)

{

var deviceEntity = await \_deviceRepository.GetDeviceByIdAsync(deviceId);

if (deviceEntity == null)

throw new Exception("دستگاه یافت نشد.");

deviceEntity.Name = deviceDto.Name;

deviceEntity.SerialNumber = deviceDto.SerialNumber;

deviceEntity.Status = Enum.Parse<DeviceStatus>(deviceDto.Status);

await \_deviceRepository.UpdateDeviceAsync(deviceEntity);

return new DeviceDto

{

DeviceId = deviceEntity.DeviceId,

Name = deviceEntity.Name,

SerialNumber = deviceEntity.SerialNumber,

Status = deviceEntity.Status.ToString(),

CreatedAt = deviceEntity.CreatedAt

};

}

public async Task<bool> DeleteDeviceAsync(string deviceId)

{

return await \_deviceRepository.DeleteDeviceAsync(deviceId);

}

private bool IsValidSerialNumber(string serialNumber)

{

// اعتبارسنجی شماره سریال

return !string.IsNullOrEmpty(serialNumber) && serialNumber.Length == 10;

}

}

```

---

### \*\*3.3. Repositories (مخازن)\*\*

#### \*\*هدف:\*\*

مخازن وظیفه مدیریت دسترسی به دیتابیس را دارند. آن‌ها شامل عملیات CRUD (Create, Read, Update, Delete) برای داده‌ها هستند.

#### \*\*مثال: DeviceRepository.cs\*\*

```csharp

public class DeviceRepository : IDeviceRepository

{

private readonly DeviceDbContext \_context;

public DeviceRepository(DeviceDbContext context)

{

\_context = context;

}

public async Task SaveDeviceAsync(DeviceEntity device)

{

\_context.Devices.Add(device);

await \_context.SaveChangesAsync();

}

public async Task<DeviceEntity> GetDeviceByIdAsync(string deviceId)

{

return await \_context.Devices.FirstOrDefaultAsync(d => d.DeviceId == deviceId);

}

public async Task UpdateDeviceAsync(DeviceEntity device)

{

\_context.Devices.Update(device);

await \_context.SaveChangesAsync();

}

public async Task<bool> DeleteDeviceAsync(string deviceId)

{

var device = await \_context.Devices.FirstOrDefaultAsync(d => d.DeviceId == deviceId);

if (device == null)

return false;

\_context.Devices.Remove(device);

await \_context.SaveChangesAsync();

return true;

}

}

```

---

### \*\*3.4. Models (مدل‌ها)\*\*

#### \*\*هدف:\*\*

مدل‌ها شامل داده‌هایی هستند که در سراسر برنامه استفاده می‌شوند. آن‌ها به دو دسته تقسیم می‌شوند:

- \*\*DTO (Data Transfer Object)\*\*: برای انتقال داده‌ها بین لایه‌ها.

- \*\*Entities\*\*: برای ذخیره‌سازی داده‌ها در دیتابیس.

#### \*\*مثال: DeviceDto.cs و DeviceEntity.cs\*\*

```csharp

// DTO برای انتقال داده‌ها

public class DeviceDto

{

public string DeviceId { get; set; }

public string Name { get; set; }

public string SerialNumber { get; set; }

public string Status { get; set; }

public DateTime CreatedAt { get; set; }

}

// Entity برای ذخیره‌سازی در دیتابیس

public class DeviceEntity

{

public string DeviceId { get; set; }

public string Name { get; set; }

public string SerialNumber { get; set; }

public DeviceStatus Status { get; set; }

public DateTime CreatedAt { get; set; }

}

public enum DeviceStatus

{

Active,

Inactive,

Disconnected

}

```

---

### \*\*3.5. Infrastructure (زیرساخت)\*\*

#### \*\*هدف:\*\*

این بخش شامل ابزارهایی است که برای مدیریت ارتباطات، دیتابیس‌ها، و پیام‌رسانی استفاده می‌شوند.

#### \*\*مثال: DeviceDbContext.cs\*\*

```csharp

public class DeviceDbContext : DbContext

{

public DeviceDbContext(DbContextOptions<DeviceDbContext> options) : base(options) { }

public DbSet<DeviceEntity> Devices { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<DeviceEntity>().HasKey(d => d.DeviceId);

modelBuilder.Entity<DeviceEntity>().Property(d => d.CreatedAt).HasDefaultValueSql("GETUTCDATE()");

}

}

```

---

### \*\*3.6. Extensions (افزونه‌ها)\*\*

#### \*\*هدف:\*\*

این بخش شامل تنظیمات Dependency Injection و سرویس‌های جانبی است.

#### \*\*مثال: ServiceCollectionExtensions.cs\*\*

```csharp

public static class ServiceCollectionExtensions

{

public static IServiceCollection AddDeviceManagementServices(this IServiceCollection services, IConfiguration configuration)

{

services.AddScoped<IDeviceService, DeviceService>();

services.AddScoped<IDeviceRepository, DeviceRepository>();

services.AddDbContext<DeviceDbContext>(options =>

options.UseSqlServer(configuration.GetConnectionString("DefaultConnection")));

return services;

}

}

```

---

### \*\*3.7. Program.cs\*\*

هدف:\*\*

این فایل برنامه را راه‌اندازی می‌کند و تنظیمات اولیه را اعمال می‌کند.

```csharp

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

builder.Services.AddDeviceManagementServices(builder.Configuration);

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.MapControllers();

app.Run();

```

---

4. ویژگی‌ها و قابلیت‌ها\*\*

4.1. اعتبارسنجی شماره سریال\*\*

- \*\*هدف:\*\* اطمینان از صحت شماره سریال دستگاه‌ها قبل از ثبت.

4.2. مدیریت وضعیت دستگاه\*\*

- \*\*هدف:\*\* مدیریت وضعیت دستگاه‌ها (فعال، غیرفعال، قطع شده) با استفاده از \*\*Background Service\*\*.

4.3. اتصال دستگاه به حساب کاربری\*\*

- \*\*هدف:\*\* اختصاص دستگاه‌ها به کاربران و مدیریت تعامل بین دستگاه‌ها و کاربران.

---

5. اضافه‌کردن موارد پیشنهادی\*\*

5.1. Validation مستقل\*\*

- استفاده از \*\*FluentValidation\*\* برای اعتبارسنجی داده‌ها.

- ایجاد کلاس‌های جداگانه برای اعتبارسنجی داده‌ها.

5.2. Unit of Work\*\*

- مدیریت تراکنش‌ها در سطح بالاتر برای اطمینان از یکپارچگی داده‌ها.

5.3. Domain Events\*\*

- ایجاد رویدادهای دامنه‌ای برای مدیریت ارتباط بین اجزا.

5.4. ایمن‌سازی داده‌ها\*\*

- رمزنگاری شماره سریال با AES یا SHA256.

- اعمال سیاست‌های Authorization برای محافظت از API.

5.5. لاگ‌گذاری\*\*

- استفاده از \*\*ILogger\*\* برای ثبت لاگ‌های دقیق.

5.6. جستجوی پیشرفته\*\*

- اضافه کردن متدهای جستجوی پیشرفته در Repository.

5.7. Soft Delete\*\*

- مدیریت حذف منطقی با افزودن فیلد `IsDeleted`.

5.8. مدیریت اتصال MQTT\*\*

- به‌روزرسانی وضعیت اتصال دستگاه‌ها به‌صورت بلادرنگ.

**4. نکات امنیتی و توسعه‌پذیری**

**✅ اعتبارسنجی ورودی‌ها**

* در سرویس‌ها از Validation برای داده‌های ورودی استفاده شود (به خصوص SerialNumber).
* امکان اضافه‌کردن FluentValidation یا Data Annotations برای سطوح پیشرفته‌تر.

**✅ مدیریت خطاها**

* استفاده از Exception Middleware برای بازگرداندن خطاهای ساختاریافته به کلاینت‌ها.
* در آینده می‌تونید از کلاس‌هایی مانند ApiResponse<T> و ErrorDetails برای پاسخ‌دهی استاندارد استفاده کنید.

**✅ JWT Auth**

* اضافه‌کردن احراز هویت JWT در سطح API برای جلوگیری از دسترسی غیرمجاز به دستگاه‌ها.
* می‌تونی [Authorize] روی کنترلر DevicesController بذاری تا فقط کاربران احراز هویت‌شده بتونن درخواست بدن.

**✅ لاگ‌گیری و مانیتورینگ**

* استفاده از Serilog برای لاگ‌گیری رویدادها و خطاها.
* نگهداری تاریخچه تغییرات دستگاه‌ها با Audit Log مفید هست (برای استفاده بعدی در بخش‌های حقوقی یا مانیتورینگ).

**جمع‌بندی**

ماژول DeviceManagementService اکنون دارای:

* ساختار ماژولار و تمیز
* لایه‌بندی صحیح: Controller, Service, Repository, DTO/Entity
* قابلیت توسعه‌پذیر، امن، و قابل نگهداری
* آماده برای گسترش با مواردی مثل MQTT، وضعیت اتصال لحظه‌ای، و حتی WebSocket

اگر خواستی این ماژول رو به پروژه اصلی اضافه کنیم، یا در مورد اتصال به ماژول‌های دیگر مثل UserManagement یا DeviceTracking صحبت کنیم، فقط بگو، تا بهم متصلش کنیم.

### \*\*توضیح کامل ماژول CommandService\*\*

ماژول \*\*CommandService\*\* یکی از اجزای کلیدی در سیستم مدیریت دستگاه‌های GPS است. این ماژول مسئولیت ارسال، ذخیره‌سازی، و مدیریت دستوراتی را بر عهده دارد که به دستگاه‌ها ارسال می‌شوند. همچنین، این ماژول شامل قابلیت‌هایی مانند اعتبارسنجی داده‌ها، مدیریت خطاها، پشتیبانی از ACK/Response از سمت دستگاه، ارتباط بلادرنگ با MQTT یا Kafka، لاگ‌گذاری حرفه‌ای، و مدیریت وضعیت دستورات است. در ادامه، تمام جنبه‌های این ماژول به‌صورت دقیق و ساختارمند توضیح داده شده است.

---

## \*\*1. ساختار کلی ماژول\*\*

```plaintext

/CommandService

│

├── Controllers

│ └── CommandsController.cs

│

├── Services

│ ├── CommandService.cs

│ └── ICommandService.cs

│

├── Repositories

│ ├── CommandRepository.cs

│ └── ICommandRepository.cs

│

├── Models

│ ├── CommandDto.cs

│ └── CommandEntity.cs

│

├── Enums

│ ├── CommandType.cs

│ └── CommandStatus.cs

│

├── Infrastructure

│ ├── Persistence

│ │ └── CommandDbContext.cs

│ └── Messaging

│ ├── KafkaEventProducer.cs

│ └── MqttMessageProducer.cs

│

├── Extensions

│ └── ServiceCollectionExtensions.cs

│

└── Program.cs

```

---

## \*\*2. توضیحات جامع برای هر بخش\*\*

### \*\*2.1. Controllers (کنترلرها)\*\*

#### \*\*هدف:\*\*

کنترلرها واسط بین کلاینت‌ها (مثل وب‌اپلیکیشن یا موبایل) و سرویس‌های داخلی هستند. آن‌ها درخواست‌های HTTP را دریافت کرده و پاسخ مناسب ارسال می‌کنند.

#### \*\*مثال: CommandsController.cs\*\*

```csharp

[ApiController]

[Route("api/[controller]")]

public class CommandsController : ControllerBase

{

private readonly ICommandService \_commandService;

public CommandsController(ICommandService commandService)

{

\_commandService = commandService;

}

/// <summary>

/// ارسال دستور به دستگاه

/// </summary>

[HttpPost]

public async Task<IActionResult> SendCommand([FromBody] CommandDto commandDto)

{

try

{

await \_commandService.SendCommandAsync(commandDto);

return Ok(ApiResponse<string>.SuccessResponse("دستور با موفقیت ارسال شد."));

}

catch (Exception ex)

{

return BadRequest(ApiResponse<string>.ErrorResponse(ex.Message));

}

}

/// <summary>

/// دریافت لیست دستورات ارسال‌شده به یک دستگاه

/// </summary>

[HttpGet("{deviceId}")]

public async Task<IActionResult> GetCommandsByDevice(

string deviceId,

[FromQuery] CommandType? commandType = null,

[FromQuery] CommandStatus? status = null,

[FromQuery] DateTime? fromDate = null,

[FromQuery] DateTime? toDate = null)

{

var commands = await \_commandService.GetCommandsByDeviceAsync(deviceId, commandType, status, fromDate, toDate);

return Ok(ApiResponse<List<CommandDto>>.SuccessResponse(commands));

}

}

```

---

### \*\*2.2. Services (سرویس‌ها)\*\*

#### \*\*هدف:\*\*

سرویس‌ها شامل منطق کسب‌وکار برنامه هستند. آن‌ها کارهایی مثل ارسال دستورات، ذخیره‌سازی دستورات، و دریافت لیست دستورات مرتبط با یک دستگاه را انجام می‌دهند.

#### \*\*مثال: ICommandService.cs\*\*

```csharp

public interface ICommandService

{

Task SendCommandAsync(CommandDto commandDto);

Task<List<CommandDto>> GetCommandsByDeviceAsync(string deviceId, CommandType? commandType, CommandStatus? status, DateTime? fromDate, DateTime? toDate);

Task UpdateCommandStatusAsync(Guid commandId, CommandStatus newStatus);

}

```

#### \*\*مثال: CommandService.cs\*\*

```csharp

public class CommandService : ICommandService

{

private readonly ICommandRepository \_commandRepository;

private readonly IMessageProducer \_messageProducer;

private readonly ILogger<CommandService> \_logger;

public CommandService(ICommandRepository commandRepository, IMessageProducer messageProducer, ILogger<CommandService> logger)

{

\_commandRepository = commandRepository;

\_messageProducer = messageProducer;

\_logger = logger;

}

public async Task SendCommandAsync(CommandDto commandDto)

{

// اعتبارسنجی داده‌ها

if (string.IsNullOrEmpty(commandDto.DeviceId))

throw new ArgumentException("شناسه دستگاه نمی‌تواند خالی باشد.");

if (!Enum.IsDefined(typeof(CommandType), commandDto.CommandType))

throw new ArgumentException("نوع دستور نامعتبر است.");

try

{

// ارسال دستور به دستگاه از طریق Kafka یا MQTT

await \_messageProducer.PublishAsync(commandDto);

// ذخیره دستور در دیتابیس

var commandEntity = new CommandEntity

{

DeviceId = commandDto.DeviceId,

Type = commandDto.CommandType,

Status = CommandStatus.Pending,

CreatedAt = DateTime.UtcNow

};

await \_commandRepository.SaveCommandAsync(commandEntity);

\_logger.LogInformation("دستور به دستگاه {DeviceId} با نوع {Type} ارسال شد.", commandDto.DeviceId, commandDto.CommandType);

}

catch (Exception ex)

{

\_logger.LogError(ex, "خطا در ارسال دستور به دستگاه {DeviceId}.", commandDto.DeviceId);

throw;

}

}

public async Task<List<CommandDto>> GetCommandsByDeviceAsync(string deviceId, CommandType? commandType, CommandStatus? status, DateTime? fromDate, DateTime? toDate)

{

var query = \_commandRepository.GetCommandsByDeviceAsync(deviceId);

if (commandType.HasValue)

query = query.Where(c => c.Type == commandType.Value);

if (status.HasValue)

query = query.Where(c => c.Status == status.Value);

if (fromDate.HasValue)

query = query.Where(c => c.CreatedAt >= fromDate.Value);

if (toDate.HasValue)

query = query.Where(c => c.CreatedAt <= toDate.Value);

return await query.ToListAsync();

}

public async Task UpdateCommandStatusAsync(Guid commandId, CommandStatus newStatus)

{

var command = await \_commandRepository.GetCommandByIdAsync(commandId);

if (command == null) throw new Exception("دستور پیدا نشد.");

command.Status = newStatus;

command.ExecutedAt = DateTime.UtcNow;

await \_commandRepository.UpdateCommandAsync(command);

}

}

```

---

### \*\*2.3. Repositories (مخازن)\*\*

#### \*\*هدف:\*\*

مخازن وظیفه مدیریت دسترسی به دیتابیس را دارند. آن‌ها شامل عملیات CRUD (Create, Read) برای داده‌ها هستند.

#### \*\*مثال: ICommandRepository.cs\*\*

```csharp

public interface ICommandRepository

{

Task SaveCommandAsync(CommandEntity command);

IQueryable<CommandEntity> GetCommandsByDeviceAsync(string deviceId);

Task<CommandEntity> GetCommandByIdAsync(Guid commandId);

Task UpdateCommandAsync(CommandEntity command);

}

```

#### \*\*مثال: CommandRepository.cs\*\*

```csharp

public class CommandRepository : ICommandRepository

{

private readonly CommandDbContext \_context;

public CommandRepository(CommandDbContext context)

{

\_context = context;

}

public async Task SaveCommandAsync(CommandEntity command)

{

\_context.Commands.Add(command);

await \_context.SaveChangesAsync();

}

public IQueryable<CommandEntity> GetCommandsByDeviceAsync(string deviceId)

{

return \_context.Commands.Where(c => c.DeviceId == deviceId);

}

public async Task<CommandEntity> GetCommandByIdAsync(Guid commandId)

{

return await \_context.Commands.FindAsync(commandId);

}

public async Task UpdateCommandAsync(CommandEntity command)

{

\_context.Commands.Update(command);

await \_context.SaveChangesAsync();

}

}

```

---

### \*\*2.4. Models (مدل‌ها)\*\*

#### \*\*هدف:\*\*

مدل‌ها شامل داده‌هایی هستند که در سراسر برنامه استفاده می‌شوند. آن‌ها به دو دسته تقسیم می‌شوند:

- \*\*DTO (Data Transfer Object)\*\*: برای انتقال داده‌ها بین لایه‌ها.

- \*\*Entities\*\*: برای ذخیره‌سازی داده‌ها در دیتابیس.

#### \*\*مثال: CommandDto.cs\*\*

```csharp

public class CommandDto

{

public string DeviceId { get; set; }

public CommandType CommandType { get; set; }

public CommandStatus Status { get; set; }

public DateTime CreatedAt { get; set; }

public DateTime? ExecutedAt { get; set; }

}

```

#### \*\*مثال: CommandEntity.cs\*\*

```csharp

public class CommandEntity

{

public Guid Id { get; set; }

public string DeviceId { get; set; }

public CommandType Type { get; set; }

public CommandStatus Status { get; set; }

public DateTime CreatedAt { get; set; }

public DateTime? ExecutedAt { get; set; }

}

```

---

### \*\*2.5. Enums (انواع شمارشی)\*\*

#### \*\*هدف:\*\*

استفاده از انواع شمارشی برای کنترل بهتر و امن‌تر مقادیر.

#### \*\*مثال: CommandType.cs\*\*

```csharp

public enum CommandType

{

Reboot, // راه‌اندازی مجدد

GetLocation, // دریافت موقعیت

SetConfig // تنظیم پیکربندی

}

```

#### \*\*مثال: CommandStatus.cs\*\*

```csharp

public enum CommandStatus

{

Pending, // در حال انتظار

Completed, // تکمیل شده

Failed // شکست خورده

}

```

---

### \*\*2.6. Infrastructure (زیرساخت)\*\*

#### \*\*هدف:\*\*

این بخش شامل ابزارهایی است که برای مدیریت ارتباطات، دیتابیس‌ها، و پیام‌رسانی استفاده می‌شوند.

#### \*\*مثال: CommandDbContext.cs\*\*

```csharp

public class CommandDbContext : DbContext

{

public DbSet<CommandEntity> Commands { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<CommandEntity>().HasKey(c => c.Id);

modelBuilder.Entity<CommandEntity>().Property(c => c.CreatedAt).HasDefaultValueSql("GETUTCDATE()");

}

}

```

#### \*\*مثال: KafkaEventProducer.cs\*\*

```csharp

public class KafkaEventProducer : IMessageProducer

{

private readonly IProducer<Null, string> \_producer;

public KafkaEventProducer()

{

var config = new ProducerConfig

{

BootstrapServers = "localhost:9092"

};

\_producer = new ProducerBuilder<Null, string>(config).Build();

}

public async Task PublishAsync<T>(T message)

{

int retryCount = 3;

while (retryCount > 0)

{

try

{

var serializedMessage = JsonConvert.SerializeObject(message);

await \_producer.ProduceAsync("commands", new Message<Null, string> { Value = serializedMessage });

break; // اگر ارسال موفق بود، حلقه را متوقف کن

}

catch (Exception ex)

{

retryCount--;

if (retryCount == 0)

{

throw new Exception("ارسال دستور به Kafka شکست خورد.", ex);

}

await Task.Delay(1000); // تاخیر قبل از تلاش مجدد

}

}

}

}

```

---

### \*\*2.7. Extensions (افزونه‌ها)\*\*

#### \*\*هدف:\*\*

این بخش شامل تنظیمات Dependency Injection و سرویس‌های جانبی است.

#### \*\*مثال: ServiceCollectionExtensions.cs\*\*

```csharp

public static class ServiceCollectionExtensions

{

public static IServiceCollection AddCommandServices(this IServiceCollection services, IConfiguration configuration)

{

services.AddDbContext<CommandDbContext>(options =>

options.UseSqlServer(configuration.GetConnectionString("DefaultConnection")));

services.AddScoped<ICommandRepository, CommandRepository>();

services.AddScoped<ICommandService, CommandService>();

services.AddSingleton<IMessageProducer, KafkaEventProducer>();

return services;

}

}

```

---

### \*\*2.8. Program.cs\*\*

#### \*\*هدف:\*\*

این فایل برنامه را راه‌اندازی می‌کند و تنظیمات اولیه را اعمال می‌کند.

#### \*\*مثال: Program.cs\*\*

```csharp

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

builder.Services.AddCommandServices(builder.Configuration);

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.MapControllers();

app.Run();

```

CommandService

│

├── Consumers

│ ├── CommandAckConsumer.cs // مصرف‌کننده ACKها از دستگاه

│ └── CommandResponseConsumer.cs // مصرف‌کننده پاسخ یا نتیجه دستور

│

├── Events

│ ├── CommandAckEvent.cs // رخداد ACK

│ └── CommandResponseEvent.cs // رخداد پاسخ دستور

│

├── Handlers

│ ├── ICommandAckHandler.cs

│ ├── CommandAckHandler.cs

│ ├── ICommandResponseHandler.cs

│ └── CommandResponseHandler.cs

public class CommandAckEvent

{

public Guid CommandId { get; set; }

public string DeviceId { get; set; }

public DateTime AcknowledgedAt { get; set; }

}

public class CommandResponseEvent

{

public Guid CommandId { get; set; }

public string DeviceId { get; set; }

public string Payload { get; set; } // نتیجه اجرای دستور

public bool IsSuccess { get; set; }

public DateTime RespondedAt { get; set; }

}

public class CommandAckConsumer : IHostedService

{

private readonly IServiceProvider \_serviceProvider;

private readonly ILogger<CommandAckConsumer> \_logger;

public CommandAckConsumer(IServiceProvider serviceProvider, ILogger<CommandAckConsumer> logger)

{

\_serviceProvider = serviceProvider;

\_logger = logger;

}

public Task StartAsync(CancellationToken cancellationToken)

{

// راه‌اندازی Kafka/MQTT listener

Task.Run(async () =>

{

// شبیه‌سازی دریافت پیام

while (!cancellationToken.IsCancellationRequested)

{

var ackEvent = await FakeReceiveAck(); // در عمل این پیام از Kafka یا MQTT میاد

using var scope = \_serviceProvider.CreateScope();

var handler = scope.ServiceProvider.GetRequiredService<ICommandAckHandler>();

await handler.HandleAsync(ackEvent);

}

});

return Task.CompletedTask;

}

public Task StopAsync(CancellationToken cancellationToken) => Task.CompletedTask;

private Task<CommandAckEvent> FakeReceiveAck() => Task.FromResult(new CommandAckEvent

{

CommandId = Guid.NewGuid(),

DeviceId = "device-001",

AcknowledgedAt = DateTime.UtcNow

});

}

**3.3. Handlers (پردازش رخدادها)**

**ICommandAckHandler.cs**

public interface ICommandAckHandler

{

Task HandleAsync(CommandAckEvent ackEvent);

}

CommandAckHandler.cs

public class CommandAckHandler : ICommandAckHandler

{

private readonly ICommandRepository \_repository;

private readonly ILogger<CommandAckHandler> \_logger;

public CommandAckHandler(ICommandRepository repository, ILogger<CommandAckHandler> logger)

{

\_repository = repository;

\_logger = logger;

}

public async Task HandleAsync(CommandAckEvent ackEvent)

{

var command = await \_repository.GetCommandByIdAsync(ackEvent.CommandId);

if (command == null)

{

\_logger.LogWarning("ACK دریافت شد اما دستور یافت نشد: {CommandId}", ackEvent.CommandId);

return;

}

command.Status = CommandStatus.Completed; // یا در صورت لزوم Acknowledged

command.ExecutedAt = ackEvent.AcknowledgedAt;

await \_repository.UpdateCommandAsync(command);

\_logger.LogInformation("ACK برای دستور {CommandId} ذخیره شد.", ackEvent.CommandId);

}

}

برای پاسخ کامل دستور (CommandResponseEvent) هم می‌تونی همون ساختار رو داشته باشی با ویژگی‌هایی مثل:

* ذخیره لاگ اجرای دستور
* تحلیل payload (مثلاً برای پاسخ GPS)
* **ثبت در DI Container**
* در ServiceCollectionExtensions.cs:

services.AddScoped<ICommandAckHandler, CommandAckHandler>();

services.AddHostedService<CommandAckConsumer>();

## \*\*3. نتیجه‌گیری\*\*

ماژول \*\*CommandService\*\* یک ماژول کامل و مستقل است که می‌تواند به‌عنوان بخشی از یک سیستم ردیابی GPS استفاده شود. تمام بخش‌ها شامل کنترلرها، سرویس‌ها، مخازن، مدیریت دیتابیس، ارتباط بلادرنگ با Kafka یا MQTT، و تنظیمات برنامه به همراه کامنت‌های توضیحی نوشته شده‌اند. اگر نیاز به توضیحات بیشتری دارید،

### \*\*ماژول NotificationService\*\*

در ادامه، ماژول \*\*NotificationService\*\* به‌صورت کامل و با توجه به تمام نیازها و قابلیت‌های پیشنهادی شما پیاده‌سازی می‌شود. این ماژول شامل بخش‌های مختلفی است که هر کدام وظیفه خاصی را بر عهده دارند. تمام کدها به همراه توضیحات کامل نوشته شده‌اند.

---

## \*\*1. ساختار کلی ماژول\*\*

```plaintext

/NotificationService

│

├── Controllers

│ └── NotificationsController.cs

│

├── Services

│ ├── NotificationService.cs

│ └── INotificationService.cs

│

├── Repositories

│ ├── NotificationRepository.cs

│ └── INotificationRepository.cs

│

├── Models

│ ├── NotificationDto.cs

│ └── NotificationEntity.cs

│

├── Infrastructure

│ ├── Persistence

│ │ └── NotificationDbContext.cs

│ ├── Messaging

│ │ └── KafkaEventConsumer.cs

│ └── RealTimeCommunication

│ └── SignalRNotificationHub.cs

│

├── Extensions

│ └── ServiceCollectionExtensions.cs

│

└── Program.cs

```

---

## \*\*2. پیاده‌سازی هر بخش\*\*

### \*\*2.1. Controllers (کنترلرها)\*\*

#### \*\*هدف:\*\*

کنترلرها واسط بین کلاینت‌ها (مثل وب‌اپلیکیشن یا موبایل) و سرویس‌های داخلی هستند. آن‌ها درخواست‌های HTTP را دریافت کرده و پاسخ مناسب ارسال می‌کنند.

#### \*\*مثال: NotificationsController.cs\*\*

```csharp

[ApiController]

[Route("api/[controller]")]

public class NotificationsController : ControllerBase

{

private readonly INotificationService \_notificationService;

public NotificationsController(INotificationService notificationService)

{

\_notificationService = notificationService;

}

/// <summary>

/// ارسال اعلان جدید

/// </summary>

[HttpPost]

public async Task<IActionResult> SendNotification([FromBody] NotificationDto notificationDto)

{

await \_notificationService.SendNotificationAsync(notificationDto);

return Ok(new { Message = "اعلان با موفقیت ارسال شد." });

}

/// <summary>

/// دریافت لیست اعلان‌های یک کاربر

/// </summary>

[HttpGet("{userId}")]

public async Task<IActionResult> GetUserNotifications(string userId)

{

var notifications = await \_notificationService.GetUserNotificationsAsync(userId);

return Ok(notifications);

}

/// <summary>

/// علامت‌گذاری اعلان به عنوان خوانده‌شده

/// </summary>

[HttpPut("{notificationId}/mark-as-read")]

public async Task<IActionResult> MarkAsRead(int notificationId)

{

await \_notificationService.MarkAsReadAsync(notificationId);

return Ok(new { Message = "اعلان به عنوان خوانده‌شده علامت‌گذاری شد." });

}

/// <summary>

/// حذف اعلان

/// </summary>

[HttpDelete("{notificationId}")]

public async Task<IActionResult> DeleteNotification(int notificationId)

{

await \_notificationService.DeleteNotificationAsync(notificationId);

return Ok(new { Message = "اعلان با موفقیت حذف شد." });

}

}

```

---

### \*\*2.2. Services (سرویس‌ها)\*\*

#### \*\*هدف:\*\*

سرویس‌ها شامل منطق کسب‌وکار برنامه هستند. آن‌ها کارهایی مثل ارسال اعلانات، ذخیره‌سازی اعلانات، و دریافت لیست اعلانات مرتبط با یک کاربر را انجام می‌دهند.

#### \*\*مثال: INotificationService.cs\*\*

```csharp

public interface INotificationService

{

Task SendNotificationAsync(NotificationDto notificationDto);

Task<List<NotificationDto>> GetUserNotificationsAsync(string userId);

Task MarkAsReadAsync(int notificationId);

Task DeleteNotificationAsync(int notificationId);

}

```

#### \*\*مثال: NotificationService.cs\*\*

```csharp

public class NotificationService : INotificationService

{

private readonly INotificationRepository \_notificationRepository;

private readonly IHubContext<SignalRNotificationHub> \_signalRHubContext;

public NotificationService(INotificationRepository notificationRepository, IHubContext<SignalRNotificationHub> signalRHubContext)

{

\_notificationRepository = notificationRepository;

\_signalRHubContext = signalRHubContext;

}

public async Task SendNotificationAsync(NotificationDto notificationDto)

{

var notificationEntity = new NotificationEntity

{

UserId = notificationDto.UserId,

Title = notificationDto.Title,

Message = notificationDto.Message,

Type = notificationDto.Type,

IsRead = false,

CreatedAt = DateTime.UtcNow,

ExpiresAt = DateTime.UtcNow.AddMinutes(30) // زمان انقضا

};

await \_notificationRepository.SaveNotificationAsync(notificationEntity);

// ارسال بلادرنگ اعلان به کلاینت

await \_signalRHubContext.Clients.User(notificationDto.UserId).SendAsync("ReceiveNotification", notificationDto);

}

public async Task<List<NotificationDto>> GetUserNotificationsAsync(string userId)

{

var notifications = await \_notificationRepository.GetUserNotificationsAsync(userId);

return notifications.Select(n => new NotificationDto

{

Id = n.Id,

UserId = n.UserId,

Title = n.Title,

Message = n.Message,

Type = n.Type,

IsRead = n.IsRead,

CreatedAt = n.CreatedAt

}).ToList();

}

public async Task MarkAsReadAsync(int notificationId)

{

await \_notificationRepository.MarkAsReadAsync(notificationId);

}

public async Task DeleteNotificationAsync(int notificationId)

{

await \_notificationRepository.DeleteNotificationAsync(notificationId);

}

}

```

---

### \*\*2.3. Repositories (مخازن)\*\*

#### \*\*هدف:\*\*

مخازن وظیفه مدیریت دسترسی به دیتابیس را دارند. آن‌ها شامل عملیات CRUD (Create, Read, Update, Delete) برای داده‌ها هستند.

#### \*\*مثال: INotificationRepository.cs\*\*

```csharp

public interface INotificationRepository

{

Task SaveNotificationAsync(NotificationEntity notification);

Task<List<NotificationEntity>> GetUserNotificationsAsync(string userId);

Task MarkAsReadAsync(int notificationId);

Task DeleteNotificationAsync(int notificationId);

}

```

#### \*\*مثال: NotificationRepository.cs\*\*

```csharp

public class NotificationRepository : INotificationRepository

{

private readonly NotificationDbContext \_context;

public NotificationRepository(NotificationDbContext context)

{

\_context = context;

}

public async Task SaveNotificationAsync(NotificationEntity notification)

{

await \_context.Notifications.AddAsync(notification);

await \_context.SaveChangesAsync();

}

public async Task<List<NotificationEntity>> GetUserNotificationsAsync(string userId)

{

return await \_context.Notifications.Where(n => n.UserId == userId).ToListAsync();

}

public async Task MarkAsReadAsync(int notificationId)

{

var notification = await \_context.Notifications.FindAsync(notificationId);

if (notification != null)

{

notification.IsRead = true;

notification.ReadAt = DateTime.UtcNow;

await \_context.SaveChangesAsync();

}

}

public async Task DeleteNotificationAsync(int notificationId)

{

var notification = await \_context.Notifications.FindAsync(notificationId);

if (notification != null)

{

\_context.Notifications.Remove(notification);

await \_context.SaveChangesAsync();

}

}

}

```

---

### \*\*2.4. Models (مدل‌ها)\*\*

#### \*\*هدف:\*\*

مدل‌ها شامل داده‌هایی هستند که در سراسر برنامه استفاده می‌شوند. آن‌ها به دو دسته تقسیم می‌شوند:

- \*\*DTO (Data Transfer Object)\*\*: برای انتقال داده‌ها بین لایه‌ها.

- \*\*Entities\*\*: برای ذخیره‌سازی داده‌ها در دیتابیس.

#### \*\*مثال: NotificationDto.cs\*\*

```csharp

public class NotificationDto

{

public Guid Id { get; set; }

public string UserId { get; set; }

public string Title { get; set; }

public string Message { get; set; }

public NotificationType Type { get; set; }

public bool IsRead { get; set; }

public DateTime CreatedAt { get; set; }

}

```

#### \*\*مثال: NotificationEntity.cs\*\*

```csharp

public class NotificationEntity

{

public Guid Id { get; set; }

public string UserId { get; set; }

public string Title { get; set; }

public string Message { get; set; }

public NotificationType Type { get; set; }

public bool IsRead { get; set; }

public DateTime CreatedAt { get; set; }

public DateTime? ReadAt { get; set; }

public DateTime ExpiresAt { get; set; }

}

```

---

### \*\*2.5. Infrastructure (زیرساخت)\*\*

#### \*\*هدف:\*\*

این بخش شامل تنظیمات مربوط به دیتابیس، ارتباط بلادرنگ با SignalR، و مصرف رویدادها از طریق Kafka است.

#### \*\*مثال: NotificationDbContext.cs\*\*

```csharp

public class NotificationDbContext : DbContext

{

public NotificationDbContext(DbContextOptions<NotificationDbContext> options) : base(options) { }

public DbSet<NotificationEntity> Notifications { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<NotificationEntity>(entity =>

{

entity.HasKey(n => n.Id);

entity.Property(n => n.CreatedAt).HasDefaultValueSql("GETUTCDATE()");

});

}

}

```

#### \*\*مثال: SignalRNotificationHub.cs\*\*

```csharp

public class SignalRNotificationHub : Hub

{

public async Task SendNotificationToUser(string userId, NotificationDto notification)

{

await Clients.User(userId).SendAsync("ReceiveNotification", notification);

}

}

```

#### \*\*مثال: KafkaEventConsumer.cs\*\*

```csharp

public class KafkaEventConsumer

{

private readonly IConsumer<Ignore, string> \_consumer;

public KafkaEventConsumer(string kafkaBootstrapServers)

{

var config = new ConsumerConfig

{

BootstrapServers = kafkaBootstrapServers,

GroupId = "notification-group",

AutoOffsetReset = AutoOffsetReset.Earliest

};

\_consumer = new ConsumerBuilder<Ignore, string>(config).Build();

}

public async Task ConsumeAsync()

{

\_consumer.Subscribe("notifications");

while (true)

{

var consumeResult = \_consumer.Consume();

var notificationDto = JsonConvert.DeserializeObject<NotificationDto>(consumeResult.Message.Value);

// ارسال اعلان به کاربر

// می‌توانید اینجا اعلان را ذخیره یا به کاربر ارسال کنید

}

}

}

```

---

### \*\*2.6. Extensions (تنظیمات Dependency Injection)\*\*

#### \*\*هدف:\*\*

این بخش شامل تنظیمات Dependency Injection و سرویس‌های جانبی است.

#### \*\*مثال: ServiceCollectionExtensions.cs\*\*

```csharp

public static class ServiceCollectionExtensions

{

public static IServiceCollection AddNotificationServices(this IServiceCollection services, IConfiguration configuration)

{

services.AddDbContext<NotificationDbContext>(options =>

options.UseSqlServer(configuration.GetConnectionString("DefaultConnection")));

services.AddScoped<INotificationRepository, NotificationRepository>();

services.AddScoped<INotificationService, NotificationService>();

services.AddSignalR();

return services;

}

}

```

---

### \*\*2.7. Program.cs\*\*

#### \*\*هدف:\*\*

این فایل برنامه را راه‌اندازی می‌کند و تنظیمات اولیه را اعمال می‌کند.

#### \*\*مثال: Program.cs\*\*

```csharp

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

builder.Services.AddNotificationServices(builder.Configuration);

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.MapControllers();

app.MapHub<SignalRNotificationHub>("/notifications");

app.Run();

**امنیت و احراز هویت**

* بررسی کن که userId در GetUserNotificationsAsync با توکن JWT تطابق داشته باشه. یعنی:
* if (User.FindFirst(ClaimTypes.NameIdentifier)?.Value != userId)
* return Unauthorized();

**حذف خودکار اعلان‌های منقضی‌شده**

* بهتره یه **Background Service** اضافه کنی که اعلان‌هایی با ExpiresAt < DateTime.UtcNow رو حذف یا غیرفعال کنه (مثلاً با IsExpired).
  + می‌تونی از IHostedService یا Quartz.NET استفاده کنی.

**📈 وضعیت ارسال SignalR**

* می‌تونی از SignalR Feedback استفاده کنی که بفهمی پیام واقعاً به کلاینت رسیده یا نه.
* برای اطمینان بیشتر، یه جدول DeliveryStatus هم می‌تونه اضافه شه.

**🧪 تست‌پذیری**

* تزریق ILogger<T> به سرویس‌ها برای لاگ دقیق اتفاقات.
* تعریف تست‌های یونیت با استفاده از Moq برای INotificationService و INotificationRepository.

**🗂️ دسته‌بندی اعلان‌ها**

* می‌تونی Enum نوع اعلان (NotificationType) رو توسعه بدی مثلاً:  
  Info, Warning, Error, System, Message و...
* بعد توی UI یا کلاینت بر اساس نوع دسته‌بندی نمایش بدی.

**📤 دریافت اعلان‌های خوانده‌نشده**

* اضافه کردن متد جدید:

[HttpGet("{userId}/unread")]

public async Task<IActionResult> GetUnreadNotifications(string userId) { ... }

قابلیت | توضیح

ارسال گروهی | به جای UserId، یه لیست از UserIdها دریافت کن

زمان‌بندی اعلان‌ها | ارسال در زمان مشخص آینده (مثل ScheduledAt)

ذخیره تاریخچه تغییر وضعیت | مثل ReadAt و DeletedAt، برای audit دقیق‌تر

اعلانات سیستمی | مثل اعلان‌های عمومی بدون وابستگی به UserId

// ساختار پروژه ASP.NET Core - NotificationService

// Program.cs

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

builder.Services.AddSignalR();

// Dependency Injections

builder.Services.AddScoped<INotificationService, NotificationService>();

builder.Services.AddScoped<INotificationRepository, NotificationRepository>();

builder.Services.AddDbContext<NotificationDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

endpoints.MapHub<SignalRNotificationHub>("/notificationhub");

});

app.Run();

// Models/NotificationDto.cs

public class NotificationDto

{

public Guid Id { get; set; }

public string UserId { get; set; }

public string Title { get; set; }

public string Message { get; set; }

public NotificationType Type { get; set; }

public bool IsRead { get; set; }

public DateTime CreatedAt { get; set; }

}

// Models/NotificationEntity.cs

public class NotificationEntity

{

public Guid Id { get; set; }

public string UserId { get; set; }

public string Title { get; set; }

public string Message { get; set; }

public NotificationType Type { get; set; }

public bool IsRead { get; set; }

public DateTime CreatedAt { get; set; }

public DateTime? ReadAt { get; set; }

public DateTime ExpiresAt { get; set; }

}

// Enums/NotificationType.cs

public enum NotificationType

{

Info,

Warning,

Error,

Success

}

// Infrastructure/NotificationDbContext.cs

public class NotificationDbContext : DbContext

{

public NotificationDbContext(DbContextOptions<NotificationDbContext> options) : base(options) {}

public DbSet<NotificationEntity> Notifications { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<NotificationEntity>().HasKey(n => n.Id);

modelBuilder.Entity<NotificationEntity>().Property(n => n.CreatedAt).HasDefaultValueSql("GETUTCDATE()");

}

}

// Services/INotificationService.cs

public interface INotificationService

{

Task SendNotificationAsync(NotificationDto notificationDto);

Task<List<NotificationDto>> GetUserNotificationsAsync(string userId);

Task MarkAsReadAsync(int notificationId);

Task DeleteNotificationAsync(int notificationId);

}

// Services/NotificationService.cs

public class NotificationService : INotificationService

{

private readonly INotificationRepository \_repo;

private readonly IHubContext<SignalRNotificationHub> \_hub;

public NotificationService(INotificationRepository repo, IHubContext<SignalRNotificationHub> hub)

{

\_repo = repo;

\_hub = hub;

}

public async Task SendNotificationAsync(NotificationDto dto)

{

var entity = new NotificationEntity

{

Id = Guid.NewGuid(),

UserId = dto.UserId,

Title = dto.Title,

Message = dto.Message,

Type = dto.Type,

IsRead = false,

CreatedAt = DateTime.UtcNow,

ExpiresAt = DateTime.UtcNow.AddMinutes(30)

};

await \_repo.SaveNotificationAsync(entity);

await \_hub.Clients.User(dto.UserId).SendAsync("ReceiveNotification", dto);

}

public async Task<List<NotificationDto>> GetUserNotificationsAsync(string userId)

{

var list = await \_repo.GetUserNotificationsAsync(userId);

return list.Select(n => new NotificationDto

{

Id = n.Id,

UserId = n.UserId,

Title = n.Title,

Message = n.Message,

Type = n.Type,

IsRead = n.IsRead,

CreatedAt = n.CreatedAt

}).ToList();

}

public Task MarkAsReadAsync(int id) => \_repo.MarkAsReadAsync(id);

public Task DeleteNotificationAsync(int id) => \_repo.DeleteNotificationAsync(id);

}

// Repositories/INotificationRepository.cs

public interface INotificationRepository

{

Task SaveNotificationAsync(NotificationEntity notification);

Task<List<NotificationEntity>> GetUserNotificationsAsync(string userId);

Task MarkAsReadAsync(int id);

Task DeleteNotificationAsync(int id);

}

// Repositories/NotificationRepository.cs

public class NotificationRepository : INotificationRepository

{

private readonly NotificationDbContext \_context;

public NotificationRepository(NotificationDbContext context)

{

\_context = context;

}

public async Task SaveNotificationAsync(NotificationEntity notification)

{

await \_context.Notifications.AddAsync(notification);

await \_context.SaveChangesAsync();

}

public Task<List<NotificationEntity>> GetUserNotificationsAsync(string userId)

=> \_context.Notifications.Where(n => n.UserId == userId).ToListAsync();

public async Task MarkAsReadAsync(int id)

{

var n = await \_context.Notifications.FindAsync(id);

if (n != null)

{

n.IsRead = true;

n.ReadAt = DateTime.UtcNow;

await \_context.SaveChangesAsync();

}

}

public async Task DeleteNotificationAsync(int id)

{

var n = await \_context.Notifications.FindAsync(id);

if (n != null)

{

\_context.Notifications.Remove(n);

await \_context.SaveChangesAsync();

}

}

}

// Controllers/NotificationsController.cs

[ApiController]

[Route("api/[controller]")]

public class NotificationsController : ControllerBase

{

private readonly INotificationService \_service;

public NotificationsController(INotificationService service)

{

\_service = service;

}

[HttpPost]

public async Task<IActionResult> Send([FromBody] NotificationDto dto)

{

await \_service.SendNotificationAsync(dto);

return Ok("اعلان ارسال شد.");

}

[HttpGet("{userId}")]

public async Task<IActionResult> Get(string userId)

{

var list = await \_service.GetUserNotificationsAsync(userId);

return Ok(list);

}

[HttpPut("{id}/read")]

public async Task<IActionResult> Read(int id)

{

await \_service.MarkAsReadAsync(id);

return Ok("علامت‌گذاری شد.");

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(int id)

{

await \_service.DeleteNotificationAsync(id);

return Ok("حذف شد.");

}

}

// RealTimeCommunication/SignalRNotificationHub.cs

public class SignalRNotificationHub : Hub

{

public async Task SendNotificationToUser(string userId, NotificationDto notification)

{

await Clients.User(userId).SendAsync("ReceiveNotification", notification);

}

}

معماری تمیز (Service، Repository)

استفاده از SignalR برای اعلان بلادرنگ

مدل داده‌ای با تاریخ انقضا و زمان خوانده شدن اعلان

APIهایی برای ارسال، خواندن، دریافت و حذف اعلان

ثبت در دیتابیس با Entity Framework Core

قابلیت گسترش‌پذیری با enum نوع اعلان

dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer

dotnet add package System.IdentityModel.Tokens.Jwt

"JwtSettings": {

"Issuer": "MyApp",

"Audience": "MyAppAudience",

"SecretKey": "this\_is\_a\_very\_secret\_key\_1234567890",

"ExpiresInMinutes": 60

}

public class JwtSettings

{

public string Issuer { get; set; }

public string Audience { get; set; }

public string SecretKey { get; set; }

public int ExpiresInMinutes { get; set; }

}

builder.Services.Configure<JwtSettings>(builder.Configuration.GetSection("JwtSettings"));

var jwtSettings = builder.Configuration.GetSection("JwtSettings").Get<JwtSettings>();

builder.Services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(options =>

{

options.RequireHttpsMetadata = false;

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = jwtSettings.Issuer,

ValidAudience = jwtSettings.Audience,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSettings.SecretKey))

};

// Attach user ID to SignalR context

options.Events = new JwtBearerEvents

{

OnMessageReceived = context =>

{

var accessToken = context.Request.Query["access\_token"];

var path = context.HttpContext.Request.Path;

if (!string.IsNullOrEmpty(accessToken) && path.StartsWithSegments("/notificationHub"))

{

context.Token = accessToken;

}

return Task.CompletedTask;

}

};

});

builder.Services.AddAuthorization();

[ApiController]

[Route("api/auth")]

public class AuthController : ControllerBase

{

private readonly JwtSettings \_jwtSettings;

public AuthController(IOptions<JwtSettings> jwtSettings)

{

\_jwtSettings = jwtSettings.Value;

}

[HttpPost("login")]

public IActionResult Login([FromBody] LoginRequest model)

{

if (model.Username == "admin" && model.Password == "1234")

{

var claims = new[]

{

new Claim(ClaimTypes.NameIdentifier, "1"),

new Claim(ClaimTypes.Name, model.Username)

};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_jwtSettings.SecretKey));

var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var token = new JwtSecurityToken(

issuer: \_jwtSettings.Issuer,

audience: \_jwtSettings.Audience,

claims: claims,

expires: DateTime.UtcNow.AddMinutes(\_jwtSettings.ExpiresInMinutes),

signingCredentials: creds

);

return Ok(new { token = new JwtSecurityTokenHandler().WriteToken(token) });

}

return Unauthorized();

}

}

public class LoginRequest

{

public string Username { get; set; }

public string Password { get; set; }

}

dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Tools

public class ApplicationDbContext : IdentityDbContext<ApplicationUser, IdentityRole, string>

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

// جدول‌های دیگرت رو هم اینجا تعریف کن

}

public class ApplicationUser : IdentityUser

{

public string FullName { get; set; }

// فیلدهای دلخواه مثل تاریخ تولد، وضعیت، نوتیفیکیشن فعال بودن و ... اینجا اضافه کن

}

builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

builder.Services.AddIdentity<ApplicationUser, IdentityRole>(options =>

{

options.Password.RequireDigit = true;

options.Password.RequiredLength = 6;

options.User.RequireUniqueEmail = true;

})

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddDefaultTokenProviders();

dotnet ef migrations add InitIdentity

dotnet ef database update

public class TokenService

{

private readonly JwtSettings \_jwtSettings;

public TokenService(IOptions<JwtSettings> jwtSettings)

{

\_jwtSettings = jwtSettings.Value;

}

public string GenerateToken(ApplicationUser user, IList<string> roles)

{

var claims = new List<Claim>

{

new Claim(ClaimTypes.NameIdentifier, user.Id),

new Claim(ClaimTypes.Name, user.UserName)

};

foreach (var role in roles)

{

claims.Add(new Claim(ClaimTypes.Role, role));

}

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_jwtSettings.SecretKey));

var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var token = new JwtSecurityToken(

\_jwtSettings.Issuer,

\_jwtSettings.Audience,

claims,

expires: DateTime.UtcNow.AddMinutes(\_jwtSettings.ExpiresInMinutes),

signingCredentials: creds

);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

[Route("api/auth")]

[ApiController]

public class AuthController : ControllerBase

{

private readonly UserManager<ApplicationUser> \_userManager;

private readonly SignInManager<ApplicationUser> \_signInManager;

private readonly TokenService \_tokenService;

public AuthController(UserManager<ApplicationUser> userManager, SignInManager<ApplicationUser> signInManager, TokenService tokenService)

{

\_userManager = userManager;

\_signInManager = signInManager;

\_tokenService = tokenService;

}

[HttpPost("register")]

public async Task<IActionResult> Register([FromBody] RegisterDto dto)

{

var user = new ApplicationUser

{

UserName = dto.Username,

Email = dto.Email,

FullName = dto.FullName

};

var result = await \_userManager.CreateAsync(user, dto.Password);

if (!result.Succeeded)

return BadRequest(result.Errors);

// نقش پیش‌فرض

await \_userManager.AddToRoleAsync(user, "User");

return Ok("User registered successfully");

}

[HttpPost("login")]

public async Task<IActionResult> Login([FromBody] LoginDto dto)

{

var user = await \_userManager.FindByNameAsync(dto.Username);

if (user == null) return Unauthorized("Invalid credentials");

var result = await \_signInManager.CheckPasswordSignInAsync(user, dto.Password, false);

if (!result.Succeeded) return Unauthorized("Invalid credentials");

var roles = await \_userManager.GetRolesAsync(user);

var token = \_tokenService.GenerateToken(user, roles);

return Ok(new { token });

}

}

public class RegisterDto

{

public string Username { get; set; }

public string Email { get; set; }

public string FullName { get; set; }

public string Password { get; set; }

}

public class LoginDto

{

public string Username { get; set; }

public string Password { get; set; }

}

[Authorize(Roles = "Admin")]

[HttpGet("admin-area")]

public IActionResult GetAdminArea()

{

return Ok("Welcome Admin");

}

var roleManager = app.Services.GetRequiredService<RoleManager<IdentityRole>>();

if (!await roleManager.RoleExistsAsync("Admin"))

await roleManager.CreateAsync(new IdentityRole("Admin"));

if (!await roleManager.RoleExistsAsync("User"))

await roleManager.CreateAsync(new IdentityRole("User"));

## \*\*3. نتیجه‌گیری\*\*

ماژول \*\*NotificationService\*\* یک ماژول کامل و مستقل است که می‌تواند به‌عنوان بخشی از یک سیستم ردیابی GPS استفاده شود. تمام بخش‌ها شامل کنترلرها، سرویس‌ها، مخازن، مدیریت دیتابیس، ارتباط بلادرنگ با SignalR، و تنظیمات برنامه به همراه کامنت‌های توضیحی نوشته شده‌اند.

بله، متوجه شدم. در ادامه، ماژول \*\*AuthService\*\* به‌صورت کامل و با تمام جزئیات توضیح داده می‌شود. این ماژول شامل تمام بخش‌های لازم برای مدیریت احراز هویت (Authentication) و تولید توکن‌های JWT است. تمام کدها و توضیحات به‌صورت واضح و مرتب نوشته شده‌اند.

---

## \*\*ماژول AuthService\*\*

### \*\*1. ساختار کلی ماژول\*\*

```plaintext

/AuthService

│

├── Controllers

│ └── AuthController.cs

│

├── Services

│ ├── AuthService.cs

│ └── IAuthService.cs

│

├── Repositories

│ ├── UserRepository.cs

│ └── IUserRepository.cs

│

├── Models

│ ├── UserDto.cs

│ └── UserEntity.cs

│

├── Enums

│ └── UserRole.cs

│

├── Token

│ └── JwtTokenGenerator.cs

│

├── Security

│ ├── IPTrackingMiddleware.cs

│ └── MfaService.cs

│

├── OpenIddict

│ └── OpenIddictConfiguration.cs

│

├── appsettings.json

│

└── Program.cs

```

---

### \*\*2. توضیحات جامع برای هر بخش\*\*

#### \*\*2.1. Controllers (کنترلرها)\*\*

##### \*\*هدف:\*\*

کنترلرها واسط بین کلاینت‌ها (مثل وب‌اپلیکیشن یا موبایل) و سرویس‌های داخلی هستند. آن‌ها درخواست‌های HTTP را دریافت کرده و پاسخ مناسب ارسال می‌کنند.

##### \*\*مثال: AuthController.cs\*\*

```csharp

[ApiController]

[Route("api/[controller]")]

public class AuthController : ControllerBase

{

private readonly IAuthService \_authService;

public AuthController(IAuthService authService)

{

\_authService = authService;

}

/// <summary>

/// ثبت‌نام کاربر جدید

/// </summary>

[HttpPost("register")]

public async Task<IActionResult> Register([FromBody] UserDto userDto)

{

var result = await \_authService.RegisterAsync(userDto);

return Ok(result);

}

/// <summary>

/// ورود کاربر و دریافت توکن

/// </summary>

[HttpPost("login")]

public async Task<IActionResult> Login([FromBody] UserDto userDto)

{

var token = await \_authService.LoginAsync(userDto);

return Ok(new { Token = token });

}

/// <summary>

/// تولید توکن جدید با Refresh Token

/// </summary>

[HttpPost("refresh-token")]

public async Task<IActionResult> RefreshToken([FromBody] string refreshToken)

{

var newToken = await \_authService.RefreshTokenAsync(refreshToken);

return Ok(new { Token = newToken });

}

/// <summary>

/// تأیید دومرحله‌ای (MFA) با OTP

/// </summary>

[HttpPost("verify-otp")]

public async Task<IActionResult> VerifyOtp([FromBody] OtpVerificationDto otpDto)

{

var isValid = \_authService.VerifyOtp(otpDto.UserId, otpDto.Otp);

if (!isValid)

return Unauthorized(new { Message = "OTP نامعتبر است." });

return Ok(new { Message = "تأیید دومرحله‌ای با موفقیت انجام شد." });

}

}

```

---

#### \*\*2.2. Services (سرویس‌ها)\*\*

##### \*\*هدف:\*\*

سرویس‌ها شامل منطق کسب‌وکار برنامه هستند. آن‌ها کارهایی مثل اعتبارسنجی کاربران، تولید توکن‌های JWT، و مدیریت نقش‌ها را انجام می‌دهند.

##### \*\*مثال: IAuthService.cs\*\*

```csharp

public interface IAuthService

{

Task<ServiceResult<string>> RegisterAsync(UserDto userDto);

Task<ServiceResult<string>> LoginAsync(UserDto userDto);

Task<ServiceResult<string>> RefreshTokenAsync(string refreshToken);

bool VerifyOtp(string userId, string otp);

}

```

##### \*\*مثال: AuthService.cs\*\*

```csharp

public class AuthService : IAuthService

{

private readonly IUserRepository \_userRepository;

private readonly JwtTokenGenerator \_jwtTokenGenerator;

private readonly MfaService \_mfaService;

public AuthService(IUserRepository userRepository, JwtTokenGenerator jwtTokenGenerator, MfaService mfaService)

{

\_userRepository = userRepository;

\_jwtTokenGenerator = jwtTokenGenerator;

\_mfaService = mfaService;

}

public async Task<ServiceResult<string>> RegisterAsync(UserDto userDto)

{

// بررسی وجود کاربر

var userExists = await \_userRepository.GetUserByEmailAsync(userDto.Email);

if (userExists != null)

return ServiceResult<string>.Failure("کاربر قبلاً ثبت‌نام کرده است.");

// ایجاد کاربر جدید

var userEntity = new UserEntity

{

Email = userDto.Email,

PasswordHash = HashPassword(userDto.Password),

Role = userDto.Role

};

await \_userRepository.AddUserAsync(userEntity);

// تولید توکن

var token = \_jwtTokenGenerator.GenerateToken(userEntity);

return ServiceResult<string>.Success(token, "ثبت‌نام با موفقیت انجام شد.");

}

public async Task<ServiceResult<string>> LoginAsync(UserDto userDto)

{

var user = await \_userRepository.GetUserByEmailAsync(userDto.Email);

if (user == null || !VerifyPassword(userDto.Password, user.PasswordHash))

return ServiceResult<string>.Failure("ایمیل یا رمز عبور نادرست است.");

var token = \_jwtTokenGenerator.GenerateToken(user);

return ServiceResult<string>.Success(token, "ورود با موفقیت انجام شد.");

}

public async Task<ServiceResult<string>> RefreshTokenAsync(string refreshToken)

{

// اعتبارسنجی Refresh Token و تولید توکن جدید

var newToken = \_jwtTokenGenerator.GenerateNewToken(refreshToken);

return ServiceResult<string>.Success(newToken, "توکن جدید صادر شد.");

}

public bool VerifyOtp(string userId, string otp)

{

return \_mfaService.VerifyOtp(userId, otp);

}

private string HashPassword(string password)

{

return BCrypt.Net.BCrypt.HashPassword(password);

}

private bool VerifyPassword(string password, string hashedPassword)

{

return BCrypt.Net.BCrypt.Verify(password, hashedPassword);

}

}

```

---

#### \*\*2.3. Models (مدل‌ها)\*\*

##### \*\*هدف:\*\*

مدل‌ها شامل داده‌هایی هستند که در سراسر برنامه استفاده می‌شوند. آن‌ها به دو دسته تقسیم می‌شوند:

- \*\*DTO (Data Transfer Object)\*\*: برای انتقال داده‌ها بین لایه‌ها.

- \*\*Entities\*\*: برای ذخیره‌سازی داده‌ها در دیتابیس.

##### \*\*مثال: UserDto.cs\*\*

```csharp

public class UserDto

{

public string Email { get; set; }

public string Password { get; set; }

public UserRole Role { get; set; }

}

```

##### \*\*مثال: UserEntity.cs\*\*

```csharp

public class UserEntity

{

public Guid Id { get; set; }

public string Email { get; set; }

public string PasswordHash { get; set; }

public UserRole Role { get; set; }

public DateTime CreatedAt { get; set; }

public string RefreshToken { get; set; }

public DateTime? RefreshTokenExpiry { get; set; }

}

```

---

#### \*\*2.4. Enums (انواع شمارشی)\*\*

##### \*\*هدف:\*\*

استفاده از انواع شمارشی برای کنترل بهتر و امن‌تر مقادیر.

##### \*\*مثال: UserRole.cs\*\*

```csharp

public enum UserRole

{

Admin,

User,

Guest

}

```

---

#### \*\*2.5. Token (تولید توکن‌ها)\*\*

##### \*\*هدف:\*\*

این بخش شامل منطق تولید توکن‌های JWT است.

##### \*\*مثال: JwtTokenGenerator.cs\*\*

```csharp

public class JwtTokenGenerator

{

private readonly string \_secretKey;

public JwtTokenGenerator(IConfiguration configuration)

{

\_secretKey = configuration["Jwt:Secret"];

}

public string GenerateToken(UserEntity user)

{

var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_secretKey));

var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);

var claims = new[]

{

new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),

new Claim(ClaimTypes.Email, user.Email),

new Claim(ClaimTypes.Role, user.Role.ToString())

};

var token = new JwtSecurityToken(

issuer: "your-issuer",

audience: "your-audience",

claims: claims,

expires: DateTime.UtcNow.AddHours(1),

signingCredentials: credentials

);

return new JwtSecurityTokenHandler().WriteToken(token);

}

public string GenerateNewToken(string refreshToken)

{

// اعتبارسنجی Refresh Token و تولید توکن جدید

return "new-jwt-token";

}

}

```

---

#### \*\*2.6. Security (امنیت)\*\*

##### \*\*هدف:\*\*

این بخش شامل قابلیت‌های امنیتی مانند \*\*IP Tracking\*\* و \*\*MFA (Multi-Factor Authentication)\*\* است.

##### \*\*مثال: IPTrackingMiddleware.cs\*\*

```csharp

public class IPTrackingMiddleware

{

private readonly RequestDelegate \_next;

private static readonly ConcurrentDictionary<string, int> \_ipAttempts = new();

public IPTrackingMiddleware(RequestDelegate next)

{

\_next = next;

}

public async Task InvokeAsync(HttpContext context)

{

var ipAddress = context.Connection.RemoteIpAddress?.ToString();

if (string.IsNullOrEmpty(ipAddress))

{

context.Response.StatusCode = StatusCodes.Status403Forbidden;

return;

}

if (\_ipAttempts.ContainsKey(ipAddress) && \_ipAttempts[ipAddress] >= 5)

{

context.Response.StatusCode = StatusCodes.Status429TooManyRequests;

await context.Response.WriteAsync("Too many login attempts. Please try again later.");

return;

}

try

{

await \_next(context);

}

catch (UnauthorizedAccessException)

{

\_ipAttempts.AddOrUpdate(ipAddress, 1, (\_, count) => count + 1);

}

}

}

```

##### \*\*مثال: MfaService.cs\*\*

```csharp

public class MfaService

{

private readonly IUserRepository \_userRepository;

public MfaService(IUserRepository userRepository)

{

\_userRepository = userRepository;

}

public string GenerateOtp(string userId)

{

var otp = new Random().Next(100000, 999999).ToString();

\_userRepository.SaveOtp(userId, otp);

return otp;

}

public bool VerifyOtp(string userId, string otp)

{

var storedOtp = \_userRepository.GetOtp(userId);

return storedOtp == otp;

}

}

```

---

#### \*\*2.7. OpenIddict (ادغام با OpenIddict)\*\*

##### \*\*هدف:\*\*

استفاده از \*\*OpenIddict\*\* برای پشتیبانی از OAuth 2.0 و OpenID Connect.

##### \*\*مثال: OpenIddictConfiguration.cs\*\*

```csharp

public static class OpenIddictConfiguration

{

public static void AddOpenIddict(this IServiceCollection services)

{

services.AddOpenIddict()

.AddCore(options =>

{

options.UseEntityFrameworkCore()

.UseDbContext<AuthDbContext>();

})

.AddServer(options =>

{

options.SetAuthorizationEndpointUris("/connect/authorize")

.SetTokenEndpointUris("/connect/token")

.AllowPasswordFlow()

.AllowRefreshTokenFlow()

.UseAspNetCore()

.EnableTokenEndpointPassthrough();

options.DisableAccessTokenEncryption();

options.UseRollingTokens();

options.SetAccessTokenLifetime(TimeSpan.FromMinutes(30));

})

.AddValidation(options =>

{

options.UseLocalServer();

options.UseAspNetCore();

});

}

}

```

---

#### \*\*2.8. appsettings.json\*\*

##### \*\*هدف:\*\*

این فایل شامل تنظیمات اصلی برنامه است، مانند اتصال به دیتابیس و تنظیمات JWT.

##### \*\*مثال: appsettings.json\*\*

```json

{

"ConnectionStrings": {

"DefaultConnection": "Server=localhost;Database=AuthDb;Trusted\_Connection=True;"

},

"Jwt": {

"Secret": "your-secret-key",

"Issuer": "your-issuer",

"Audience": "your-audience"

}

}

```

**2.6. ادامه Security: MfaService**

**هدف:**

مدیریت احراز هویت دومرحله‌ای (Two-Factor Authentication) با استفاده از OTP (رمز یک‌بار مصرف).

کد کامل: MfaService.cs

public class MfaService

{

private readonly Dictionary<string, string> \_userOtps = new();

private readonly Dictionary<string, DateTime> \_otpExpiries = new();

private readonly TimeSpan \_otpValidity = TimeSpan.FromMinutes(5);

public string GenerateOtp(string userId)

{

var otp = new Random().Next(100000, 999999).ToString();

\_userOtps[userId] = otp;

\_otpExpiries[userId] = DateTime.UtcNow.Add(\_otpValidity);

// اینجا می‌تونی OTP رو از طریق SMS یا ایمیل ارسال کنی

Console.WriteLine($"OTP for {userId}: {otp}");

return otp;

}

public bool VerifyOtp(string userId, string otp)

{

if (!\_userOtps.ContainsKey(userId)) return false;

if (\_otpExpiries[userId] < DateTime.UtcNow)

{

\_userOtps.Remove(userId);

\_otpExpiries.Remove(userId);

return false;

}

var valid = \_userOtps[userId] == otp;

if (valid)

{

\_userOtps.Remove(userId);

\_otpExpiries.Remove(userId);

}

return valid;

}}

**2.7. OpenIddict Configuration (تنظیمات OAuth/OpenID)**

**هدف:**

راه‌اندازی OpenIddict برای پشتیبانی از احراز هویت استاندارد OAuth2/OpenID Connect در صورت نیاز.

**کد: OpenIddictConfiguration.cs**

public static class OpenIddictConfiguration

{

public static void Configure(IServiceCollection services)

{

services.AddOpenIddict()

.AddCore(options =>

{

options.UseEntityFrameworkCore()

.UseDbContext<ApplicationDbContext>();

})

.AddServer(options =>

{

options.SetTokenEndpointUris("/connect/token");

options.AllowPasswordFlow()

.AllowRefreshTokenFlow();

options.AcceptAnonymousClients();

options.AddEphemeralEncryptionKey()

.AddEphemeralSigningKey()

.DisableAccessTokenEncryption();

options.UseAspNetCore()

.EnableTokenEndpointPassthrough();

});

}

}

2.8. appsettings.json

{

"Jwt": {

"Secret": "your\_super\_secure\_secret\_key\_here\_should\_be\_long\_enough",

"Issuer": "your-issuer",

"Audience": "your-audience"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*"

}

**2.9. Program.cs**

var builder = WebApplication.CreateBuilder(args);

// اضافه کردن سرویس‌ها

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

// وابستگی‌ها

builder.Services.AddScoped<IUserRepository, UserRepository>();

builder.Services.AddScoped<IAuthService, AuthService>();

builder.Services.AddSingleton<JwtTokenGenerator>();

builder.Services.AddSingleton<MfaService>();

// فعال‌سازی Middlewareهای امنیتی

builder.Services.AddTransient<IPTrackingMiddleware>();

// اضافه کردن OpenIddict (در صورت نیاز)

OpenIddictConfiguration.Configure(builder.Services);

var app = builder.Build();

// استفاده از Middlewareها

app.UseMiddleware<IPTrackingMiddleware>();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

3. نتیجه‌گیری\*\*

ماژول **AuthService** حالا به‌صورت کامل طراحی شده و شامل موارد زیر هست:

* کنترلرها برای مدیریت API
* سرویس‌ها برای منطق احراز هویت
* تولید و اعتبارسنجی JWT
* پشتیبانی از MFA با OTP
* ردیابی IP برای جلوگیری از حملات Brute-force
* تعریف نقش‌ها با Enum
* قابلیت استفاده از OpenIddict برای OAuth/OpenID
* ساختار تمیز و قابل توسعه

ماژول \*\*AuthService\*\* یک ماژول کامل و مستقل است که می‌تواند به‌عنوان بخشی از یک سیستم ردیابی GPS استفاده شود. تمام بخش‌ها شامل کنترلرها، سرویس‌ها، مخازن، مدیریت دیتابیس، و امنیت به همراه کامنت‌های توضیحی نوشته شده‌اند.

### \*\*توضیح کامل ماژول SubscriptionService\*\*

ماژول \*\*SubscriptionService\*\* یک بخش کلیدی در سیستم مدیریت اشتراک‌ها (Subscriptions) است. این ماژول به کاربران امکان می‌دهد تا اشتراک‌های خود را مدیریت کنند، پرداخت انجام دهند، و وضعیت اشتراک‌هایشان را بررسی کنند. در ادامه، تمام بخش‌های این ماژول به‌صورت دقیق و ساختارمند توضیح داده شده‌اند.

---

## \*\*1. ساختار کلی ماژول\*\*

```plaintext

/SubscriptionService

│

├── Controllers

│ └── SubscriptionsController.cs

│

├── Services

│ ├── SubscriptionService.cs

│ └── ISubscriptionService.cs

│

├── Repositories

│ ├── SubscriptionRepository.cs

│ └── ISubscriptionRepository.cs

│

├── Models

│ ├── SubscriptionDto.cs

│ └── SubscriptionEntity.cs

│

├── Infrastructure

│ ├── Persistence

│ │ └── SubscriptionDbContext.cs

│ └── Payment

│ └── PaymentGateway.cs

│

├── Extensions

│ └── ServiceCollectionExtensions.cs

│

└── Program.cs

```

---

## \*\*2. توضیحات جامع برای هر بخش\*\*

---

### \*\*2.1. Program.cs\*\*

##### \*\*هدف:\*\*

این فایل برنامه را راه‌اندازی می‌کند و تنظیمات اولیه را اعمال می‌کند.

##### \*\*مثال: Program.cs\*\*

```csharp

var builder = WebApplication.CreateBuilder(args);

// افزودن سرویس‌های پیش‌فرض ASP.NET Core

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

// افزودن تنظیمات اختصاصی ماژول

builder.Services.AddSubscriptionServices(builder.Configuration);

var app = builder.Build();

// تنظیم خط لول درخواست‌ها

if (app.Environment.IsDevelopment())

{

app.UseSwagger(); // فعال کردن Swagger در حالت توسعه

app.UseSwaggerUI(); // فعال کردن UI Swagger

}

app.UseHttpsRedirection(); // هدایت به HTTPS

app.MapControllers(); // مپ کردن کنترلرها

app.Run(); // اجرای برنامه

```

---

### \*\*2.2. ServiceCollectionExtensions.cs\*\*

##### \*\*هدف:\*\*

این فایل شامل تنظیمات \*\*Dependency Injection\*\* برای ثبت سرویس‌ها و مخازن است.

##### \*\*مثال: ServiceCollectionExtensions.cs\*\*

```csharp

public static class ServiceCollectionExtensions

{

public static IServiceCollection AddSubscriptionServices(this IServiceCollection services, IConfiguration configuration)

{

// ثبت DbContext برای دسترسی به دیتابیس

services.AddDbContext<SubscriptionDbContext>(options =>

options.UseSqlServer(configuration.GetConnectionString("DefaultConnection")));

// ثبت مخازن

services.AddScoped<ISubscriptionRepository, SubscriptionRepository>();

// ثبت سرویس‌ها

services.AddScoped<ISubscriptionService, SubscriptionService>();

// ثبت درگاه پرداخت

services.AddSingleton<PaymentGateway>();

return services;

}

}

```

---

### \*\*2.3. Controllers (کنترلرها)\*\*

##### \*\*هدف:\*\*

کنترلرها واسط بین کلاینت‌ها (مثل وب‌اپلیکیشن یا موبایل) و سرویس‌های داخلی هستند. آن‌ها درخواست‌های HTTP را دریافت کرده و پاسخ مناسب ارسال می‌کنند.

##### \*\*مثال: SubscriptionsController.cs\*\*

```csharp

[ApiController]

[Route("api/[controller]")]

public class SubscriptionsController : ControllerBase

{

private readonly ISubscriptionService \_subscriptionService;

public SubscriptionsController(ISubscriptionService subscriptionService)

{

\_subscriptionService = subscriptionService;

}

/// <summary>

/// خرید یک اشتراک جدید

/// </summary>

[HttpPost("purchase")]

public async Task<IActionResult> PurchaseSubscription([FromBody] SubscriptionDto subscriptionDto)

{

var result = await \_subscriptionService.PurchaseSubscriptionAsync(subscriptionDto);

return Ok(result);

}

/// <summary>

/// دریافت اطلاعات اشتراک یک کاربر

/// </summary>

[HttpGet("{userId}")]

public async Task<IActionResult> GetUserSubscription(string userId)

{

var subscription = await \_subscriptionService.GetUserSubscriptionAsync(userId);

return Ok(subscription);

}

/// <summary>

/// تمدید اشتراک

/// </summary>

[HttpPut("{subscriptionId}/renew")]

public async Task<IActionResult> RenewSubscription(int subscriptionId)

{

var result = await \_subscriptionService.RenewSubscriptionAsync(subscriptionId);

return Ok(result);

}

/// <summary>

/// لغو اشتراک

/// </summary>

[HttpDelete("{subscriptionId}")]

public async Task<IActionResult> CancelSubscription(int subscriptionId)

{

await \_subscriptionService.CancelSubscriptionAsync(subscriptionId);

return Ok(new { Message = "اشتراک با موفقیت لغو شد." });

}

}

```

---

### \*\*2.4. Services (سرویس‌ها)\*\*

##### \*\*هدف:\*\*

سرویس‌ها شامل منطق کسب‌وکار برنامه هستند. آن‌ها کارهایی مثل خرید اشتراک، تمدید اشتراک، و مدیریت وضعیت اشتراک را انجام می‌دهند.

##### \*\*مثال: ISubscriptionService.cs\*\*

```csharp

public interface ISubscriptionService

{

Task<ServiceResult<SubscriptionDto>> PurchaseSubscriptionAsync(SubscriptionDto subscriptionDto);

Task<SubscriptionDto> GetUserSubscriptionAsync(string userId);

Task<ServiceResult<SubscriptionDto>> RenewSubscriptionAsync(int subscriptionId);

Task CancelSubscriptionAsync(int subscriptionId);

}

```

##### \*\*مثال: SubscriptionService.cs\*\*

```csharp

public class SubscriptionService : ISubscriptionService

{

private readonly ISubscriptionRepository \_subscriptionRepository;

private readonly PaymentGateway \_paymentGateway;

private readonly ILogger<SubscriptionService> \_logger;

public SubscriptionService(

ISubscriptionRepository subscriptionRepository,

PaymentGateway paymentGateway,

ILogger<SubscriptionService> logger)

{

\_subscriptionRepository = subscriptionRepository;

\_paymentGateway = paymentGateway;

\_logger = logger;

}

public async Task<ServiceResult<SubscriptionDto>> PurchaseSubscriptionAsync(SubscriptionDto subscriptionDto)

{

\_logger.LogInformation($"User {subscriptionDto.UserId} initiated a purchase for plan {subscriptionDto.PlanType}.");

var paymentResult = \_paymentGateway.ProcessPayment(subscriptionDto.UserId, subscriptionDto.Amount);

if (!paymentResult.Success)

{

\_logger.LogWarning($"Payment failed for user {subscriptionDto.UserId}.");

return ServiceResult<SubscriptionDto>.Failure(paymentResult.Message);

}

var subscriptionEntity = new SubscriptionEntity

{

UserId = subscriptionDto.UserId,

PlanType = subscriptionDto.PlanType,

StartDate = DateTime.UtcNow,

EndDate = DateTime.UtcNow.AddMonths(1),

Amount = subscriptionDto.Amount,

Status = SubscriptionStatus.Active

};

await \_subscriptionRepository.SaveSubscriptionAsync(subscriptionEntity);

\_logger.LogInformation($"User {subscriptionDto.UserId} successfully purchased a subscription.");

return ServiceResult<SubscriptionDto>.Success(MapToDto(subscriptionEntity));

}

public async Task<SubscriptionDto> GetUserSubscriptionAsync(string userId)

{

var subscriptionEntity = await \_subscriptionRepository.GetUserSubscriptionAsync(userId);

return MapToDto(subscriptionEntity);

}

public async Task<ServiceResult<SubscriptionDto>> RenewSubscriptionAsync(int subscriptionId)

{

var subscriptionEntity = await \_subscriptionRepository.GetSubscriptionByIdAsync(subscriptionId);

if (subscriptionEntity == null || subscriptionEntity.Status != SubscriptionStatus.Active)

return ServiceResult<SubscriptionDto>.Failure("اشتراک نامعتبر است.");

subscriptionEntity.EndDate = subscriptionEntity.EndDate.AddMonths(1);

await \_subscriptionRepository.UpdateSubscriptionAsync(subscriptionEntity);

\_logger.LogInformation($"Subscription {subscriptionId} renewed successfully.");

return ServiceResult<SubscriptionDto>.Success(MapToDto(subscriptionEntity));

}

public async Task CancelSubscriptionAsync(int subscriptionId)

{

var subscriptionEntity = await \_subscriptionRepository.GetSubscriptionByIdAsync(subscriptionId);

if (subscriptionEntity != null)

{

subscriptionEntity.Status = SubscriptionStatus.Canceled;

await \_subscriptionRepository.UpdateSubscriptionAsync(subscriptionEntity);

\_logger.LogInformation($"Subscription {subscriptionId} canceled by user.");

}

}

private SubscriptionDto MapToDto(SubscriptionEntity entity)

{

return new SubscriptionDto

{

Id = entity.Id,

UserId = entity.UserId,

PlanType = entity.PlanType,

StartDate = entity.StartDate,

EndDate = entity.EndDate,

Amount = entity.Amount,

Status = entity.Status

};

}

}

```

---

### \*\*2.5. Repositories (مخازن)\*\*

##### \*\*هدف:\*\*

مخازن مسئولیت تعامل مستقیم با دیتابیس را بر عهده دارند.

##### \*\*مثال: ISubscriptionRepository.cs\*\*

```csharp

public interface ISubscriptionRepository

{

Task SaveSubscriptionAsync(SubscriptionEntity subscription);

Task<SubscriptionEntity> GetUserSubscriptionAsync(string userId);

Task<SubscriptionEntity> GetSubscriptionByIdAsync(int subscriptionId);

Task UpdateSubscriptionAsync(SubscriptionEntity subscription);

}

```

##### \*\*مثال: SubscriptionRepository.cs\*\*

```csharp

public class SubscriptionRepository : ISubscriptionRepository

{

private readonly SubscriptionDbContext \_context;

public SubscriptionRepository(SubscriptionDbContext context)

{

\_context = context;

}

public async Task SaveSubscriptionAsync(SubscriptionEntity subscription)

{

\_context.Subscriptions.Add(subscription);

await \_context.SaveChangesAsync();

}

public async Task<SubscriptionEntity> GetUserSubscriptionAsync(string userId)

{

return await \_context.Subscriptions.FirstOrDefaultAsync(s => s.UserId == userId && s.Status == SubscriptionStatus.Active);

}

public async Task<SubscriptionEntity> GetSubscriptionByIdAsync(int subscriptionId)

{

return await \_context.Subscriptions.FindAsync(subscriptionId);

}

public async Task UpdateSubscriptionAsync(SubscriptionEntity subscription)

{

\_context.Subscriptions.Update(subscription);

await \_context.SaveChangesAsync();

}

}

```

---

### \*\*2.6. Models (مدل‌ها)\*\*

##### \*\*هدف:\*\*

مدل‌ها شامل داده‌هایی هستند که در سراسر برنامه استفاده می‌شوند.

##### \*\*مثال: SubscriptionDto.cs\*\*

```csharp

public class SubscriptionDto

{

public int Id { get; set; }

public string UserId { get; set; }

public string PlanType { get; set; }

public DateTime StartDate { get; set; }

public DateTime EndDate { get; set; }

public decimal Amount { get; set; }

public SubscriptionStatus Status { get; set; }

}

```

##### \*\*مثال: SubscriptionEntity.cs\*\*

```csharp

public class SubscriptionEntity

{

public int Id { get; set; }

public string UserId { get; set; }

public string PlanType { get; set; }

public DateTime StartDate { get; set; }

public DateTime EndDate { get; set; }

public decimal Amount { get; set; }

public SubscriptionStatus Status { get; set; }

}

```

##### \*\*مثال: SubscriptionStatus.cs\*\*

```csharp

public enum SubscriptionStatus

{

Active, // اشتراک فعال

Canceled, // اشتراک لغو شده

Expired // اشتراک منقضی شده

}

```

---

### \*\*2.7. Persistence (مدیریت دیتابیس)\*\*

##### \*\*هدف:\*\*

این بخش شامل تنظیمات مربوط به دیتابیس و مدل‌های موجودیت است.

##### \*\*مثال: SubscriptionDbContext.cs\*\*

```csharp

public class SubscriptionDbContext : DbContext

{

public SubscriptionDbContext(DbContextOptions<SubscriptionDbContext> options) : base(options) { }

public DbSet<SubscriptionEntity> Subscriptions { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<SubscriptionEntity>()

.HasKey(s => s.Id);

modelBuilder.Entity<SubscriptionEntity>()

.Property(s => s.Status)

.HasConversion<string>(); // ذخیره Enum به‌صورت رشته

}

}

```

---

### \*\*2.8. Payment (درگاه پرداخت)\*\*

##### \*\*هدف:\*\*

این بخش شامل منطق پرداخت است.

##### \*\*مثال: PaymentGateway.cs\*\*

```csharp

public class PaymentGateway

{

public PaymentResult ProcessPayment(string userId, decimal amount)

{

// شبیه‌سازی پرداخت

bool success = true; // در صورت استفاده از یک درگاه واقعی، اینجا پرداخت انجام می‌شود

return new PaymentResult

{

Success = success,

Message = success ? "پرداخت موفقیت‌آمیز بود." : "خطا در پرداخت."

};

}

}

public class PaymentResult

{

public bool Success { get; set; }

public string Message { get; set; }

}

افزودن Enum برای وضعیت اشتراک‌ها (SubscriptionStatus)

public enum SubscriptionStatus

{

Pending,

Active,

Expired,

Canceled

}

**افزودن ServiceResult**

public class ServiceResult<T>

{

public bool Success { get; set; }

public string Message { get; set; }

public T Data { get; set; }

public static ServiceResult<T> Success(T data, string message = "") =>

new ServiceResult<T> { Success = true, Data = data, Message = message };

public static ServiceResult<T> Failure(string message) =>

new ServiceResult<T> { Success = false, Message = message };

}

SubscriptionEntity.cs

public class SubscriptionEntity

{

[Key]

public int Id { get; set; }

[Required]

public string UserId { get; set; }

[Required]

[MaxLength(50)]

public string PlanType { get; set; }

[Required]

public DateTime StartDate { get; set; }

[Required]

public DateTime EndDate { get; set; }

[Required]

[Column(TypeName = "decimal(18,2)")]

public decimal Amount { get; set; }

[Required]

public SubscriptionStatus Status { get; set; }

public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

public DateTime? UpdatedAt { get; set; }

}

\*\*3. نتیجه‌گیری\*\*

ماژول \*\*SubscriptionService\*\* یک ماژول کامل و مستقل است که می‌تواند به‌عنوان بخشی از یک سیستم ردیابی GPS استفاده شود. تمام بخش‌ها شامل کنترلرها، سرویس‌ها، مخازن، مدیریت دیتابیس، و مدیریت پرداخت‌ها به همراه کامنت‌های توضیحی نوشته شده‌اند

### \*\*توضیح کامل ماژول BillingService\*\*

ماژول \*\*BillingService\*\* یک بخش کلیدی در سیستم مدیریت صورتحساب‌ها (Billing) است. این ماژول به کاربران امکان می‌دهد تا صورتحساب‌های خود را مدیریت کنند، پرداخت‌ها را ثبت کنند و وضعیت مالی خود را بررسی کنند. در ادامه، تمام بخش‌های این ماژول به‌صورت دقیق و ساختارمند توضیح داده شده‌اند.

---

## \*\*1. ساختار کلی ماژول\*\*

```plaintext

/BillingService

│

├── Controllers

│ └── BillingController.cs

│

├── Services

│ ├── BillingService.cs

│ └── IBillingService.cs

│

├── Repositories

│ ├── BillingRepository.cs

│ └── IBillingRepository.cs

│

├── Models

│ ├── BillingDto.cs

│ └── BillingEntity.cs

│

├── Persistence

│ ├── BillingDbContext.cs

│ └── EntityConfigurations

│ └── BillingConfiguration.cs

│

├── appsettings.json

│

└── Program.cs

```

---

## \*\*2. توضیحات جامع برای هر بخش\*\*

---

### \*\*2.1. Program.cs\*\*

##### \*\*هدف:\*\*

این فایل برنامه را راه‌اندازی می‌کند و تنظیمات اولیه را اعمال می‌کند.

##### \*\*مثال: Program.cs\*\*

```csharp

var builder = WebApplication.CreateBuilder(args);

// افزودن سرویس‌های پیش‌فرض ASP.NET Core

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

// افزودن تنظیمات اختصاصی ماژول

builder.Services.AddBillingServices(builder.Configuration);

var app = builder.Build();

// تنظیم خط لول درخواست‌ها

if (app.Environment.IsDevelopment())

{

app.UseSwagger(); // فعال کردن Swagger در حالت توسعه

app.UseSwaggerUI(); // فعال کردن UI Swagger

}

app.UseHttpsRedirection(); // هدایت به HTTPS

app.MapControllers(); // مپ کردن کنترلرها

app.Run(); // اجرای برنامه

```

---

### \*\*2.2. ServiceCollectionExtensions.cs\*\*

##### \*\*هدف:\*\*

این فایل شامل تنظیمات \*\*Dependency Injection\*\* برای ثبت سرویس‌ها و مخازن است.

##### \*\*مثال: ServiceCollectionExtensions.cs\*\*

```csharp

public static class ServiceCollectionExtensions

{

public static IServiceCollection AddBillingServices(this IServiceCollection services, IConfiguration configuration)

{

// ثبت DbContext برای دسترسی به دیتابیس

services.AddDbContext<BillingDbContext>(options =>

options.UseSqlServer(configuration.GetConnectionString("DefaultConnection")));

// ثبت مخازن

services.AddScoped<IBillingRepository, BillingRepository>();

// ثبت سرویس‌ها

services.AddScoped<IBillingService, BillingService>();

return services;

}

}

```

---

### \*\*2.3. Controllers (کنترلرها)\*\*

##### \*\*هدف:\*\*

کنترلرها واسط بین کلاینت‌ها (مثل وب‌اپلیکیشن یا موبایل) و سرویس‌های داخلی هستند. آن‌ها درخواست‌های HTTP را دریافت کرده و پاسخ مناسب ارسال می‌کنند.

##### \*\*مثال: BillingController.cs\*\*

```csharp

[ApiController]

[Route("api/[controller]")]

public class BillingController : ControllerBase

{

private readonly IBillingService \_billingService;

public BillingController(IBillingService billingService)

{

\_billingService = billingService;

}

/// <summary>

/// ثبت یک صورتحساب جدید

/// </summary>

[HttpPost("create")]

public async Task<IActionResult> CreateBill([FromBody] BillingDto billingDto)

{

var result = await \_billingService.CreateBillAsync(billingDto);

return Ok(result);

}

/// <summary>

/// دریافت صورتحساب‌های یک کاربر

/// </summary>

[HttpGet("{userId}")]

public async Task<IActionResult> GetUserBills(string userId)

{

var bills = await \_billingService.GetUserBillsAsync(userId);

return Ok(bills);

}

/// <summary>

/// پرداخت یک صورتحساب

/// </summary>

[HttpPut("{billId}/pay")]

public async Task<IActionResult> PayBill(int billId)

{

var result = await \_billingService.PayBillAsync(billId);

return Ok(result);

}

/// <summary>

/// لغو یک صورتحساب

/// </summary>

[HttpDelete("{billId}")]

public async Task<IActionResult> CancelBill(int billId)

{

await \_billingService.CancelBillAsync(billId);

return Ok(new { Message = "صورتحساب با موفقیت لغو شد." });

}

}

```

---

### \*\*2.4. Services (سرویس‌ها)\*\*

##### \*\*هدف:\*\*

سرویس‌ها شامل منطق کسب‌وکار برنامه هستند. آن‌ها کارهایی مثل ایجاد صورتحساب، پرداخت صورتحساب، و مدیریت وضعیت صورتحساب را انجام می‌دهند.

##### \*\*مثال: IBillingService.cs\*\*

```csharp

public interface IBillingService

{

Task<ServiceResult<BillingDto>> CreateBillAsync(BillingDto billingDto);

Task<List<BillingDto>> GetUserBillsAsync(string userId);

Task<ServiceResult<BillingDto>> PayBillAsync(int billId);

Task CancelBillAsync(int billId);

}

```

##### \*\*مثال: BillingService.cs\*\*

```csharp

public class BillingService : IBillingService

{

private readonly IBillingRepository \_billingRepository;

private readonly PaymentGateway \_paymentGateway;

private readonly ILogger<BillingService> \_logger;

public BillingService(

IBillingRepository billingRepository,

PaymentGateway paymentGateway,

ILogger<BillingService> logger)

{

\_billingRepository = billingRepository;

\_paymentGateway = paymentGateway;

\_logger = logger;

}

public async Task<ServiceResult<BillingDto>> CreateBillAsync(BillingDto billingDto)

{

\_logger.LogInformation($"Creating a new bill for user {billingDto.UserId}.");

var billEntity = new BillingEntity

{

UserId = billingDto.UserId,

Amount = billingDto.Amount,

Description = billingDto.Description,

Status = BillingStatus.Pending,

CreatedAt = DateTime.UtcNow

};

await \_billingRepository.SaveBillAsync(billEntity);

\_logger.LogInformation($"Bill {billEntity.Id} created successfully.");

return ServiceResult<BillingDto>.Success(MapToDto(billEntity));

}

public async Task<List<BillingDto>> GetUserBillsAsync(string userId)

{

var billEntities = await \_billingRepository.GetUserBillsAsync(userId);

return billEntities.Select(MapToDto).ToList();

}

public async Task<ServiceResult<BillingDto>> PayBillAsync(int billId)

{

var billEntity = await \_billingRepository.GetBillByIdAsync(billId);

if (billEntity == null || billEntity.Status != BillingStatus.Pending)

return ServiceResult<BillingDto>.Failure("صورتحساب نامعتبر است.");

var paymentResult = \_paymentGateway.ProcessPayment(billEntity.UserId, billEntity.Amount);

if (!paymentResult.Success)

return ServiceResult<BillingDto>.Failure(paymentResult.Message);

billEntity.Status = BillingStatus.Paid;

billEntity.PaidAt = DateTime.UtcNow;

await \_billingRepository.UpdateBillAsync(billEntity);

\_logger.LogInformation($"Bill {billId} paid successfully.");

return ServiceResult<BillingDto>.Success(MapToDto(billEntity));

}

public async Task CancelBillAsync(int billId)

{

var billEntity = await \_billingRepository.GetBillByIdAsync(billId);

if (billEntity != null)

{

billEntity.Status = BillingStatus.Canceled;

await \_billingRepository.UpdateBillAsync(billEntity);

\_logger.LogInformation($"Bill {billId} canceled by user.");

}

}

private BillingDto MapToDto(BillingEntity entity)

{

return new BillingDto

{

Id = entity.Id,

UserId = entity.UserId,

Amount = entity.Amount,

Description = entity.Description,

Status = entity.Status,

CreatedAt = entity.CreatedAt,

PaidAt = entity.PaidAt

};

}

}

```

---

### \*\*2.5. Repositories (مخازن)\*\*

##### \*\*هدف:\*\*

مخازن مسئولیت تعامل مستقیم با دیتابیس را بر عهده دارند.

##### \*\*مثال: IBillingRepository.cs\*\*

```csharp

public interface IBillingRepository

{

Task SaveBillAsync(BillingEntity bill);

Task<List<BillingEntity>> GetUserBillsAsync(string userId);

Task<BillingEntity> GetBillByIdAsync(int billId);

Task UpdateBillAsync(BillingEntity bill);

}

```

##### \*\*مثال: BillingRepository.cs\*\*

```csharp

public class BillingRepository : IBillingRepository

{

private readonly BillingDbContext \_context;

public BillingRepository(BillingDbContext context)

{

\_context = context;

}

public async Task SaveBillAsync(BillingEntity bill)

{

\_context.Bills.Add(bill);

await \_context.SaveChangesAsync();

}

public async Task<List<BillingEntity>> GetUserBillsAsync(string userId)

{

return await \_context.Bills.Where(b => b.UserId == userId).ToListAsync();

}

public async Task<BillingEntity> GetBillByIdAsync(int billId)

{

return await \_context.Bills.FindAsync(billId);

}

public async Task UpdateBillAsync(BillingEntity bill)

{

\_context.Bills.Update(bill);

await \_context.SaveChangesAsync();

}

}

```

---

### \*\*2.6. Models (مدل‌ها)\*\*

##### \*\*هدف:\*\*

مدل‌ها شامل داده‌هایی هستند که در سراسر برنامه استفاده می‌شوند.

##### \*\*مثال: BillingDto.cs\*\*

```csharp

public class BillingDto

{

public int Id { get; set; }

public string UserId { get; set; }

public decimal Amount { get; set; }

public string Description { get; set; }

public BillingStatus Status { get; set; }

public DateTime CreatedAt { get; set; }

public DateTime? PaidAt { get; set; }

}

```

##### \*\*مثال: BillingEntity.cs\*\*

```csharp

public class BillingEntity

{

public int Id { get; set; }

public string UserId { get; set; }

public decimal Amount { get; set; }

public string Description { get; set; }

public BillingStatus Status { get; set; }

public DateTime CreatedAt { get; set; }

public DateTime? PaidAt { get; set; }

}

```

##### \*\*مثال: BillingStatus.cs\*\*

```csharp

public enum BillingStatus

{

Pending, // در انتظار پرداخت

Paid, // پرداخت شده

Canceled // لغو شده

}

```

---

### \*\*2.7. Persistence (مدیریت دیتابیس)\*\*

##### \*\*هدف:\*\*

این بخش شامل تنظیمات مربوط به دیتابیس و مدل‌های موجودیت است.

##### \*\*مثال: BillingDbContext.cs\*\*

```csharp

public class BillingDbContext : DbContext

{

public BillingDbContext(DbContextOptions<BillingDbContext> options) : base(options) { }

public DbSet<BillingEntity> Bills { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<BillingEntity>()

.HasKey(b => b.Id);

modelBuilder.Entity<BillingEntity>()

.Property(b => b.Status)

.HasConversion<string>(); // ذخیره Enum به‌صورت رشته

}

}

```

---

### \*\*2.8. appsettings.json\*\*

##### \*\*هدف:\*\*

این فایل شامل تنظیمات اصلی برنامه است، مانند اتصال به دیتابیس و تنظیمات درگاه پرداخت.

##### \*\*مثال: appsettings.json\*\*

```json

{

"ConnectionStrings": {

"DefaultConnection": "Server=localhost;Database=BillingDb;Trusted\_Connection=True;"

},

"PaymentGateway": {

"ApiKey": "your-payment-gateway-api-key"

}

}

**تکمیل BillingEntity.cs**

public class BillingEntity

{

public int Id { get; set; }

public string UserId { get; set; }

public decimal Amount { get; set; }

public string Description { get; set; }

public BillingStatus Status { get; set; }

public DateTime CreatedAt { get; set; }

public DateTime? PaidAt { get; set; }

// ارتباط با Subscription (در صورتی که صورتحساب مربوط به اشتراک باشد)

public int? SubscriptionId { get; set; }

public SubscriptionEntity? Subscription { get; set; }

}

تعریف کلاس SubscriptionEntity.cs

public class SubscriptionEntity

{

public int Id { get; set; }

public string UserId { get; set; }

public string PlanName { get; set; }

public decimal Price { get; set; }

public DateTime StartDate { get; set; }

public DateTime EndDate { get; set; }

public bool IsActive { get; set; }

// ارتباط با صورتحساب‌ها

public List<BillingEntity> Bills { get; set; } = new();

}

**BillingStatus Enum**

public enum BillingStatus

{

Pending = 0,

Paid = 1,

Canceled = 2

}

**پیکربندی Entityها در BillingConfiguration.cs**

public class SubscriptionConfiguration : IEntityTypeConfiguration<SubscriptionEntity>

{

public void Configure(EntityTypeBuilder<SubscriptionEntity> builder)

{

builder.HasKey(s => s.Id);

builder.Property(s => s.Price).HasColumnType("decimal(18,2)");

}

}

public class SubscriptionConfiguration : IEntityTypeConfiguration<SubscriptionEntity>

{

public void Configure(EntityTypeBuilder<SubscriptionEntity> builder)

{

builder.HasKey(s => s.Id);

builder.Property(s => s.Price).HasColumnType("decimal(18,2)");

}

}

افزودن DbSet در BillingDbContext

public class BillingDbContext : DbContext

{

public BillingDbContext(DbContextOptions<BillingDbContext> options) : base(options) { }

public DbSet<BillingEntity> Bills { get; set; }

public DbSet<SubscriptionEntity> Subscriptions { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.ApplyConfiguration(new BillingConfiguration());

modelBuilder.ApplyConfiguration(new SubscriptionConfiguration());

}

}

### \*\*3. نتیجه‌گیری\*\*

ماژول \*\*BillingService\*\* یک ماژول کامل و مستقل است که می‌تواند به‌عنوان بخشی از یک سیستم ردیابی GPS استفاده شود. تمام بخش‌ها شامل کنترلرها، سرویس‌ها، مخازن، مدیریت دیتابیس، و مدیریت پرداخت‌ها به همراه کامنت‌های توضیحی نوشته شده‌اند.

### \*\*توضیح کامل ماژول EventBus\*\*

ماژول \*\*EventBus\*\* یک بخش کلیدی در سیستم‌های مبتنی بر معماری \*\*Microservices\*\* است. این ماژول به مدیریت و پردازش رویدادها (Events) بین میکروسرویس‌ها می‌پردازد. با استفاده از این ماژول، می‌توان رویدادها را به‌صورت غیرهمزمان منتشر کرد و مصرف کرد. در ادامه، تمام بخش‌های این ماژول به‌صورت دقیق و ساختارمند توضیح داده شده‌اند.

---

## \*\*1. ساختار کلی ماژول\*\*

```plaintext

/EventBus

│

├── IEventPublisher.cs

├── IEventSubscriber.cs

├── KafkaEventBus.cs

├── Outbox

│ ├── OutboxMessageEntity.cs

│ └── OutboxPublisherService.cs

│

├── appsettings.json

└── Program.cs

```

---

## \*\*2. توضیحات جامع برای هر بخش\*\*

---

### \*\*2.1. IEventPublisher.cs\*\*

##### \*\*هدف:\*\*

این فایل شامل اینترفیسی است که تعیین می‌کند چگونه یک رویداد منتشر شود. این اینترفیس به‌عنوان قرارداد برای کلاس‌هایی است که مسئول انتشار رویدادها هستند.

##### \*\*مثال: IEventPublisher.cs\*\*

```csharp

public interface IEventPublisher

{

Task PublishAsync<T>(T @event) where T : class;

}

```

---

### \*\*2.2. IEventSubscriber.cs\*\*

##### \*\*هدف:\*\*

این فایل شامل اینترفیسی است که تعیین می‌کند چگونه یک رویداد مصرف شود. این اینترفیس به‌عنوان قرارداد برای کلاس‌هایی است که مسئول مصرف رویدادها هستند.

##### \*\*مثال: IEventSubscriber.cs\*\*

```csharp

public interface IEventSubscriber

{

Task SubscribeAsync<T>(Func<T, Task> handler) where T : class;

}

```

---

### \*\*2.3. KafkaEventBus.cs\*\*

##### \*\*هدف:\*\*

این فایل شامل پیاده‌سازی موتور رویدادها (Event Bus) است که از \*\*Apache Kafka\*\* برای مدیریت رویدادها استفاده می‌کند. این کلاس هم به‌عنوان منتشرکننده (Publisher) و هم به‌عنوان مصرف‌کننده (Subscriber) عمل می‌کند.

##### \*\*مثال: KafkaEventBus.cs\*\*

```csharp

public class KafkaEventBus : IEventPublisher, IEventSubscriber

{

private readonly IProducer<Null, string> \_producer;

private readonly IConsumer<Null, string> \_consumer;

public KafkaEventBus(string kafkaBootstrapServers)

{

var producerConfig = new ProducerConfig { BootstrapServers = kafkaBootstrapServers };

\_producer = new ProducerBuilder<Null, string>(producerConfig).Build();

var consumerConfig = new ConsumerConfig

{

BootstrapServers = kafkaBootstrapServers,

GroupId = "event-bus-group",

AutoOffsetReset = AutoOffsetReset.Earliest

};

\_consumer = new ConsumerBuilder<Null, string>(consumerConfig).Build();

}

public async Task PublishAsync<T>(T @event) where T : class

{

var topic = typeof(T).Name; // نام تاپیک بر اساس نوع رویداد

var message = JsonSerializer.Serialize(@event);

await \_producer.ProduceAsync(topic, new Message<Null, string> { Value = message });

}

public Task SubscribeAsync<T>(Func<T, Task> handler) where T : class

{

var topic = typeof(T).Name; // نام تاپیک بر اساس نوع رویداد

\_consumer.Subscribe(topic);

Task.Run(async () =>

{

while (true)

{

var consumeResult = \_consumer.Consume();

var @event = JsonSerializer.Deserialize<T>(consumeResult.Message.Value);

await handler(@event);

}

});

return Task.CompletedTask;

}

}

```

---

### \*\*2.4. Outbox (Pattern)\*\*

#### \*\*هدف:\*\*

برای مدیریت رویدادها در سیستم‌های توزیع‌شده، از الگوی \*\*Outbox\*\* استفاده می‌شود. این الگو اطمینان می‌دهد که رویدادها در صورت بروز مشکل در ارسال، از دست نروند و بعداً مجدداً ارسال شوند.

---

#### \*\*2.4.1. OutboxMessageEntity.cs\*\*

##### \*\*هدف:\*\*

این فایل شامل مدلی است که رویدادهای منتظر ارسال را در دیتابیس ذخیره می‌کند.

##### \*\*مثال: OutboxMessageEntity.cs\*\*

```csharp

public class OutboxMessageEntity

{

public Guid Id { get; set; }

public string EventType { get; set; } // نوع رویداد

public string Payload { get; set; } // داده‌های رویداد

public DateTime CreatedAt { get; set; } // زمان ایجاد

public DateTime? PublishedAt { get; set; } // زمان ارسال

}

```

---

#### \*\*2.4.2. OutboxPublisherService.cs\*\*

##### \*\*هدف:\*\*

این سرویس مسئولیت ارسال رویدادهای موجود در جدول \*\*Outbox\*\* را بر عهده دارد. این سرویس رویدادها را به‌صورت دوره‌ای بررسی کرده و آن‌ها را منتشر می‌کند.

##### \*\*مثال: OutboxPublisherService.cs\*\*

```csharp

public class OutboxPublisherService

{

private readonly ApplicationDbContext \_context;

private readonly IEventPublisher \_eventPublisher;

public OutboxPublisherService(ApplicationDbContext context, IEventPublisher eventPublisher)

{

\_context = context;

\_eventPublisher = eventPublisher;

}

public async Task ProcessOutboxMessagesAsync()

{

var pendingMessages = await \_context.OutboxMessages

.Where(m => m.PublishedAt == null)

.ToListAsync();

foreach (var message in pendingMessages)

{

try

{

var eventType = Type.GetType(message.EventType);

var @event = JsonSerializer.Deserialize(message.Payload, eventType);

await \_eventPublisher.PublishAsync(@event);

message.PublishedAt = DateTime.UtcNow;

}

catch

{

// در صورت خطا، می‌توانید لاگ‌نویسی کنید یا اقدامات دیگری انجام دهید

}

}

await \_context.SaveChangesAsync();

}

}

```

---

### \*\*2.5. appsettings.json\*\*

##### \*\*هدف:\*\*

این فایل شامل تنظیمات اصلی برنامه است، مانند آدرس \*\*Kafka Broker\*\*.

##### \*\*مثال: appsettings.json\*\*

```json

{

"Kafka": {

"BootstrapServers": "localhost:9092"

},

"ConnectionStrings": {

"DefaultConnection": "Server=localhost;Database=EventBusDb;Trusted\_Connection=True;"

}

}

```

---

### \*\*2.6. Program.cs\*\*

##### \*\*هدف:\*\*

این فایل برنامه را راه‌اندازی می‌کند و تنظیمات اولیه را اعمال می‌کند.

##### \*\*مثال: Program.cs\*\*

```csharp

var builder = WebApplication.CreateBuilder(args);

// افزودن سرویس‌های پیش‌فرض ASP.NET Core

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

// افزودن تنظیمات اختصاصی ماژول

builder.Services.AddSingleton<IEventPublisher, KafkaEventBus>();

builder.Services.AddSingleton<IEventSubscriber, KafkaEventBus>();

builder.Services.AddScoped<OutboxPublisherService>();

// افزودن DbContext برای دسترسی به دیتابیس

builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

var app = builder.Build();

// تنظیم خط لول درخواست‌ها

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.MapControllers();

// اجرای سرویس Outbox Publisher به‌صورت دوره‌ای

var outboxPublisher = app.Services.GetRequiredService<OutboxPublisherService>();

await outboxPublisher.ProcessOutboxMessagesAsync();

app.Run();

پیکربندی امنیتی Kafka:

برای فعال‌سازی رمزنگاری TLS در Kafka، می‌توان تنظیمات زیر را در فایل پیکربندی Kafka قرار داد:

"security.protocol": "SSL",

"ssl.keystore.location": "/path/to/keystore.jks",

"ssl.keystore.password": "password",

"ssl.key.password": "password",

"ssl.truststore.location": "/path/to/truststore.jks",

"ssl.truststore.password": "password"

**استفاده از Partitioning در Kafka:** با استفاده از **Partitioning** در Kafka، می‌توان بار را به‌طور مساوی بین چندین مصرف‌کننده توزیع کرد. این روش باعث افزایش مقیاس‌پذیری و کارایی می‌شود.

3. نتیجه‌گیری\*\*

ماژول \*\*EventBus\*\* یک ماژول کامل و مستقل است که می‌تواند به‌عنوان بخشی از یک سیستم ردیابی GPS استفاده شود. تمام بخش‌ها شامل اینترفیس‌ها، پیاده‌سازی‌ها، مدیریت دیتابیس، و ارسال و دریافت رویدادها به همراه کامنت‌های توضیحی نوشته شده‌اند.

### \*\*توضیح کامل ماژول EventStoreService\*\*

ماژول \*\*EventStoreService\*\* یک بخش کلیدی در سیستم‌های مبتنی بر معماری \*\*Event Sourcing\*\* است. این ماژول به ذخیره‌سازی و بازیابی رویدادها (Events) می‌پردازد و به‌عنوان منبع اصلی داده‌ها عمل می‌کند. در ادامه، تمام بخش‌های این ماژول به‌صورت دقیق و ساختارمند توضیح داده شده‌اند.

---

## \*\*1. ساختار کلی ماژول\*\*

```plaintext

/EventStoreService

│

├── Controllers

│ └── EventsController.cs

│

├── Services

│ ├── EventStoreService.cs

│ └── IEventStoreService.cs

│

├── Repositories

│ ├── EventRepository.cs

│ └── IEventRepository.cs

│

├── Models

│ ├── EventDto.cs

│ └── EventEntity.cs

│

├── Persistence

│ ├── EventDbContext.cs

│ └── EntityConfigurations

│ └── EventConfiguration.cs

│

├── Streaming

│ └── StreamProcessor.cs

│

├── appsettings.json

│

└── Program.cs

```

---

## \*\*2. توضیحات جامع برای هر بخش\*\*

---

### \*\*2.1. Program.cs\*\*

##### \*\*هدف:\*\*

این فایل برنامه را راه‌اندازی می‌کند و تنظیمات اولیه را اعمال می‌کند.

##### \*\*مثال: Program.cs\*\*

```csharp

var builder = WebApplication.CreateBuilder(args);

// افزودن سرویس‌های پیش‌فرض ASP.NET Core

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

// افزودن تنظیمات اختصاصی ماژول

builder.Services.AddEventStoreServices(builder.Configuration);

var app = builder.Build();

// تنظیم خط لول درخواست‌ها

if (app.Environment.IsDevelopment())

{

app.UseSwagger(); // فعال کردن Swagger در حالت توسعه

app.UseSwaggerUI(); // فعال کردن UI Swagger

}

app.UseHttpsRedirection(); // هدایت به HTTPS

app.MapControllers(); // مپ کردن کنترلرها

app.Run(); // اجرای برنامه

```

---

### \*\*2.2. ServiceCollectionExtensions.cs\*\*

##### \*\*هدف:\*\*

این فایل شامل تنظیمات \*\*Dependency Injection\*\* برای ثبت سرویس‌ها و مخازن است.

##### \*\*مثال: ServiceCollectionExtensions.cs\*\*

```csharp

public static class ServiceCollectionExtensions

{

public static IServiceCollection AddEventStoreServices(this IServiceCollection services, IConfiguration configuration)

{

// ثبت DbContext برای دسترسی به دیتابیس

services.AddDbContext<EventDbContext>(options =>

options.UseSqlServer(configuration.GetConnectionString("DefaultConnection")));

// ثبت مخازن

services.AddScoped<IEventRepository, EventRepository>();

// ثبت سرویس‌ها

services.AddScoped<IEventStoreService, EventStoreService>();

return services;

}

}

```

---

### \*\*2.3. Controllers (کنترلرها)\*\*

##### \*\*هدف:\*\*

کنترلرها واسط بین کلاینت‌ها (مثل وب‌اپلیکیشن یا موبایل) و سرویس‌های داخلی هستند. آن‌ها درخواست‌های HTTP را دریافت کرده و پاسخ مناسب ارسال می‌کنند.

##### \*\*مثال: EventsController.cs\*\*

```csharp

[ApiController]

[Route("api/[controller]")]

public class EventsController : ControllerBase

{

private readonly IEventStoreService \_eventStoreService;

public EventsController(IEventStoreService eventStoreService)

{

\_eventStoreService = eventStoreService;

}

/// <summary>

/// ذخیره یک رویداد جدید

/// </summary>

[HttpPost("save")]

public async Task<IActionResult> SaveEvent([FromBody] EventDto eventDto)

{

try

{

await \_eventStoreService.SaveEventAsync(eventDto);

return Ok(new { Message = "رویداد با موفقیت ذخیره شد." });

}

catch (Exception ex)

{

// لاگ‌گیری از خطای رخ‌داده

Console.Error.WriteLine($"خطا در ذخیره رویداد: {ex.Message}");

return StatusCode(500, new { Message = "خطایی در ذخیره رویداد رخ داد." });

}

}

/// <summary>

/// دریافت رویدادهای یک جریان خاص

/// </summary>

[HttpGet("stream/{streamId}")]

public async Task<IActionResult> GetEventsByStreamId(Guid streamId)

{

var events = await \_eventStoreService.GetEventsByStreamIdAsync(streamId);

return Ok(events);

}

}

```

---

### \*\*2.4. Services (سرویس‌ها)\*\*

##### \*\*هدف:\*\*

سرویس‌ها شامل منطق کسب‌وکار برنامه هستند. آن‌ها کارهایی مثل ذخیره‌سازی رویدادها، بازیابی رویدادها، و مدیریت جریان‌ها (Streams) را انجام می‌دهند.

##### \*\*مثال: IEventStoreService.cs\*\*

```csharp

public interface IEventStoreService

{

Task SaveEventAsync(EventDto eventDto);

Task<List<EventDto>> GetEventsByStreamIdAsync(Guid streamId);

}

```

##### \*\*مثال: EventStoreService.cs\*\*

```csharp

public class EventStoreService : IEventStoreService

{

private readonly IEventRepository \_eventRepository;

public EventStoreService(IEventRepository eventRepository)

{

\_eventRepository = eventRepository;

}

public async Task SaveEventAsync(EventDto eventDto)

{

var eventEntity = new EventEntity

{

StreamId = eventDto.StreamId,

Type = eventDto.Type,

Data = eventDto.Data,

Timestamp = DateTime.UtcNow

};

await \_eventRepository.SaveEventAsync(eventEntity);

}

public async Task<List<EventDto>> GetEventsByStreamIdAsync(Guid streamId)

{

var events = await \_eventRepository.GetEventsByStreamIdAsync(streamId);

return events.Select(MapToDto).ToList();

}

private EventDto MapToDto(EventEntity entity)

{

return new EventDto

{

Id = entity.Id,

StreamId = entity.StreamId,

Type = entity.Type,

Data = entity.Data,

Timestamp = entity.Timestamp

};

}

}

```

---

### \*\*2.5. Repositories (مخازن)\*\*

##### \*\*هدف:\*\*

مخازن مسئولیت تعامل مستقیم با دیتابیس را بر عهده دارند.

##### \*\*مثال: IEventRepository.cs\*\*

```csharp

public interface IEventRepository

{

Task SaveEventAsync(EventEntity @event);

Task<List<EventEntity>> GetEventsByStreamIdAsync(Guid streamId);

}

```

##### \*\*مثال: EventRepository.cs\*\*

```csharp

public class EventRepository : IEventRepository

{

private readonly EventDbContext \_context;

public EventRepository(EventDbContext context)

{

\_context = context;

}

public async Task SaveEventAsync(EventEntity @event)

{

try

{

\_context.Events.Add(@event);

await \_context.SaveChangesAsync();

}

catch (DbUpdateException ex)

{

// لاگ‌گیری از خطای رخ‌داده

Console.Error.WriteLine($"خطا در ذخیره رویداد: {ex.Message}");

throw new ApplicationException("خطا در ذخیره رویداد در دیتابیس.", ex);

}

}

public async Task<List<EventEntity>> GetEventsByStreamIdAsync(Guid streamId)

{

return await \_context.Events.Where(e => e.StreamId == streamId).ToListAsync();

}

}

```

---

### \*\*2.6. Models (مدل‌ها)\*\*

##### \*\*هدف:\*\*

مدل‌ها شامل داده‌هایی هستند که در سراسر برنامه استفاده می‌شوند.

##### \*\*مثال: EventDto.cs\*\*

```csharp

public class EventDto

{

public Guid Id { get; set; }

public Guid StreamId { get; set; } // شناسه جریان رویداد

public string Type { get; set; } // نوع رویداد

public string Data { get; set; } // داده‌های رویداد

public DateTime Timestamp { get; set; } // زمان ثبت رویداد

}

```

##### \*\*مثال: EventEntity.cs\*\*

```csharp

public class EventEntity

{

public Guid Id { get; set; }

public Guid StreamId { get; set; }

public string Type { get; set; }

public string Data { get; set; }

public DateTime Timestamp { get; set; }

}

```

---

### \*\*2.7. Persistence (مدیریت دیتابیس)\*\*

##### \*\*هدف:\*\*

این بخش شامل تنظیمات مربوط به دیتابیس و مدل‌های موجودیت است.

##### \*\*مثال: EventDbContext.cs\*\*

```csharp

public class EventDbContext : DbContext

{

public EventDbContext(DbContextOptions<EventDbContext> options) : base(options) { }

public DbSet<EventEntity> Events { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.ApplyConfiguration(new EventConfiguration());

}

}

```

##### \*\*مثال: EventConfiguration.cs\*\*

```csharp

public class EventConfiguration : IEntityTypeConfiguration<EventEntity>

{

public void Configure(EntityTypeBuilder<EventEntity> builder)

{

builder.HasKey(e => e.Id);

builder.Property(e => e.StreamId).IsRequired();

builder.Property(e => e.Type).IsRequired().HasMaxLength(255);

builder.Property(e => e.Data).IsRequired();

builder.Property(e => e.Timestamp).HasDefaultValueSql("GETUTCDATE()");

}

}

```

---

### \*\*2.8. Streaming (پردازش جریان‌ها)\*\*

#### \*\*هدف:\*\*

این بخش شامل کلاس‌هایی است که برای پردازش جریان‌های رویدادها (Event Streams) استفاده می‌شوند.

##### \*\*مثال: StreamProcessor.cs\*\*

```csharp

public class StreamProcessor

{

private readonly IEventStoreService \_eventStoreService;

public StreamProcessor(IEventStoreService eventStoreService)

{

\_eventStoreService = eventStoreService;

}

public async Task ProcessStreamAsync(Guid streamId)

{

var events = await \_eventStoreService.GetEventsByStreamIdAsync(streamId);

foreach (var @event in events)

{

// پردازش هر رویداد

Console.WriteLine($"Processing event: {@event.Type} - {@event.Data}");

}

}

}

2.9. appsettings.json\*\*

هدف:\*\*

این فایل شامل تنظیمات اصلی برنامه است، مانند اتصال به دیتابیس.

مثال: appsettings.json\*\*

```json

{

"ConnectionStrings": {

"DefaultConnection": "Server=localhost;Database=EventStoreDb;Trusted\_Connection=True;"

}

}

3. نتیجه‌گیری\*\*

ماژول \*\*EventStoreService\*\* یک ماژول کامل و مستقل است که می‌تواند به‌عنوان بخشی از یک سیستم ردیابی GPS استفاده شود. تمام بخش‌ها شامل کنترلرها، سرویس‌ها، مخازن، مدیریت دیتابیس، و پردازش جریان‌ها به همراه کامنت‌های توضیحی نوشته شده‌اند. این ماژول قابلیت ذخیره‌سازی، بازیابی و پردازش رویدادها را فراهم می‌کند و می‌تواند به‌عنوان بخشی از یک سیستم مبتنی بر \*\*Event Sourcing\*\* استفاده شود.

### \*\*توضیح کامل ماژول SystemMonitoringService\*\*

ماژول \*\*SystemMonitoringService\*\* یک بخش حیاتی در سیستم‌های مبتنی بر نظارت و مانیتورینگ است. این ماژول به بررسی وضعیت سلامت سیستم، جمع‌آوری متریک‌ها (Metrics)، و مدیریت داده‌های مرتبط با عملکرد سیستم می‌پردازد. در ادامه، تمام بخش‌های این ماژول به‌صورت دقیق و ساختارمند توضیح داده شده‌اند.

---

## \*\*1. ساختار کلی ماژول\*\*

```plaintext

/SystemMonitoringService

│

├── Controllers

│ └── HealthCheckController.cs

│

├── Services

│ ├── MonitoringService.cs

│ └── IMonitoringService.cs

│

├── Models

│ └── HealthCheckResult.cs

│

├── Persistence

│ ├── MonitoringDbContext.cs

│ └── EntityConfigurations

│ └── MonitoringConfiguration.cs

│

├── PrometheusMetrics

│ └── MetricsCollector.cs

│

├── appsettings.json

│

└── Program.cs

```

---

## \*\*2. توضیحات جامع برای هر بخش\*\*

---

### \*\*2.1. Program.cs\*\*

##### \*\*هدف:\*\*

این فایل برنامه را راه‌اندازی می‌کند و تنظیمات اولیه را اعمال می‌کند.

##### \*\*مثال: Program.cs\*\*

```csharp

var builder = WebApplication.CreateBuilder(args);

// افزودن سرویس‌های پیش‌فرض ASP.NET Core

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

// افزودن تنظیمات اختصاصی ماژول

builder.Services.AddSystemMonitoringServices(builder.Configuration);

var app = builder.Build();

// تنظیم خط لول درخواست‌ها

if (app.Environment.IsDevelopment())

{

app.UseSwagger(); // فعال کردن Swagger در حالت توسعه

app.UseSwaggerUI(); // فعال کردن UI Swagger

}

app.UseHttpsRedirection(); // هدایت به HTTPS

app.MapControllers(); // مپ کردن کنترلرها

app.Run(); // اجرای برنامه

```

---

### \*\*2.2. ServiceCollectionExtensions.cs\*\*

##### \*\*هدف:\*\*

این فایل شامل تنظیمات \*\*Dependency Injection\*\* برای ثبت سرویس‌ها و مخازن است.

##### \*\*مثال: ServiceCollectionExtensions.cs\*\*

```csharp

public static class ServiceCollectionExtensions

{

public static IServiceCollection AddSystemMonitoringServices(this IServiceCollection services, IConfiguration configuration)

{

// ثبت DbContext برای دسترسی به دیتابیس

services.AddDbContext<MonitoringDbContext>(options =>

options.UseSqlServer(configuration.GetConnectionString("DefaultConnection")));

// ثبت سرویس‌ها

services.AddScoped<IMonitoringService, MonitoringService>();

// ثبت کلاس‌های مربوط به متریک‌ها

services.AddSingleton<MetricsCollector>();

return services;

}

}

```

---

### \*\*2.3. Controllers (کنترلرها)\*\*

##### \*\*هدف:\*\*

کنترلرها واسط بین کلاینت‌ها (مثل وب‌اپلیکیشن یا موبایل) و سرویس‌های داخلی هستند. آن‌ها درخواست‌های HTTP را دریافت کرده و پاسخ مناسب ارسال می‌کنند.

##### \*\*مثال: HealthCheckController.cs\*\*

```csharp

[ApiController]

[Route("api/[controller]")]

public class HealthCheckController : ControllerBase

{

private readonly IMonitoringService \_monitoringService;

public HealthCheckController(IMonitoringService monitoringService)

{

\_monitoringService = monitoringService;

}

/// <summary>

/// بررسی سلامت سیستم

/// </summary>

[HttpGet("check")]

public async Task<IActionResult> CheckSystemHealth()

{

var healthCheckResult = await \_monitoringService.PerformHealthCheckAsync();

return Ok(healthCheckResult);

}

/// <summary>

/// دریافت آخرین وضعیت سلامت سیستم

/// </summary>

[HttpGet("status")]

public async Task<IActionResult> GetSystemHealthStatus()

{

var status = await \_monitoringService.GetCurrentHealthStatusAsync();

return Ok(status);

}

}

```

---

### \*\*2.4. Services (سرویس‌ها)\*\*

##### \*\*هدف:\*\*

سرویس‌ها شامل منطق کسب‌وکار برنامه هستند. آن‌ها کارهایی مثل بررسی سلامت سیستم، جمع‌آوری متریک‌ها، و ذخیره‌سازی داده‌ها را انجام می‌دهند.

##### \*\*مثال: IMonitoringService.cs\*\*

```csharp

public interface IMonitoringService

{

Task<HealthCheckResult> PerformHealthCheckAsync();

Task<HealthCheckResult> GetCurrentHealthStatusAsync();

}

```

##### \*\*مثال: MonitoringService.cs\*\*

```csharp

public class MonitoringService : IMonitoringService

{

private readonly MonitoringDbContext \_context;

private readonly MetricsCollector \_metricsCollector;

public MonitoringService(MonitoringDbContext context, MetricsCollector metricsCollector)

{

\_context = context;

\_metricsCollector = metricsCollector;

}

public async Task<HealthCheckResult> PerformHealthCheckAsync()

{

var cpuUsage = \_metricsCollector.GetCpuUsage();

var memoryUsage = \_metricsCollector.GetMemoryUsage();

var diskUsage = \_metricsCollector.GetDiskUsage();

var networkStatus = \_metricsCollector.GetNetworkStatus();

var isHealthy = cpuUsage < 80 && memoryUsage < 80 && diskUsage < 90 && networkStatus == "OK";

var healthCheckResult = new HealthCheckResult

{

IsHealthy = isHealthy,

Message = isHealthy ? "System is healthy" : "System is not healthy"

};

await \_context.HealthCheckResults.AddAsync(healthCheckResult);

await \_context.SaveChangesAsync();

return healthCheckResult;

}

public async Task<HealthCheckResult> GetCurrentHealthStatusAsync()

{

return await \_context.HealthCheckResults.OrderByDescending(h => h.Timestamp).FirstOrDefaultAsync();

}

}

```

---

### \*\*2.5. Models (مدل‌ها)\*\*

##### \*\*هدف:\*\*

مدل‌ها شامل داده‌هایی هستند که در سراسر برنامه استفاده می‌شوند.

##### \*\*مثال: HealthCheckResult.cs\*\*

```csharp

public class HealthCheckResult

{

public int Id { get; set; }

public bool IsHealthy { get; set; } // وضعیت سلامت سیستم

public string Message { get; set; } // پیام مربوط به وضعیت

public double CpuUsage { get; set; } // درصد استفاده از CPU

public double MemoryUsage { get; set; } // درصد استفاده از حافظه

public double DiskUsage { get; set; } // درصد استفاده از دیسک

public string NetworkStatus { get; set; } // وضعیت شبکه

public DateTime Timestamp { get; set; } // زمان بررسی

}

```

---

### \*\*2.6. Persistence (مدیریت دیتابیس)\*\*

##### \*\*هدف:\*\*

این بخش شامل تنظیمات مربوط به دیتابیس و مدل‌های موجودیت است.

##### \*\*مثال: MonitoringDbContext.cs\*\*

```csharp

public class MonitoringDbContext : DbContext

{

public MonitoringDbContext(DbContextOptions<MonitoringDbContext> options) : base(options) { }

public DbSet<HealthCheckResult> HealthCheckResults { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.ApplyConfiguration(new MonitoringConfiguration());

}

}

```

##### \*\*مثال: MonitoringConfiguration.cs\*\*

```csharp

public class MonitoringConfiguration : IEntityTypeConfiguration<HealthCheckResult>

{

public void Configure(EntityTypeBuilder<HealthCheckResult> builder)

{

builder.HasKey(h => h.Id);

builder.Property(h => h.IsHealthy).IsRequired();

builder.Property(h => h.Message).IsRequired().HasMaxLength(255);

builder.Property(h => h.CpuUsage).IsRequired();

builder.Property(h => h.MemoryUsage).IsRequired();

builder.Property(h => h.DiskUsage).IsRequired();

builder.Property(h => h.NetworkStatus).IsRequired().HasMaxLength(50);

builder.Property(h => h.Timestamp).HasDefaultValueSql("GETUTCDATE()");

}

}

```

---

### \*\*2.7. PrometheusMetrics (جمع‌آوری متریک‌ها)\*\*

##### \*\*هدف:\*\*

این بخش متریک‌های سیستم را جمع‌آوری کرده و به Prometheus ارسال می‌کند.

##### \*\*مثال: MetricsCollector.cs\*\*

```csharp

public class MetricsCollector

{

public double GetCpuUsage()

{

// دریافت میزان استفاده از CPU

return 75.0; // مقدار نمونه

}

public double GetMemoryUsage()

{

// دریافت میزان استفاده از حافظه

return 60.0; // مقدار نمونه

}

public double GetDiskUsage()

{

// دریافت میزان استفاده از دیسک

return 45.0; // مقدار نمونه

}

public string GetNetworkStatus()

{

// دریافت وضعیت شبکه

return "OK"; // مقدار نمونه

}

}

```

---

### \*\*2.8. appsettings.json\*\*

##### \*\*هدف:\*\*

این فایل شامل تنظیمات اصلی برنامه است، مانند اتصال به دیتابیس.

##### \*\*مثال: appsettings.json\*\*

```json

{

"ConnectionStrings": {

"DefaultConnection": "Server=localhost;Database=SystemMonitoringDb;Trusted\_Connection=True;"

}

}

```

---

### \*\*3. نتیجه‌گیری\*\*

ماژول \*\*SystemMonitoringService\*\* یک ماژول کامل و مستقل است که می‌تواند به‌عنوان بخشی از یک سیستم ردیابی GPS یا هر سیستم دیگری استفاده شود. تمام بخش‌ها شامل کنترلرها، سرویس‌ها، مخازن، مدیریت دیتابیس، و جمع‌آوری متریک‌ها به همراه کامنت‌های توضیحی نوشته شده‌اند. این ماژول قابلیت بررسی سلامت سیستم، جمع‌آوری متریک‌ها، و ذخیره‌سازی داده‌ها را فراهم می‌کند و می‌تواند به‌عنوان بخشی از یک سیستم مانیتورینگ استفاده شود.

### \*\*توضیح کامل ماژول JobSchedulerService\*\*

ماژول \*\*JobSchedulerService\*\* یک بخش حیاتی در سیستم‌های مبتنی بر اجرای وظایف زمان‌بندی‌شده (Scheduled Tasks) است. این ماژول به مدیریت و اجرای خودکار وظایف مشخصی در زمان‌های تعیین‌شده می‌پردازد. در ادامه، تمام بخش‌های این ماژول به‌صورت دقیق و ساختارمند توضیح داده شده‌اند.

---

## \*\*1. ساختار کلی ماژول\*\*

```plaintext

/JobSchedulerService

│

├── QuartzJobService.cs

├── IScheduledTask.cs

├── Jobs

│ ├── RenewSubscriptionJob.cs

│ └── SendNotificationJob.cs

├── appsettings.json

└── Program.cs

```

---

## \*\*2. توضیحات جامع برای هر بخش\*\*

---

### \*\*2.1. Program.cs\*\*

##### \*\*هدف:\*\*

این فایل برنامه را راه‌اندازی می‌کند و تنظیمات اولیه را اعمال می‌کند.

##### \*\*مثال: Program.cs\*\*

```csharp

var builder = WebApplication.CreateBuilder(args);

// افزودن سرویس‌های پیش‌فرض ASP.NET Core

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

// افزودن تنظیمات اختصاصی ماژول

builder.Services.AddQuartzServices(builder.Configuration);

var app = builder.Build();

// تنظیم خط لول درخواست‌ها

if (app.Environment.IsDevelopment())

{

app.UseSwagger(); // فعال کردن Swagger در حالت توسعه

app.UseSwaggerUI(); // فعال کردن UI Swagger

}

app.UseHttpsRedirection(); // هدایت به HTTPS

app.MapControllers(); // مپ کردن کنترلرها

app.Run(); // اجرای برنامه

```

---

### \*\*2.2. ServiceCollectionExtensions.cs\*\*

##### \*\*هدف:\*\*

این فایل شامل تنظیمات \*\*Dependency Injection\*\* برای ثبت سرویس‌ها و وظایف زمان‌بندی‌شده است.

##### \*\*مثال: ServiceCollectionExtensions.cs\*\*

```csharp

public static class ServiceCollectionExtensions

{

public static IServiceCollection AddQuartzServices(this IServiceCollection services, IConfiguration configuration)

{

// ثبت Quartz Scheduler

services.AddQuartz(q =>

{

q.UseMicrosoftDependencyInjectionJobFactory();

// تنظیمات وظیفه تمدید اشتراک

var renewSubscriptionJobKey = new JobKey("RenewSubscriptionJob");

q.AddJob<RenewSubscriptionJob>(opts => opts.WithIdentity(renewSubscriptionJobKey));

q.AddTrigger(opts => opts

.ForJob(renewSubscriptionJobKey)

.WithIdentity("RenewSubscriptionJob-trigger")

.WithCronSchedule("0 0 12 \* \* ?")); // هر روز ساعت 12 ظهر

// تنظیمات وظیفه ارسال اعلان

var sendNotificationJobKey = new JobKey("SendNotificationJob");

q.AddJob<SendNotificationJob>(opts => opts.WithIdentity(sendNotificationJobKey));

q.AddTrigger(opts => opts

.ForJob(sendNotificationJobKey)

.WithIdentity("SendNotificationJob-trigger")

.WithCronSchedule("0 0/15 \* \* \* ?")); // هر 15 دقیقه یکبار

});

services.AddQuartzHostedService(q => q.WaitForJobsToComplete = true);

return services;

}

}

```

---

### \*\*2.3. QuartzJobService.cs\*\*

##### \*\*هدف:\*\*

این کلاس شامل منطق کلی برای مدیریت وظایف زمان‌بندی‌شده با استفاده از \*\*Quartz.NET\*\* است.

##### \*\*مثال: QuartzJobService.cs\*\*

```csharp

public class QuartzJobService

{

private readonly IScheduler \_scheduler;

public QuartzJobService(IScheduler scheduler)

{

\_scheduler = scheduler;

}

public async Task StartAsync()

{

await \_scheduler.Start();

}

public async Task StopAsync()

{

await \_scheduler.Shutdown();

}

}

```

---

### \*\*2.4. IScheduledTask.cs\*\*

##### \*\*هدف:\*\*

این اینترفیس یک قرارداد عمومی برای تمام وظایف زمان‌بندی‌شده ایجاد می‌کند.

##### \*\*مثال: IScheduledTask.cs\*\*

```csharp

public interface IScheduledTask

{

Task ExecuteAsync();

}

```

---

### \*\*2.5. Jobs (وظایف زمان‌بندی‌شده)\*\*

#### \*\*2.5.1. RenewSubscriptionJob.cs\*\*

##### \*\*هدف:\*\*

این کلاس وظیفه تمدید خودکار اشتراک‌های کاربران را انجام می‌دهد.

##### \*\*مثال: RenewSubscriptionJob.cs\*\*

```csharp

public class RenewSubscriptionJob : IJob

{

private readonly ISubscriptionService \_subscriptionService;

public RenewSubscriptionJob(ISubscriptionService subscriptionService)

{

\_subscriptionService = subscriptionService;

}

public async Task Execute(IJobExecutionContext context)

{

Console.WriteLine("شروع وظیفه تمدید اشتراک...");

await \_subscriptionService.RenewExpiredSubscriptionsAsync();

Console.WriteLine("وظیفه تمدید اشتراک به پایان رسید.");

}

}

```

---

#### \*\*2.5.2. SendNotificationJob.cs\*\*

##### \*\*هدف:\*\*

این کلاس وظیفه ارسال اعلان‌های زمان‌بندی‌شده به کاربران را انجام می‌دهد.

##### \*\*مثال: SendNotificationJob.cs\*\*

```csharp

public class SendNotificationJob : IJob

{

private readonly INotificationService \_notificationService;

public SendNotificationJob(INotificationService notificationService)

{

\_notificationService = notificationService;

}

public async Task Execute(IJobExecutionContext context)

{

Console.WriteLine("شروع وظیفه ارسال اعلان...");

await \_notificationService.SendScheduledNotificationsAsync();

Console.WriteLine("وظیفه ارسال اعلان به پایان رسید.");

}

}

```

---

### \*\*2.6. appsettings.json\*\*

##### \*\*هدف:\*\*

این فایل شامل تنظیمات اصلی برنامه است، مانند Cron Expression‌ها برای زمان‌بندی وظایف.

##### \*\*مثال: appsettings.json\*\*

```json

{

"Quartz": {

"Jobs": {

"RenewSubscriptionJob": {

"CronExpression": "0 0 12 \* \* ?" // هر روز ساعت 12 ظهر

},

"SendNotificationJob": {

"CronExpression": "0 0/15 \* \* \* ?" // هر 15 دقیقه یکبار

}

}

}

}

```

---

### \*\*3. نتیجه‌گیری\*\*

ماژول \*\*JobSchedulerService\*\* یک ماژول کامل و مستقل است که می‌تواند به‌عنوان بخشی از یک سیستم ردیابی GPS یا هر سیستم دیگری استفاده شود. تمام بخش‌ها شامل وظایف زمان‌بندی‌شده، مدیریت Quartz Scheduler، و تنظیمات مربوط به Cron Expression به همراه کامنت‌های توضیحی نوشته شده‌اند. این ماژول قابلیت اجرای خودکار وظایفی مانند تمدید اشتراک‌ها و ارسال اعلان‌ها را فراهم می‌کند و می‌تواند به‌عنوان بخشی از یک سیستم زمان‌بندی‌شده استفاده شود.

### \*\*ساختار کامل پوشه‌بندی ماژول SharedKernel با نام کلاس‌ها\*\*

در ادامه، ساختار کامل پوشه‌بندی ماژول \*\*SharedKernel\*\* به همراه نام دقیق کلاس‌ها و فایل‌ها نوشته شده است. این ساختار شامل تمام بخش‌های مورد نیاز برای ایجاد یک ماژول مشترک و قابل استفاده در سراسر پروژه است.

---

## \*\*1. ساختار کامل پوشه‌بندی\*\*

```plaintext

/SharedKernel

│

├── Audit/

│ └── AuditLogService.cs

│

├── Constants/

│ └── AppConstants.cs

│

├── Exceptions/

│ └── CustomException.cs

│

├── Extensions/

│ └── ServiceCollectionExtensions.cs

│

├── Interfaces/

│ └── IRepository.cs

│

├── Middleware/

│ ├── ExceptionHandlingMiddleware.cs

│ └── RequestLoggingMiddleware.cs

│

├── Utilities/

│ ├── DateTimeHelper.cs

│ ├── EncryptionUtils.cs

│ └── JsonHelper.cs

│

├── ValueObjects/

│ ├── EmailAddress.cs

│ ├── PhoneNumber.cs

│ └── GeoCoordinate.cs

│

├── Enums/

│ ├── LogLevel.cs

│ ├── UserStatus.cs

│ └── NotificationType.cs

│

└── Config/

└── appsettings.json

```

---

## \*\*2. توضیحات جامع برای هر بخش و نام کلاس‌ها\*\*

---

### \*\*2.1. Audit (مدیریت لاگ‌های عملیاتی)\*\*

##### \*\*هدف:\*\*

این بخش شامل سرویس‌های لاگ‌زنی عملیاتی است که برای ثبت تغییرات در سیستم استفاده می‌شوند.

##### \*\*فایل‌ها و کلاس‌ها:\*\*

- \*\*AuditLogService.cs\*\*

```csharp

public class AuditLogService

{

private readonly ILogger<AuditLogService> \_logger;

public AuditLogService(ILogger<AuditLogService> logger)

{

\_logger = logger;

}

public void LogAction(string action, string details)

{

\_logger.LogInformation($"Action: {action}, Details: {details}");

}

}

```

---

### \*\*2.2. Constants (ثابت‌ها)\*\*

##### \*\*هدف:\*\*

این بخش شامل ثابت‌هایی است که در سراسر پروژه استفاده می‌شوند.

##### \*\*فایل‌ها و کلاس‌ها:\*\*

- \*\*AppConstants.cs\*\*

```csharp

public class AppConstants

{

public const string DefaultConnectionStringName = "DefaultConnection";

public const int MaxPageSize = 100;

public const string DateFormat = "yyyy-MM-dd HH:mm:ss";

}

```

---

### \*\*2.3. Exceptions (مدیریت خطاها)\*\*

##### \*\*هدف:\*\*

این بخش شامل استثناهای سفارشی است که برای مدیریت خطاهای خاص در سیستم استفاده می‌شوند.

##### \*\*فایل‌ها و کلاس‌ها:\*\*

- \*\*CustomException.cs\*\*

```csharp

public class CustomException : Exception

{

public int ErrorCode { get; }

public CustomException(string message, int errorCode) : base(message)

{

ErrorCode = errorCode;

}

}

```

---

### \*\*2.4. Extensions (افزونه‌ها)\*\*

##### \*\*هدف:\*\*

این بخش شامل افزونه‌های عمومی برای کلاس‌ها و متد‌ها است.

##### \*\*فایل‌ها و کلاس‌ها:\*\*

- \*\*ServiceCollectionExtensions.cs\*\*

```csharp

public static class ServiceCollectionExtensions

{

public static IServiceCollection AddSharedKernelServices(this IServiceCollection services, IConfiguration configuration)

{

services.AddSingleton<AppConstants>();

services.AddScoped<AuditLogService>();

return services;

}

}

```

---

### \*\*2.5. Interfaces (قراردادها)\*\*

##### \*\*هدف:\*\*

این بخش شامل اینترفیس‌های عمومی است که در سراسر پروژه استفاده می‌شوند.

##### \*\*فایل‌ها و کلاس‌ها:\*\*

- \*\*IRepository.cs\*\*

```csharp

public interface IRepository<T> where T : class

{

Task<T> GetByIdAsync(Guid id);

Task<IEnumerable<T>> GetAllAsync();

Task AddAsync(T entity);

Task UpdateAsync(T entity);

Task DeleteAsync(Guid id);

}

```

---

### \*\*2.6. Middleware (میان‌افزارها)\*\*

##### \*\*هدف:\*\*

Middlewareها برای مدیریت درخواست‌ها و پاسخ‌ها در خط لول ASP.NET Core استفاده می‌شوند.

##### \*\*فایل‌ها و کلاس‌ها:\*\*

- \*\*ExceptionHandlingMiddleware.cs\*\*

```csharp

public class ExceptionHandlingMiddleware

{

private readonly RequestDelegate \_next;

public ExceptionHandlingMiddleware(RequestDelegate next)

{

\_next = next;

}

public async Task InvokeAsync(HttpContext context)

{

try

{

await \_next(context);

}

catch (CustomException ex)

{

context.Response.StatusCode = ex.ErrorCode;

await context.Response.WriteAsJsonAsync(new { Message = ex.Message });

}

catch (Exception ex)

{

context.Response.StatusCode = StatusCodes.Status500InternalServerError;

await context.Response.WriteAsJsonAsync(new { Message = "An unexpected error occurred." });

}

}

}

```

- \*\*RequestLoggingMiddleware.cs\*\*

```csharp

public class RequestLoggingMiddleware

{

private readonly RequestDelegate \_next;

private readonly ILogger<RequestLoggingMiddleware> \_logger;

public RequestLoggingMiddleware(RequestDelegate next, ILogger<RequestLoggingMiddleware> logger)

{

\_next = next;

\_logger = logger;

}

public async Task InvokeAsync(HttpContext context)

{

\_logger.LogInformation($"Request: {context.Request.Method} {context.Request.Path}");

await \_next(context);

}

}

```

---

### \*\*2.7. Utilities (ابزارهای عمومی)\*\*

##### \*\*هدف:\*\*

این بخش شامل کلاس‌ها و متدهای کمکی است که در سراسر پروژه استفاده می‌شوند.

##### \*\*فایل‌ها و کلاس‌ها:\*\*

- \*\*DateTimeHelper.cs\*\*

```csharp

public static class DateTimeHelper

{

public static string ToPersianDate(DateTime dateTime)

{

var pc = new PersianCalendar();

return $"{pc.GetYear(dateTime)}/{pc.GetMonth(dateTime):00}/{pc.GetDayOfMonth(dateTime):00}";

}

}

```

- \*\*EncryptionUtils.cs\*\*

```csharp

public static class EncryptionUtils

{

public static string HashPassword(string password)

{

using var sha256 = SHA256.Create();

var hashedBytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(password));

return BitConverter.ToString(hashedBytes).Replace("-", "").ToLower();

}

}

```

- \*\*JsonHelper.cs\*\*

```csharp

public static class JsonHelper

{

public static T Deserialize<T>(string json)

{

return JsonSerializer.Deserialize<T>(json);

}

public static string Serialize<T>(T obj)

{

return JsonSerializer.Serialize(obj);

}

}

```

---

### \*\*2.8. ValueObjects (اشیاء مقداری)\*\*

##### \*\*هدف:\*\*

اشیاء مقداری (Value Objects) برای نمایش داده‌هایی استفاده می‌شوند که بدون شناسه یکتا عمل می‌کنند.

##### \*\*فایل‌ها و کلاس‌ها:\*\*

- \*\*EmailAddress.cs\*\*

```csharp

public record EmailAddress

{

public string Value { get; }

public EmailAddress(string value)

{

if (!Regex.IsMatch(value, @"^[^@\s]+@[^@\s]+\.[^@\s]+$"))

throw new CustomException("Invalid email format", 400);

Value = value;

}

public override string ToString() => Value;

}

```

- \*\*PhoneNumber.cs\*\*

```csharp

public record PhoneNumber

{

public string Value { get; }

public PhoneNumber(string value)

{

if (!Regex.IsMatch(value, @"^\+?[0-9]{10,15}$"))

throw new CustomException("Invalid phone number format", 400);

Value = value;

}

public override string ToString() => Value;

}

```

- \*\*GeoCoordinate.cs\*\*

```csharp

public record GeoCoordinate

{

public double Latitude { get; }

public double Longitude { get; }

public GeoCoordinate(double latitude, double longitude)

{

if (latitude < -90 || latitude > 90 || longitude < -180 || longitude > 180)

throw new CustomException("Invalid geo coordinate", 400);

Latitude = latitude;

Longitude = longitude;

}

}

```

---

### \*\*2.9. Enums (تعریف انواع شمارشی سراسری)\*\*

##### \*\*هدف:\*\*

Enums برای تعریف مقادیر ثابت و استاندارد استفاده می‌شوند که در سراسر سیستم قابل استفاده هستند.

##### \*\*فایل‌ها و کلاس‌ها:\*\*

- \*\*LogLevel.cs\*\*

```csharp

public enum LogLevel

{

Info,

Warning,

Error,

Critical

}

```

- \*\*UserStatus.cs\*\*

```csharp

public enum UserStatus

{

Active,

Inactive,

Suspended

}

```

- \*\*NotificationType.cs\*\*

```csharp

public enum NotificationType

{

Info,

Warning,

Alert

}

```

---

### \*\*2.10. Config (تنظیمات پیکربندی)\*\*

##### \*\*هدف:\*\*

این بخش شامل تنظیمات اختصاصی برنامه است.

##### \*\*فایل‌ها و کلاس‌ها:\*\*

- \*\*appsettings.json\*\*

```json

{

"ConnectionStrings": {

"DefaultConnection": "Server=localhost;Database=SharedKernelDb;Trusted\_Connection=True;"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

}

}

**افزودن پوشه‌ی Results/ برای الگوهای پاسخ**

یک کلاس مثل Result<T> یا OperationResult اضافه کن:

public class Result<T>

{

public bool Success { get; set; }

public string Message { get; set; }

public T? Data { get; set; }

public static Result<T> Ok(T data) => new() { Success = true, Data = data };

public static Result<T> Fail(string message) => new() { Success = false, Message = message };

}

**فزودن Validatorهای مستقل**

پوشه‌ی Validation/ ایجاد کن و برای Value Object‌ها یا داده‌های ورودی، کلاس‌های Validator اختصاصی داشته باش، تا از Validationهای پراکنده جلوگیری شه.

**گسترش لاگ‌ها**

در AuditLogService می‌تونی متدهایی مثل LogEntityChange, LogError, LogUserAction اضافه کنی تا بازهم ساختارمندتر شه.

**پشتیبانی از Localization**

در Exceptionها یا پیام‌های ثابت، در نظر بگیر که از IStringLocalizer استفاده بشه تا ترجمه چندزبانه پشتیبانی بشه.

**استفاده از Interceptor یا Pipeline Behavior (اگر با MediatR کار می‌کنی)**

پوشه‌ی Behaviors/ برای افزودن لاگ‌زنی، هندلینگ استثناها و Performance Tracking در قالب Pipeline مفید می‌تونه باشه.

نسخه ارتقاء یافته ساختار پوشه:

/SharedKernel

│

├── Audit/

├── Behaviors/

├── Config/

├── Constants/

├── Enums/

├── Exceptions/

├── Extensions/

├── Interfaces/

├── Middleware/

├── Results/

├── Utilities/

├── Validation/

├── ValueObjects/

└── Resources/ (برای Localization اگر نیاز بود)

نتیجه‌گیری\*\*

این ساختار کامل پوشه‌بندی ماژول \*\*SharedKernel\*\* شامل تمام کلاس‌ها و فایل‌های مورد نیاز برای ایجاد یک ماژول مشترک و قابل استفاده در سراسر پروژه است. هر بخش به‌صورت جداگانه توضیح داده شده و نام کلاس‌ها و فایل‌ها به‌صورت دقیق ذکر شده است.

**اضافه کردن توضیح Dependency Injection در هر بخش**

مثلاً در RedisConfiguration یا SerilogLogger می‌تونی اشاره کنی که این متدها در Startup با builder.Services.Add...() تزریق می‌شن و دلیل استفاده از extension methods چیست (جداسازی کد کانفیگ از Program.cs و رعایت اصل SRP).)

### \*\*توضیحات کامل ماژول Infrastructure با در نظر گرفتن موارد اضافی\*\*

ماژول \*\*Infrastructure\*\* یک بخش حیاتی در سیستم‌های مبتنی بر معماری پاک (Clean Architecture) است. این ماژول شامل تمام ابزارها، تنظیمات، و پیاده‌سازی‌های لایه زیرساخت (Infrastructure Layer) است که به تعامل با منابع خارجی مانند دیتابیس‌ها، صف‌های پیام‌رسانی، لاگ‌گیری، کش‌سازی، و امنیت کمک می‌کند. در ادامه، تمام بخش‌های این ماژول به‌صورت دقیق و ساختارمند توضیح داده شده‌اند. همچنین، نکات مهم مرتبط با \*\*Dependency Injection\*\*، \*\*امنیت\*\*، و \*\*بهینه‌سازی\*\* به‌طور کامل اضافه شده‌اند.

---

## \*\*1. ساختار کلی ماژول\*\*

```plaintext

/Infrastructure

│

├── Persistence

│ └── BaseDbContext.cs

│

├── Messaging

│ ├── MqttConfiguration.cs

│ └── SignalRConfiguration.cs

│

├── Logging

│ └── SerilogLogger.cs

│

├── Caching

│ └── RedisConfiguration.cs

│

├── Security

│ └── EncryptionHelper.cs

│

├── HealthChecks

│ └── HealthCheckExtensions.cs

│

├── Extensions

│ └── InfrastructureServiceExtensions.cs

│

├── appsettings.json

│

└── Program.cs

```

---

## \*\*2. توضیحات جامع برای هر بخش\*\*

---

### \*\*2.1. Persistence (مدیریت دیتابیس)\*\*

##### \*\*هدف:\*\*

این بخش شامل کلاس‌ها و تنظیمات مربوط به اتصال به دیتابیس و مدیریت موجودیت‌ها است.

##### \*\*مثال: BaseDbContext.cs\*\*

```csharp

// کلاس پایه برای DbContext

public class BaseDbContext : DbContext

{

public BaseDbContext(DbContextOptions options) : base(options) { }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

// تنظیمات عمومی برای جداول

foreach (var entityType in modelBuilder.Model.GetEntityTypes())

{

// افزودن Timestamp به همه جداول

if (entityType.ClrType.GetProperty("CreatedOn") != null)

{

modelBuilder.Entity(entityType.ClrType).Property("CreatedOn").HasDefaultValueSql("GETUTCDATE()");

}

}

}

}

```

---

### \*\*2.2. Messaging (پیام‌رسانی)\*\*

##### \*\*هدف:\*\*

این بخش شامل تنظیمات و کلاس‌های مربوط به پیام‌رسانی (مانند MQTT و SignalR) است.

##### \*\*مثال: MqttConfiguration.cs\*\*

```csharp

// تنظیمات MQTT

public static class MqttConfiguration

{

public static MqttClientOptions GetMqttOptions(IConfiguration configuration)

{

var mqttSettings = configuration.GetSection("MqttSettings");

return new MqttClientOptionsBuilder()

.WithTcpServer(mqttSettings["Host"], int.Parse(mqttSettings["Port"]))

.WithClientId(mqttSettings["ClientId"])

.Build();

}

}

```

##### \*\*مثال: SignalRConfiguration.cs\*\*

```csharp

// تنظیمات SignalR

public static class SignalRConfiguration

{

public static IServiceCollection AddSignalRServices(this IServiceCollection services)

{

services.AddSignalR();

return services;

}

}

```

---

### \*\*2.3. Logging (لاگ‌گیری)\*\*

##### \*\*هدف:\*\*

این بخش شامل تنظیمات و کلاس‌های مربوط به لاگ‌گیری است.

##### \*\*مثال: SerilogLogger.cs\*\*

```csharp

// تنظیمات Serilog

public static class SerilogLogger

{

public static IHostBuilder ConfigureSerilog(this IHostBuilder host, IConfiguration configuration)

{

return host.UseSerilog((context, config) =>

{

config.ReadFrom.Configuration(configuration);

});

}

}

```

- \*\*Dependency Injection:\*\*

این متد در فایل `Program.cs` با استفاده از `builder.Host.ConfigureSerilog()` فراخوانی می‌شود. استفاده از Extension Method به جداسازی تنظیمات لاگ‌گیری از `Program.cs` کمک می‌کند و اصل SRP (Single Responsibility Principle) را رعایت می‌کند.

---

### \*\*2.4. Caching (کش‌سازی)\*\*

##### \*\*هدف:\*\*

این بخش شامل تنظیمات و کلاس‌های مربوط به کش‌سازی با استفاده از Redis است.

##### \*\*مثال: RedisConfiguration.cs\*\*

```csharp

// تنظیمات Redis

public static class RedisConfiguration

{

public static IServiceCollection AddRedisCache(this IServiceCollection services, IConfiguration configuration)

{

services.AddStackExchangeRedisCache(options =>

{

options.Configuration = configuration["Redis:ConnectionString"];

});

return services;

}

}

```

- \*\*Dependency Injection:\*\*

این متد در فایل `Program.cs` با استفاده از `builder.Services.AddRedisCache()` فراخوانی می‌شود. استفاده از Extension Method به جداسازی تنظیمات Redis از `Program.cs` کمک می‌کند و اصل SRP را رعایت می‌کند.

---

### \*\*2.5. Security (امنیت)\*\*

##### \*\*هدف:\*\*

این بخش شامل ابزارها و کلاس‌های مربوط به امنیت (مانند رمزگذاری) است.

##### \*\*مثال: EncryptionHelper.cs\*\*

```csharp

// ابزار رمزگذاری

public static class EncryptionHelper

{

public static string Encrypt(string plainText, string key)

{

if (key.Length != 16 && key.Length != 24 && key.Length != 32)

throw new ArgumentException("Key length must be 16, 24, or 32 bytes.");

using var aes = Aes.Create();

aes.Key = Encoding.UTF8.GetBytes(key);

aes.IV = GenerateRandomIV(); // تولید IV تصادفی

var encryptor = aes.CreateEncryptor(aes.Key, aes.IV);

using var ms = new MemoryStream();

ms.Write(aes.IV, 0, aes.IV.Length); // ذخیره IV در ابتدا

using var cs = new CryptoStream(ms, encryptor, CryptoStreamMode.Write);

using (var sw = new StreamWriter(cs))

{

sw.Write(plainText);

}

return Convert.ToBase64String(ms.ToArray());

}

public static string Decrypt(string cipherText, string key)

{

if (key.Length != 16 && key.Length != 24 && key.Length != 32)

throw new ArgumentException("Key length must be 16, 24, or 32 bytes.");

using var aes = Aes.Create();

aes.Key = Encoding.UTF8.GetBytes(key);

var data = Convert.FromBase64String(cipherText);

aes.IV = data.Take(16).ToArray(); // خواندن IV از ابتدا

var decryptor = aes.CreateDecryptor(aes.Key, aes.IV);

using var ms = new MemoryStream(data.Skip(16).ToArray());

using var cs = new CryptoStream(ms, decryptor, CryptoStreamMode.Read);

using var sr = new StreamReader(cs);

return sr.ReadToEnd();

}

private static byte[] GenerateRandomIV()

{

var iv = new byte[16];

using var rng = RandomNumberGenerator.Create();

rng.GetBytes(iv);

return iv;

}

}

```

- \*\*نکات امنیتی:\*\*

- استفاده از IV تصادفی برای افزایش امنیت.

- ذخیره IV در ابتدای خروجی رمزنگاری (`Output = base64(IV + CipherText)`).

---

### \*\*2.6. HealthChecks (بررسی سلامت سیستم)\*\*

##### \*\*هدف:\*\*

این بخش شامل تنظیمات و ابزارهای مربوط به بررسی سلامت سیستم است.

##### \*\*مثال: HealthCheckExtensions.cs\*\*

```csharp

// افزونه‌های بررسی سلامت

public static class HealthCheckExtensions

{

public static IServiceCollection AddCustomHealthChecks(this IServiceCollection services, IConfiguration configuration)

{

services.AddHealthChecks()

.AddSqlServer(configuration.GetConnectionString("DefaultConnection"), name: "Database")

.AddRedis(configuration["Redis:ConnectionString"], name: "Redis");

return services;

}

}

```

- \*\*Dependency Injection:\*\*

این متد در فایل `Program.cs` با استفاده از `builder.Services.AddCustomHealthChecks()` فراخوانی می‌شود. همچنین، باید خط زیر به `Program.cs` اضافه شود:

```csharp

app.MapHealthChecks("/health");

```

---

### \*\*2.7. Extensions (تنظیمات مشترک)\*\*

##### \*\*هدف:\*\*

این بخش شامل تمام Extension Methods مرتبط با زیرساخت است.

##### \*\*مثال: InfrastructureServiceExtensions.cs\*\*

```csharp

public static class InfrastructureServiceExtensions

{

public static IServiceCollection AddInfrastructureServices(this IServiceCollection services, IConfiguration configuration)

{

services.AddRedisCache(configuration);

services.AddCustomHealthChecks(configuration);

return services;

}

}

```

- \*\*دلیل استفاده از Extension Methods:\*\*

جمع‌آوری تمام تنظیمات زیرساخت در یک فایل واحد به جداسازی کد کانفیگ از `Program.cs` کمک می‌کند و اصل SRP را رعایت می‌کند.

---

### \*\*2.8. appsettings.json\*\*

##### \*\*هدف:\*\*

این فایل شامل تنظیمات اختصاصی برنامه است.

##### \*\*مثال: appsettings.json\*\*

```json

{

"ConnectionStrings": {

"DefaultConnection": "Server=localhost;Database=AppDb;Trusted\_Connection=True;"

},

"Redis": {

"ConnectionString": "localhost:6379"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

}

}

```

---

### \*\*2.9. Program.cs\*\*

##### \*\*هدف:\*\*

این فایل برنامه را راه‌اندازی می‌کند و تنظیمات اولیه را اعمال می‌کند.

##### \*\*مثال: Program.cs\*\*

```csharp

var builder = WebApplication.CreateBuilder(args);

// افزودن سرویس‌های پیش‌فرض ASP.NET Core

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

// افزودن تنظیمات اختصاصی ماژول

builder.Services.AddInfrastructureServices(builder.Configuration);

var app = builder.Build();

// تنظیم خط لول درخواست‌ها

if (app.Environment.IsDevelopment())

{

app.UseSwagger(); // فعال کردن Swagger در حالت توسعه

app.UseSwaggerUI(); // فعال کردن UI Swagger

}

app.UseHttpsRedirection(); // هدایت به HTTPS

app.MapControllers(); // مپ کردن کنترلرها

app.MapHealthChecks("/health"); // اضافه کردن Health Check Endpoint

app.Run(); // اجرای برنامه

**امنیت (Security) — گسترش بیشتر**

علاوه بر رمزنگاری با AES، می‌تونی موارد زیر رو هم در ماژول امنیت اضافه کنی:

public static class HashingHelper

{

public static string HashPassword(string password)

{

return BCrypt.Net.BCrypt.HashPassword(password);

}

public static bool VerifyPassword(string password, string hashedPassword)

{

return BCrypt.Net.BCrypt.Verify(password, hashedPassword);

}

}

* از کتابخانه BCrypt.Net-Next استفاده کن.
* مناسب برای رمز عبور کاربران.

**🔹 توکن‌سازی JWT (در صورت نیاز به احراز هویت)**

public static class JwtHelper

{

public static string GenerateToken(string userId, string secret, int expireMinutes = 60)

{

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secret));

var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var token = new JwtSecurityToken(

claims: new[] { new Claim(ClaimTypes.NameIdentifier, userId) },

expires: DateTime.UtcNow.AddMinutes(expireMinutes),

signingCredentials: creds

);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

**Messaging — افزودن RabbitMQ یا Kafka (در صورت نیاز)**

اگر سیستم نیاز به صف پیام‌رسانی قوی‌تری داشته باشه، اضافه کردن RabbitMQ و Kafka خیلی کاربردی خواهد بود. مثلاً برای RabbitMQ:

public static class RabbitMqConfiguration

{

public static IServiceCollection AddRabbitMq(this IServiceCollection services, IConfiguration config)

{

var factory = new ConnectionFactory()

{

HostName = config["RabbitMq:Host"],

UserName = config["RabbitMq:Username"],

Password = config["RabbitMq:Password"]

};

var connection = factory.CreateConnection();

services.AddSingleton(connection);

services.AddSingleton(connection.CreateModel());

return services;

}

}

**تنظیمات مربوط به Dependency Injection برای اینترفیس‌ها**

در فایل InfrastructureServiceExtensions.cs می‌تونی اینجوری ثبتشون کنی

services.AddScoped<IEncryptionHelper, EncryptionHelper>();

services.AddScoped<IHashingHelper, HashingHelper>();

3. نتیجه‌گیری\*\*

ماژول \*\*Infrastructure\*\* با اضافه کردن نکات مهم مرتبط با \*\*Dependency Injection\*\*، \*\*امنیت\*\*، و \*\*بهینه‌سازی\*\* به یک ماژول کامل و مستقل تبدیل شده است. این ماژول قابلیت ارائه زیرساخت‌های لازم برای اجرای عملیات‌های مختلف را فراهم می‌کند و موارد مهمی مانند جداسازی کد کانفیگ، استفاده از IV تصادفی، و اضافه کردن Health Check Endpoint را پوشش می‌دهد.