Solution İçerisindeki Katmanlar ve Sorumlulukları

Solution üç ana katmandan oluşmaktadır. Bu üç katman kendi içlerinde tekrar katmanlara bölünebilmektedirler.

1 – VDB.Architecture:

Projelerin ortak kullanma ihtiyacı duyduğu concern metotlar ve solution geneli implementasyon standartlarına uymak isteyen mikro servis ile workerların kullanabileceği base class'ları ve interfaceleri içeren katmandır. Kendi içinde 6 gruptan oluşur.

1.1 – VDB.Architecture.AppException

Uygulama standartı olarak bir süreçte hatalı bir durum olması hâlinde süreç hatayla ilgili doğru tipte bir Exception fırlatmalıdır. Bu hatayla ne yapılacağı ve client'a nasıl dönüş yapılacağı süreci yöneten mikro servis ya da worker tarafından belirlenmelidir, süreçler hata yakalama ve yönetme ile ilgilenmemelidirler. İlgili solution folder içindeki projeler bu senaryonun gerçeklenmesi için gerekli DLLleri içermektedirler.

1.1.1 – VDB.Architecture.AppException.Manager

Mikro servislerin çağırdığı süreçlerde fırlatılan Exceptionların clientlara doğrudan iletilmemesi, detayların gizlenip yaşanan Exception tipine göre bir Http durum kodu ve yaşanan Excepiton'ın uygulama koduna göre bir mesajın belirlenerek bu bilgilerin clientlara dönülmesi beklenmektedir. Bu DLL içerisindeki ExceptionParser sınıfı bu senaryoyu gerçekleştirir.

1.1.2 – VDB.Architecture.AppException.Model

Uygulama içerisindeki durumlara göre fırlatılması gereken Exception tiplerinin ve bu Exceptionların barındırdıkları mesaj tiplerinin tanımlandığı DLLdir.

1.2 – VDB.Architecture.Concern

Uygulama içerisindeki cross-cutting concernlerin bulunduğu solution folderdır.

1.2.1 – VDB.Architecture.Concern.CustomAttributes

Uygulama içerisinde birden fazla mikro servisin ortak kullanabileceği özel Attribute tanımlarını barındırır.

1.2.2 – VDB.Architecture.Concern.ExtensionMethods

Standart .NET kütüphanesindeki sınıflara eklenen Extension metotları barındırır.

1.2.3 – VDB.Architecture.Concern.GenericValidator

Uygulama genelinde dışardan erişime açık ve bir kayıt yaratan tüm servislerde aşağıdaki validasyonlara ihtiyaç duyulmaktadır:

1. Null ya da boş olabilir mi?
2. String ise izin verilen boyut aralığında mı, değilse izin verilen değer aralığında mı?
3. Sadece sabit bir değer havuzundan değer alıyorsa bu havuza uyuyor mu?

Bu validasyonlar servis bazlı kontrol edildiğinde gereksiz bir kod tekrarına sebep olmakta ve bir validasyon kütüphanesi ya da ASP.NET içerisine gömülü Model Validation kullanıldığında sınır değerleri kod içerisinde sabitler ya da uygulama bazlı config içinde değerler olmakta. Validasyon sınırlarının dinamik olarak yönetilebilmesi ve kod tekrarından kurtulmak için iş süreçleri bu DLL içerisindeki Validator sınıfını kullanabilir. Sınıf referans veri olarak aldığı değerlerle otomatik olarak gönderilen nesneyi validate edebilmektedir ve yakaladığı hataları bir ValidationException içerisinde fırlatarak süreci durdurabilmektedir.

1.2.4 – VDB.Architecture.Concern.Helper

Herhangi özel bir bağlamı olmayan yardımcı fonksiyonları bulundurur.

1.2.5 – VDB.Architecture.Concern.Options

Uygulama genelinde konfigürasyonel değerler Options patterini ile okunmaktadır. Tüm süreçlerin ihtiyaç duyabileceği aynı yapıya sahip Options sınıflarını bulundurur.

1.2.6 – VDB.Architecture.Concern.Resources

Uygulama herhangi bir mesaj (örneğin hata mesajı) üreteceği zaman ilgili mesajın referans veride bulunmasını sağlayacak anahtar sabitleri bulundurur.

1.3 – VDB.Architecture.Data

Veri erişimi için standart altyapıları barındıran solution folder.

1.3.1 – VDB.Architecture.Data.Context

Mikro servislerin kendi Entity Framework Data Contextlerini yaratırken standart yapıyı implemente etmek için kullanabilecekleri abstract sınıfları ve interfaceleri içerir.

1.3.2 – VDB.Architecture.Data.Repository

Data Contextlere erişim için uygulama genelinde Repository pattern kullanılmıştır. Herhangi bir X Entitysi için XRepository<Context, Entity> şeklinde bir repository tanımlanıp ilgili repositorynin bu DLL içerisindeki SoftDeleteRepository ya da HardDeleteRepository'den türetilmiş olması beklenmektedir. Base repositoryler içlerinde veri erişimi için yaygın metotları barındırmaktadır. Uygulama kodunun Data Context üzerinden veriye erişmek yerine XRepository üzerinden erişip base

repositorylerin sağlamadığı jenerik metotları XRepository içerisinde implemente etmesi beklenmektedir.

1.3.3 – VDB.Architecture.Data.UnitOfWork

Uygulama genelinde transaction yönetimi için Unit Of Work pattern kullanılmıştır. Her veritabanının kendi Unit Of Work'ü vardır ve içerisinde her obje için bir Repository tanımlar. Uygulama genelinde Repositorylere doğrudan erişilmeyip tüm veri işlemleri için Unit Of Work üzerinden işlem yapılması beklenir. Mevzubahis DLL standart Unit Of Work implementasyonu için gerekli base class ve interface'i içerir.

1.4 – VDB.Architecture.Model

Uygulama genelinde kullanılabilecek veri modellerinin DLLlerini içeren solution folder.

1.4.1 – VDB.Architecture.Model.Entity

Standart veri erişim implementasyonunda her Entity objesinin bu DLL içerisindeki SoftDeletedEntity ya da HardDeletedEntity'den türetilmesi beklenmektedir.

1.4.2 – VDB.Architecture.Model.Messaging

Message queuelara atılan her mesajın bu DLL içerisindeki BaseMessagingRequest'ten türetilmesi beklenmektedir. Bu sayede her mesaja id gibi standart değerler eklenebilmektedir.

1.5 – VDB.Architecture.Web

Web API projelerinin standardizasyonunda kullanılan DLLleri içeren solution folder.

1.5.1 – VDB.Architecture.Web.Core

Web API projelerinin Startup işlemlerinin ortaklaştırılması için gerekli metotları barındırır.

1.5.2 – VDB.Architecture.Worker.Core

Background worker projelerinin Startup işlemlerinin ortaklaştırılması için gerekli metotları barındırır.

2 – VDB.GateWay

Uygulamadaki hiçbir servis doğrudan dışarıya açık durumda değildir. Tüm servislere erişimler GateWay üzerinden gerçekleştirilir. GateWayler aynı zamanda BFF görevi de görmektedirler. Her BFF için bu solution folderda bir GateWay olması beklenmektedir.

2.1 – VDB.GateWay.Web

Web uygulamaları için GateWay. Ocelot ile implemente edilmiştir ve tüm konfigürasyonları /Configuration/Ocelot/Ocelot.json dosyasından yönetilir.

3 – VDB.MicroServices

Her bir mikro servisi başka bir solution folder olarak içinde barındıran ana solution folder. Buradaki MicroServices ifadesini “bağlam” olarak düşünmek gerekmektedir çünkü VDB.MicroServices.X şeklindeki bir yapı içerisinde servisleri barındıran VDB.MicroServices.X.Web şeklinde bir solution folder ve background taskları barındıran VDB.MicroServices.X.Worker şeklinde bir solution folder bulunmaktadır. Aynı veritabanı üzerinde aynı bağlamda çalıştıkları için Workerlar APIlardan VDB.Workers şeklinde bir ana katmanda ayrılmamış, gruplandırma kolaylığı olarak MicroServices katmanına dahil edilmiştir.

Anlatımda her bir mikroservisin her katmanına ayrıca değinilmeyecek, X adındaki bir mikroservis üzerinden katmanların ve DLLlerin süper seti alınarak görevleri anlatılacaktır. Burada anlatılan her katman ve DLL her mikroserviste olmak zorunda değildir. Her bir mikro servisin ve background workerın görev tanımı dökümanın başka bir kısmında anlatılacaktır.

Proje geliştirme sürecinde tek developer ile ilerlendiği ve tüm mikroservisler aynı anda iteratif olarak geliştirildiğinden yazılım kolaylığı sağlaması adına her bir mikro servis ayrı solutionda yazılmamış, ortak bir solution içinde solution folderlar ile bölünmüştür. Birbirleri arasında build dependency barındırmadıkları için kolaylıkla farklı solutionlara bölünebilirler.

3.1 – VDB.MicroServices.X

X bağlamı ile ilgili tüm katmanları barındıran ana solution folder. İçerisindeki tüm DLLler X bağlamı içerisinde anlamlıdır, diğer mikro servislerle bir paylaşım içerisinde değildir.

3.1.1 – VDB.MicroServices.X.Concern

Cross-cutting concernlerin DLLlerini içeren solution folder.

3.1.1.1 – VDB.MicroServices.X.Concern.Constants

Konfigürasyonel olmayan ve uygulamadaki implementasyonlar değişmediği sürece aynı değeri koruması istenen sabit değerler.

3.1.1.2 – VDB.MicroServices.X.Concern.Options

Bağlamdaki konfigürasyonel değerler config dosyalarından Options patterni ile okunmaktadır. Mevzubahis DLL ilgili Options sınıflarını barındırır.

3.1.2 – VDB.MicroServices.X.DB

Bağlam içerisindeki veri tabanı erişimiyle ilgili DLLleri barındıran solution folder.

3.1.2.1 – VDB.MicroServices.X.DB.Context

Bağlamdaki her veri tabanı için DataContext sınıfını barındıran DLL. Bu sınıfların standart implementasyonda VDB.Architecture.Data.Context içerisindeki base sınıftan türetilmesi beklenmektedir.

3.1.2.2 – VDB.MicroServices.X.DB.Repository

Bağlam içerisindeki her bir Entity'nin Repositorylerini barındıran DLL. Bu repositorylerin standart implementasyonda VDB.Architecture.Data.Repository içerisindeki base sınıflardan birinden türetilmesi beklenmektedir.

Eğer base sınıfların içerisindeki jenerik metotların karşılamadığı bir veri tabanı etkileşimi ihtiyacı varsa (Include patterni dışında bir join yazmak, özel sql çalıştırmak gibi) ilgili etkileşim bu katmandaki sınıflarda yazılmalıdır.

3.1.2.3 – VDB.MicroServices.X.DB.UnitOfWork

Uygulama kodunun repositorylere erişip transactionı yöneteceği Unit Of Work patterni implementasyonunu içeren DLL. Bağlamdaki her veritabanı için bir UnitOfWork olması beklenmektedir ve standart implementasyonda bu sınıflar VDB.Architecture.Data.UnitOfWork içerisindeki base sınıftan türemelidir.

3.1.3 – VDB.MicroServices.X.ExternalData

Bağlam kendi veritabanı, kendi cache'i ya da yerel konfigürasyon dosyaları dışında bir yerden veri okuyor ya da veri yazıyorsa ilgili iletişimler ExternalData katmanı üzerinden yapılmalıdır. Bu iletişimlere örnek olarak başka bir mikroservisin ya da dış sistemin servislerini çağırmak, Message Broker'a bir şeyler yazmak ve dosya sistemine erişmek verilebilir. Bu iletişimleri yöneten DLLleri mevzubahis solution folder içerir.

3.1.3.1 – VDB.MicroServices.X.ExternalData.Model

Dışarıya giden ya da dışardan gelen veri modellerini barındıran DLL.

3.1.3.2 – VDB.MicroServices.X.ExternalData.Manager

Dış sistemlerle iletişimi sağlayan DLLleri barındıran solution folder. İçerisinde her bir iletişim yöntemi için bir proje olması beklenmektedir. Web servis çağrımları için .Service, Message Brokerlara mesaj göndermek için .MessageBroker gibi.

3.1.3.2.1 – VDB.MicroServices.X.ExternalData.Manager.Contract

Bir sistemle entegrasyon t anında web servis üzerinden ile yapılıyorken t+1 anında başka bir entegrasyon yöntemine geçebilir. Bu esnekliği sağlamak adına her bir dış sistem için bu DLLde bir interface olması beklenmektedir.

3.1.3.2.2 – VDB.MicroServices.X.ExternalData.Manager.Service

Web servis entegrasyonu ile bağlanılan dış sistemlerin entegrasyonlarının yazıldığı DLL. Her sistem için bir YServiceManager şeklinde ServiceManager adı verilen sınıflar olması beklenir. Bu ServiceManagerların request ve responselarını uygulama koduna sızdırmaması, uygulama kodundan gelen DTO ya da Entity objelerden requestlerini üretip responselarını da DTO ya da Entity objelerine çevirmesi beklenir.

3.1.3.2.3 – VDB.MicroServices.X.ExternalData.Manager.MessageBroker

Message Broker'a mesaj göndermek için kullanılan sınıfları içeren DLL. Her bir queue için bir YMessageBroker şeklinde sınıf olması beklenir.

3.1.4 – VDB.MicroServices.X.Manager

Bağlam içerisindeki tüm süreçleri ve business kodları yöneten katmanları barındıran solution folder.

3.1.4.1 – VDB.MicroServices.X.Manager.Business

Süreçlerin implementasyonlarının yapıldığı BusinessManager adı verilen sınıfları içeren katman. Controller ve Workerlar doğrudan bu katmandaki metotları çağırarak iş süreçlerini başlatırlar.

3.1.4.2 – VDB.MicroServices.X.Manager.Mapper

Bağlam içindeki veri dönüşümlerini gerçekleştiren katman. Exchange – Entity – DTO – ExternalModel tipleri arasında ve bu tiplerin kendileri içinde dönüşümler yapabilir.

AutoMapper kullanarak yapılabilecek basit dönüşümler için _AutoMapperProfiles klasörü içinde bağlamon MappingProfile'larının tanımlanması beklenir.

AutoMapper ile yapılamayan/yapılmak istenmeyen dönüşümler için özel Mapperlar tanımlayabilir.

3.1.4.3 – VDB.MicroServices.X.Manager.Operation

BusinessManagerlar persistance katmanıyla doğrudan iletişime geçmemektedirler ve bir veriyi okumaları ya da kaydetmeleri gerektiğinde Operation katmanındaki her bir Entity için oluşturulmuş YEntityOperations sınıfındaki ilgili metotları çağırmaktadırlar. Operation katmanındaki sınıflar da ilgili UnitOfWorkler aracılığı ile istenen operasyonu veri okuma/yazma kurallarına göre gerçekleştirirler.

Bu katman Repository katmanı ile aynı işi yapıyor gibi görünüp gereksiz görülebilir. Bir YRepository ve bir YOperations arasındaki en temel fark YRepository DataContext'e de

erişebiliyorken YOperations Data Contexte erişememekte, UnitOfWork üzerinden YRepository'e erişebilmektedir. Bu da yazılım esnasında DataContext üzerine doğrudan sorgu yazılması gerektiğinde nereye gidilmesi gerektiğini, bu sorguların tüketilmesi istendiğinde nereye gidilmesi gerektiğini keskin bir çizgiyle ayırmaktadır. Yazılımcı Repository üzerinde bir metot yazdığında "Yazdığım kodlar IQueryable üzerinden doğrudan SQLe çevrilecek" farkındalığını daha rahat yaşayabilecektir.

3.1.5 – VDB.MicroServices.X.Model

Bağlam içerisindeki veri modellerinin DLLlerini barındıran solution folder.

3.1.5.1 – VDB.MicroServices.X.Model.DTO

Basit data transfer objectleri içeren DLL. Bir değer veri tabanı modeli olan Entity objesi ile ifade edilebiliyorsa DTO yaratmaktan kaçınıp BusinessManager üzerinde doğrudan Entity obje kullanılarak işlem yapılmalıdır. DTOlar dış servislere request olarak ya da clientlara response olarak kesinlikle verilmemelidir.

3.1.5.2 – VDB.MicroServices.X.Model.Entity

Veri tabanı modellerini karşılayan Entity sınıfları içeren DLL. Bir Entity üzerinde fonksiyonlar ya da veritabanında karşılığı olmayan alanlar yazılacaksa Entity için partial class mantığıyla iki sınıf yaratılıp birinde sadece veri tabanı alanlarını diğerinde ek property ve metotların barındırılması beklenmektedir.

3.1.5.3 – VDB.MicroServices.X.Model.Exchange

Controller, Worker ve BusinessManagerların Request/Response ikililerini içeren DLL. Controllera gelen request herhangi bir mappinge uğramadan doğrudan BusinessManager'a verilebilir, BusinessManager'dan dönen bir response herhangi bir değişime uğramadan doğrudan Controller tarafından clienta dönülebilir.

3.1.6 – VDB.MicroServices.X.Tests

Bağlam içerisindeki testleri içeren solution folder. Her test tipi için .Unit, .Integration şeklinde test projeleri olması beklenmektedir.

3.1.6.1 – VDB.MicroServices.X.Tests.Unit

Birim testleri içeren test projesi.

3.1.7 – VDB.MicroServices.X.Web

Mikroservise Web üzerinden giriş noktası vazifesi gören projeleri barındıran solution folder. Her bir entegrasyon tipi için .API, .WCF, .GRPC gibi projeler içermesi beklenir.

3.1.7.1 – VDB.MicroServices.X.Web.API

Mikroservise REST APIlarla erişmek için giriş noktası. Standart ASP.NET 5 projelerinden oluşmaktadırlar.

3.1.8 – VDB.MicroServices.X.Worker

Bağlam içerisindeki background workerları içeren solution folder. Her bir farklı background task için bir proje içermesi beklenir.

3.1.8.1 – VDB.MicroServices.X.Worker.MessageConsumer

MessageBrokera bağlanıp consumerları ayağa kaldırarak bağlamı ilgilendiren queueleri dinlemeye başlayan background worker.

3.1.8.2 – VDB.MicroServices.X.Worker.Downloader

Bağlamın periyodik olarak veritabanına veri beslemesi yapması gerektiği durumlarda ilgili beslemeyi yöneten background worker.

Solution İçerisindeki Servis ve Workerların Sorumlulukları

Solution GateWay hariç 5 servis ve 3 background workerdan oluşmaktadır. Bu kısımda bu projelerin iş süreçleri içindeki görevleri üst düzeyden anlatılacaktır. Her bir servisin içerisindeki actionların dökümantasyonu için Swagger arayüzüne başvurulmalıdır.

VDB.MicroServices.Auth.Web.API

Her servis kendi authenticationına sahip değildir, authentication ve authorization Ocelot ve Auth.Web.API sayesinde merkezileştirilmiştir. Clientlardan Ocelot config'inde authentication gerekli olarak deklare edilmiş bir endpointe gelmeden önce Auth.Web.API üzerinden Bearer Token almaları beklenmektedir. İlgili mikro servis LDAP üzerinden kişiyi authenticate ettikten sonra bir JWE (encrypted JWT) üretir ve geriye bu JWE ile kişinin OrganizationalUnit değerini döner. GateWay authorization gerekli servisler için JWE bazlı authorization yapmaktadır fakat client uygulamaların da kişinin yetkisini anlayabilmesi için OU değeri dönülür.

VDB.MicroServices.CVEData.Web.API

Zafiyet verilerinin depolandığı CVEDataDB veritabanını sorgulayan REST APIı içerir. İçerisindeki tüm servisler authentication gerektirmektedir.

VDB.MicroServices.CVEData.Worker.Downloader

Uygulama ilk ayağa kalktığı andan itibaren aktif olan bir background tasktır ve görevi zafiyet sorgulaması yaparak CVEDataDB'yi güncel tutmaktır. CVEDownloadLogs tablosundaki en güncel CVEDataTimestamp değerini okuyarak (eğer kayıt bulunamazsa /Configuration/CVEData/CVEDownloaderSettings.json içerisindeki DefaultDownloadStartingDateValue değerini başlangıç tarihi olarak kabul eder) bu tarihten itibaren verileri indirerek sistemi güncel hâle getirir.

Eğer ilgili CVEDownloadLogs kaydının IsDownloadBySearch flagi false ise sistem güncel veride değildir ve anlık arama ile kayıt indirmez, yıllık veri dönen servisi kullanarak verileri yıl yıl kaydeder. Yıl yıl taraması bittikten sonra anlık arama servisiyle en son yaptığı isteğin üstünden bu yana geçen verileri de alır ve sistemi güncel hâle getirir. Bu aşamadan sonra worker /Configuration/CVEData/CVEDownloaderSettings.json dosyasında milisaniye cinsinden belirtilmiş DownloadFrequency değeri kadar süre geçtikten sonra tekrar çalışır ve sistem açık olduğu süre boyunca sürekli en son sorgulamasından bu yana güncelleme görmüş kayıtları servisten çekerek CVEDataDB'de saklar.

Her güncellemeden sonra eğer sistemde yeni bir zafiyet kaydı yaratıldıysa VulnerabilityDetector.Web.API servisinin Report/CreateReport() action'ı çağrılır ve yeni kayıtların yeni zafiyetlere sebep olup olmadığını görmek adına tarama başlatılır.

VDB.MicroServices.InventoryManager.Web.API

Ürün kataloğu verilerini tutan InventoryManagerDB veritabanıyla etkileşimi sağlayan REST APIları içerir. İçerisindeki tüm servisler authentication gerektirmektedir. GET servisler authorization gerektirmezken PUT, POST ve DELETE servisleri VDBAdmins yetkisine sahip bir kullanıcı gerektirmektedir.

VDB.MicroServices.NotificationCenter.Web.API

Üretilen raporların hangi alıcılara ne şekilde iletileceğinin yönetimini sağlayan servisleri içerir. İçerisindeki tüm servisler authentication gerektirmektedir. GET servisler authorization gerektirmezken PUT, POST ve DELETE servisleri VDBAdmins yetkisine sahip bir kullanıcı gerektirmektedir.

VDB.MicroServices.NotificationCenter.Worker.MessageConsumer

Uygulama ile beraber ayağa kalkıp /Configuration/MessageBroker/MessageBrokerSettings.json dosyasındaki NotificationQueueName değeriyle belirtilmiş mesaj kuyruğunu dinlemeye başlar.

Notifikasyon kuyruğuna mesajlar "X mail adresine mail at" şeklinde gelmemektedir. NotificationCenter Bağlam → Tipler → Alıcılar şeklinde çalışmaktadır ve kuyruğa mesajlar "X bağlamına Y mesajını yayınla" şeklinde gelmektedir. NotificationCenter bu mesajdaki X bağlamını NotificationContexts tablosundan bulup (bulamazsa hata verecektir) bağlam için NotificationTypes tablosuna giderek ne tipte (mail, sms, jira vs vs) notifikasyonlar gönderileceğini öğrenir. Ardından

NotificationAudiences tablosundan ilgili bağlam ve tip için tanımlı alıcı varsa alıcıyı öğrenip notifikasyon gönderimini gerçekleştirir. Sistem şu anda sadece VDBReport isimli bağlamı ve Email notifikasyon tipini desteklemektedir. Kullanıcılara ön yüzden bağlam ve tip tanımlama yetkisi verilmemiştir, bu bağlam ve tipe alıcı ekleyebilmektedirler.

Bu süreçte herhangi bir hata olması durumunda worker ayarlarda belirtilmiş Redis cachede mesajın tekil idsi ile bir hash yaratır ve değerini 1 arttırır. Ardından cachedeki bu değeri /Configuration/MessageBroker/MessageBrokerSettings.json dosyasındaki RetryCount ile kıyaslar. Eğer cachedeki değer dosyadaki değerden küçükse mesaj tekrar kuyruğa sokulur ve yeniden denenir. Aksi hâlde mesaj yeniden kuyruğa sokulmaz.

VDB.MicroServices.VulnerabilityDetector.Web.API

Zafiyet raporu verilerini tutan VulnerabilityDetectorDB veritabanıyla etkileşimi sağlayan REST APIları içerir. İçerisindeki tüm servisler authentication gerektirmektedir. GET servisler authorization gerektirmezken CreateReport hariç PUT, POST ve DELETE servisleri VDBAdmins yetkisine sahip bir kullanıcı gerektirmektedir.

VDB.MicroServices.VulnerabilityDetector.Worker.MessageConsumer

Uygulama ile beraber ayağa kalkıp /Configuration/MessageBroker/MessageBrokerSettings.json dosyasındaki ReportQueueName değeriyle belirtilmiş mesaj kuyruğunu dinlemeye başlar.

Kuyruğa mesajlar VulnerabilityDetector.Web.API'ın CreateReport'unda üretilmiş bir ReportID değeri ile gelir ve worker ilgili idye karşılık gelen VulnerabilityReport kaydının detaylarını doldurmaya başlar. İlk işlem olarak InventoryManager.Web.API üzerinden sistemdeki ürün kataloğu alınır. Ardından katalogdaki her bir ürün için CVEData.Web.API'ın Firma/Ürün/Versiyon bilgileri ile arama yapan servisi çağrılır ve eşleşen zafiyetler bulunur. Ardından zafiyet bulunduysa ReportContents tablosunda ilgili bulgular kaydedilir ve zafiyet raporu HTML formatına çevrilerek Base64 encoding ile VulnerabilityReport kaydının Base64ReportData alanında saklanır. Ardından /Configuration/MessageBroker/MessageBrokerSettings.json dosyasındaki NotificationQueueName değeri ile belirtilmiş kuyruğa ilgili rapor verileri mesaj olarak gönderilir ve notifikasyon iletilmesi sağlanır.

Süreç esnasında herhangi bir hata yaşanırsa VulnerabilityReport kaydının ErrorMessage kısmına kullanıcıya göstermek için hata mesajı, ErrorDetail kısmına ise hata tespitinin kolaylaşması için Stack Trace yazılır.

Bu süreçte herhangi bir hata olması durumunda worker ayarlarda belirtilmiş Redis cachede mesajın tekil idsi ile bir hash yaratır ve değerini 1 arttırır. Ardından cachedeki bu değeri /Configuration/MessageBroker/MessageBrokerSettings.json dosyasındaki RetryCount ile kıyaslar. Eğer cachedeki değer dosyadaki değerden küçükse mesaj tekrar kuyruğa sokulur ve yeniden denenir. Aksi hâlde mesaj yeniden kuyruğa sokulmaz.