LLD

Low Level Design

The comprehensive technical document for architectural design and code structure

Engineering & Operations Team (WSS)

Technical Application Profile

Project : NotificationPublisher Ecosystem

Paths (Endpoints) : Email (Single/Bulk), SMS, WhatsApp, FCM

Data protection (Secrets) : HashiCorp Vault KV v2

Implementation stage : Production - OKE Cluster

Core Tech : ASP.NET Core 8.0, RabbitMQ

1. Dependency Injection and Initialization

The basic logic in PublisherOperations that coordinates all operations:

```
public class PublisherOperations : IPublisherOperations
{
    private readonly IPublisherHelper publisherHelper;
    private readonly IPublisherCore publisherCore;
    private readonly IWhatsAppService whatsAppService;

    public PublisherOperations(
        IPublisherHelper helper, IPublisherCore core,
        IWhatsAppService waService)
    {
        this.publisherHelper = helper;
        this.publisherCore = core;
        this.whatsAppService = waService;
    }
}
```

- IPublisherHelper: Interfaces with SQL Server via Dapper to manage quotas and insert records
- IPublisherCore: A wrapper for a RabbitMQ C# client that manages channel creation and continuous publishing
- IWhatsAppService: Builds Meta templates and generates signed HMAC file links

2. Email Push Pipeline

When an email request arrives it passes through a strict serial pipeline:

- Application Verification: GetApplicationData(AppId, MsgTypes.Email)
- Database logging: InsertEmailMessage() inserts a draft status and returns a unique Guid (MsgId)
- Availability Gateway: If the application is inactive it is marked as failed immediately
- If the mail setting is inactive: Returns EmailSettingNotActive and stops the flow
- Queue push: A MsgId is attached to the payload and sent to RabbitMQ via email_route
- Error Handling: Any unexpected failure is logged via Serilog

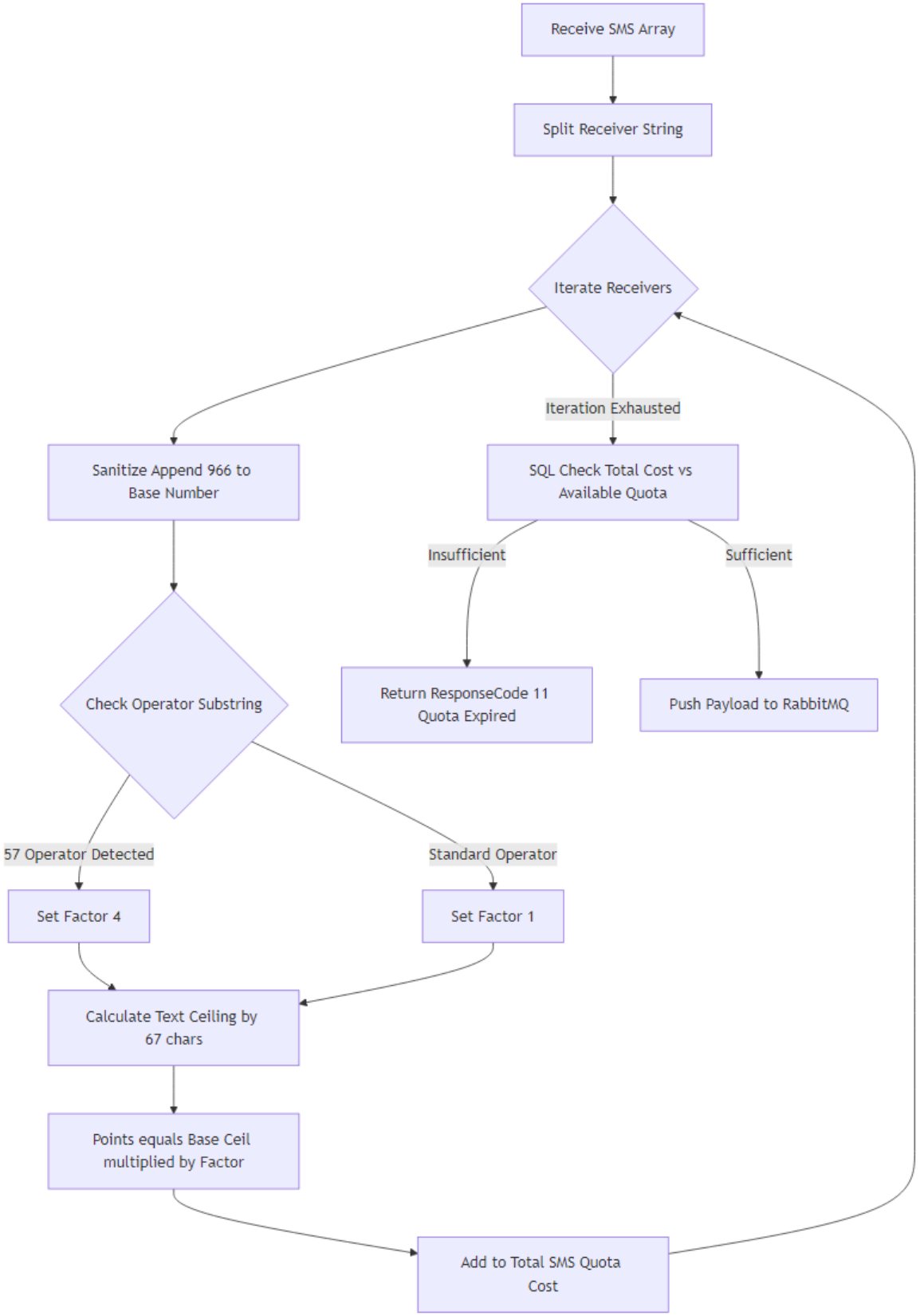
Bulk Email Pipeline

```
foreach (var item in data)
{
    var result = publisherHelper.GetApplicationData(item.AppId, ...);
    var MsgId = publisherHelper.InsertEmailMessage(...);
    // Validate -> continue on failure (don't halt batch)
    if (!result.IsActive) { publisherHelper.FailMessage(...); continue; }
    if (!result.IsSettingActive) { ... continue; }
    publisherCore.PublisherSend(result, ...);
}
```

Main design: Bulk mail uses continue to skip failed items. This guarantees partial success.

3. Complex SMS Point Algorithm

Unlike email, SMS requires complex text processing and quota checking:



SMS pricing chart with carrier factor and segmentation

- Backup sender customization: If data.Message.Sender is empty replace it with appData.Sender
- Batch numbers: a string of numbers separated by a comma -> ValidateRecieverNumberLengthAndGetListOfNumbers
- Iterative processing: Loop through each batch of receivers

CountMessage() cost calculation algorithm

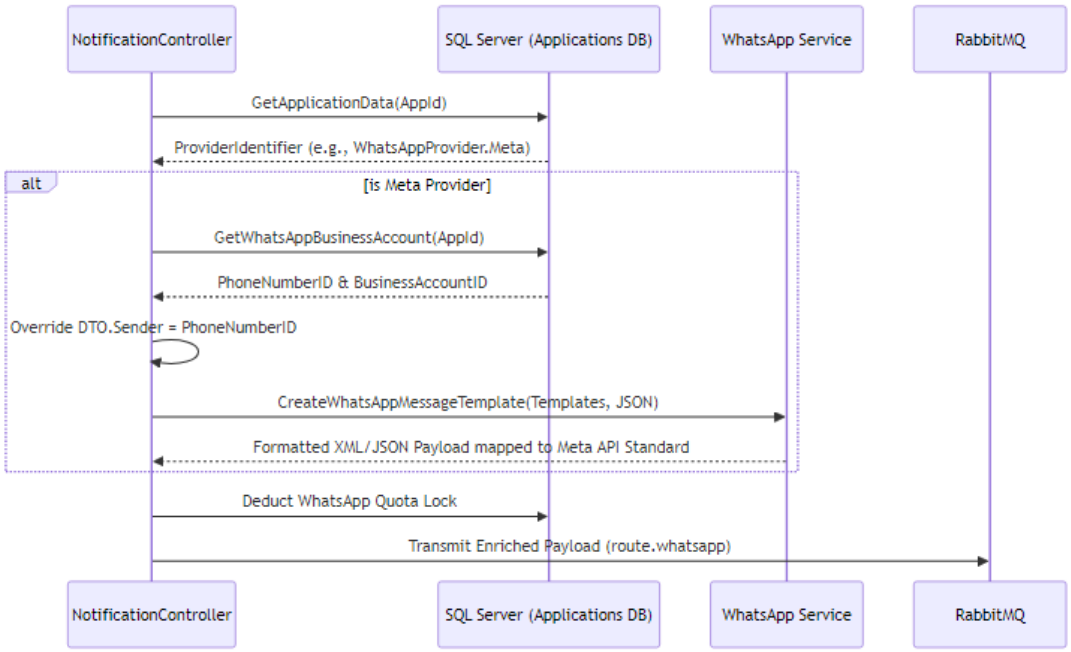
```
private int CountMessage(string receivers, string message)
{
    var receiverList = receivers.Split(",").ToList();
    var messageCount = 0;
    foreach (var receiver in receiverList)
    {
        // Normalize: 0551... -> 966551...
        var receiverNumber = receiver.StartsWith("966")
            ? receiver
            : $"966{(receiver.StartsWith("0")
                ? receiver.Remove(0, 1) : receiver)}";
        // Carrier factor: "57" = 4x, others = 1x
        var factorMessage =
            receiverNumber.Substring(3, 2) == "57" ? 4 : 1;
        // Segmentation: <= 70 = 1 part, else ceil(len/67)
        messageCount += (message.Length <= 70
            ? 1
            : (int)Math.Ceiling((double)message.Length / 67.0))
            * factorMessage;
    }
    return messageCount;
}
```

Normalization of phone numbers

Result	Conversion	Input
966551234567	Already contains the country code	966551234567
966551234567	Remove 0 + add 966	0551234567
966551234567	Add 966	551234567

4. WhatsApp Template Construction for Meta (WhatsApp Template Construction)

If the message targets WhatsApp Provider ID = Meta:



WhatsApp Templates Flow Chart with Meta Business API

- Provider Solution: Query GetWhatsAppBusinessAccount to get the settings
- Anonymization: Override data.Message.Sender with official PhoneNumberID
- Template condition: Process the template only when a TemplateName or TemplateId is provided
- Content conversion: CreateWhatsAppMessageTemplate() builds Meta-compatible JSON

Structure of template components

Purpose	sub_type	Component type
Text, image, video or document	-	header
Dynamic text variables	-	body
Dynamic URL button with index 0	url	Button

JSON example of the generated template

```
{
   "name": "otp_verification",
   "language": { "code": "ar" },
   "components": [
      {
         "type": "header",
         "parameters": [
            { "type": "text", "text": "Verification" }
         ]
      },
      {
         "type": "body",
         "parameters": [
            { "type": "text", "text": "1234" },
            { "type": "text", "text": "5 minutes" }
         ]
      },
      {
         "type": "Button",
         "sub_type": "url",
         "index": "0",
         "parameters": [
            { "type": "text",
              "text": "https://verify.example.com/1234" }
         ]
      }
   ]
}
```

5. FCM (Push Notification Flow)

The Firebase Cloud Messaging pipeline follows the same pipeline as Email:

```
Request -> GetApplicationData(AppId, MsgTypes.PushNotification)
      -> InsertPushNotificationMessage()
      -> IsActive check
      -> IsSettingActive check (PushNotificationSettingNotActive)
      -> PublisherCore.PublisherSend() -> RabbitMQ FCM queue
```

Key difference: Quotas are not checked for FCM - Firebase manages rate control internally.

6. Publishing engine in RabbitMQ (PublisherCore Pipeline)

PublisherSend() performs nine sequential steps:

the details	Process	Step
Secure verification of connection and channel	ConnectionExists()	1
Enable publisher confirmations	ConfirmSelect()	2
Declaring the exchange as permanent	ExchangeDeclare()	3
Queue declaration: durable + lazy	QueueDeclare()	4
Interconnect the queue via routing switch	QueueBind()	5
Serialize the payload to JSON -> UTF-8 bytes	UTF8 Encoding	6
Deployed with DeliveryMode = 2 (permanent)	BasicPublish()	7
Block until broker confirmation or timeout expires	WaitForConfirms(5s)	8
On confirmation -> Update DB. When finished -> PushFail	UpdateStatus()	9

7. FileDownloadController

Secure proxy between external clients (Meta servers) and private S3 bucket:

Failure response	Examination	Step
400 Bad Request	fileName empty	1
403 Forbidden	expires or missing signature	2
410 Gone	currentTime > expires (terminal interface)	3
403 Forbidden	signature != expectedSignature (forged)	4
404 Not Found	The object does not exist in S3	5
200 OK + file stream	success	6

8. Excel bulk import module (ExcelController)

- Parse file: ParseImportFile(stream, messageType) extracts rows into ImportRecipient objects
- Salutation solution: Query GetSalutation(appId, language) with default Dear or Dear
- Message composition: {greeting} {name}, {message content}
- Channel Send: Call appropriate PushXxxMessage() based on MessageType
- Error Reports: Failed rows are reported with IsValid=false and ErrorMessage

```
// ExcelImportReport Response
{
   "totalRows": 100,
   "successCount": 95,
   "failedCount": 5,
   "details": [
      { "name": "Ahmed", "isValid": true },
      { "name": "Test", "isValid": false,
        "errorMessage": "Invalid number" }
   ]
}
```

9. Messages & Statistics inquiry

Purpose	Stored Procedure
SMS log with pagination	SP_GetSMSMessages
WhatsApp history with pagination	SP_GetWhatsAppMessages
Email log with pagination	SP_GetEmailMessages
Total/Sent/Failed for each channel	SP_GetMessagesStats