PhonePe Payment Integration Technical Document

Document Control

- Version: 1.0
- Date: March 29, 2025
- Status: Implementation Guide

1. Introduction

This technical document outlines the implementation of PhonePe payment gateway integration with webhook handling for transaction capture. The document covers the complete payment flow from initiation to transaction confirmation, including both production and UAT (User Acceptance Testing) environments.

2. System Architecture

2.1 Components

- Payment Controller: Handles payment initiation and status verification
- PhonePe Auth Service: Manages OAuth token retrieval and caching
- Webhook Controller: Processes payment callbacks from PhonePe
- UI Component: Embeds PhonePe payment page
- ngrok: Tunneling service for local webhook testing

2.2 Environment Configurations

- UAT (Sandbox):
  - Base URL: https://api-preprod.phonepe.com/apis/pg-sandbox
  - Merchant ID: [UAT_MERCHANT_ID]
  - SaltKey: [UAT_SALT_KEY]

- Production:
  - Base URL: https://api.phonepe.com/apis/hermes
  - Merchant ID: [PRODUCTION_MERCHANT_ID]

- Salt Key: [PRODUCTION_SALT_KEY]

3. PhonePe Authentication

3.1 OAuth Token Service

The PhonePeAuthService class handles authentication token management:

```
using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading;
using System.Threading.Tasks;
using Newtonsoft.Json;
public class PhonePeAuthService
{
    private static string _accessToken;
    private static DateTime _tokenExpiry;
    private static readonly HttpClient _client = new HttpClient();
    private static Timer _tokenRefreshTimer;
    private readonly IConfiguration _configuration;
    public PhonePeAuthService(IConfiguration configuration)
    {
        // Set the default headers
                                                _client.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));
```

```
    // Start the token refresh process when the service is initialized
    StartTokenRefreshProcess();
    _configuration = configuration;
}

public async Task<string> GetAccessTokenAsync()
{
    // Check if we already have a valid token
    //if (!string.IsNullOrEmpty(_accessToken) && DateTime.UtcNow < _tokenExpiry)
    //{
    //    return _accessToken;
    //}
    try
    {
        // If the token is invalid, fetch a new one
        var token = await FetchNewAccessTokenAsync();
        return token;
    }
    catch (Exception ex)
    {
        throw new Exception($"Error fetching access token: {ex.Message}");
    }
}
```

```
    private async Task<string> FetchNewAccessTokenAsync()
    {
        try
        {
            var formData = new List<KeyValuePair<string, string>>
            {
                                                    new KeyValuePair<string, string>("client_id",
$"{_configuration["PhonePeSettings:MerchantId"] }"),
                                                    new KeyValuePair<string, string>("client_secret",
$"{_configuration["PhonePeSettings:SaltKey"] }"),
                                                    new KeyValuePair<string, string>("client_version",
$"{_configuration["PhonePeSettings:ClientVersion"] }"),
                                                    new KeyValuePair<string, string>("grant_type",
$"{_configuration["PhonePeSettings:GrantType"] }")
            };

            var content = new FormUrlEncodedContent(formData);

            // Set the proper content-type header
                                                    content.Headers.ContentType = new
MediaTypeHeaderValue("application/x-www-form-urlencoded");

                                                    var response = await
_client.PostAsync($"{_configuration["PhonePeSettings:Tokenurl"]}", content);
            var responseContent = await response.Content.ReadAsStringAsync();
```

```
            if (response.IsSuccessStatusCode)
            {
                var jsonResponse = JsonConvert.DeserializeObject<dynamic>(responseContent);
                if (jsonResponse?.access_token != null && jsonResponse?.expires_at != null)
                {
                    _accessToken = jsonResponse.access_token;
                        _tokenExpiry = DateTime.UtcNow.AddSeconds((int)jsonResponse.expires_at -
60); // Subtract 60 sec buffer
                    return _accessToken;
                }
                else
                {
                    throw new Exception("Failed to get access token: Missing expiration details.");
                }
            }
            else
            {
                throw new Exception($"Failed to get access token: {responseContent}");
            }
        }
        catch (Exception ex)
        {
            throw new Exception($"Error fetching access token: {ex.Message}");
        }
    }
```

```
    private void StartTokenRefreshProcess()
    {
        // Refresh the token every 5 minutes (before it expires)
        _tokenRefreshTimer = new Timer(async _ =>
        {
            if (_tokenExpiry < DateTime.UtcNow)
            {
                await FetchNewAccessTokenAsync();
                Console.WriteLine("✅ Token refreshed successfully.");
            }
        }, null, TimeSpan.Zero, TimeSpan.FromMinutes(5)); // Check every 5 minutes
    }
}
```

3.2 Authentication Flow

1. Check if a valid cached token exists
2. If not, request a new token using client credentials
3. Cache the token until expiry (with 60-second buffer)
4. Return the token for API requests

4. Payment Initiation Process

4.1 Payment Request Structure

```
public class PaymentRequest
{
```

```
 public int Amount { get; set; }
    public string plan { get; set; }
    public string Currency { get; set; }
    public string RedirectUrl { get; set; }
    public string Userid { get; set; }
}
```

4.2 Payment Initiation Flow

1. Receive payment request from client application
2. Convert non-INR currencies to INR if needed
3. Generate merchant order ID and transaction ID
4. Create payment payload with required parameters
5. Encode payload in Base64 format
6. Generate checksum for authentication
7. Send request to PhonePe API
8. Return redirect URL to client for payment page embedding

4.3 Implementation

```
    [HttpPost("initiate")]
    public async Task<IActionResult> InitiatePayment([FromBody] PaymentRequest request)
    {
        try
        {
            if (request.Currency != "INR")
            {
                request.Amount = await ConvertToINR(request.Amount, request.Currency);
            }
```

```
string accessToken = await _phonePeAuthService.GetAccessTokenAsync();
Console.WriteLine("📌 Access Token: " + accessToken);

var transactionId = $"MT{DateTime.UtcNow.Ticks}";

var payload = new
{
    merchantOrderId = transactionId,
    amount = request.Amount * 100,
    expireAfter = 1200,
    metaInfo = new
    {
        udf1 = request.Currency,
        udf2 = request.plan,
        udf3 = request.Userid,
        udf4 = request.packagetype
    },
    paymentFlow = new
    {
        type = "PG_CHECKOUT",
        message = "Payment message used for collect requests",
        merchantUrls = new
        {
```

```
                redirectUrl = request.RedirectUrl
            }
        }
    };

                var content = new StringContent(JsonConvert.SerializeObject(payload), Encoding.UTF8, "application/json");
    using var client = new HttpClient();
    client.DefaultRequestHeaders.Add("Authorization", $"O-Bearer {accessToken}");

                                                    string apiUrl = $"{_configuration["PhonePeSettings:BaseUrl"]}{_configuration["PhonePeSettings:Endpoint"]}";
    Console.WriteLine("📡 Hitting API: " + apiUrl);

    var response = await client.PostAsync(apiUrl, content);
    var responseData = await response.Content.ReadAsStringAsync();

    Console.WriteLine("📥 Response: " + responseData);

    if (!response.IsSuccessStatusCode)
    {
            return StatusCode((int)response.StatusCode, new { success = false, message = "PhonePe call failed", details = responseData });
    }
```

```
            var jsonObject = JObject.Parse(responseData);


            var accesToken = await _phonePeAuthService.GetAccessTokenAsync();


            return Ok(new
            {
                OrderId = jsonObject["orderId"]?.ToString(),
                RedirectUrl = jsonObject["redirectUrl"]?.ToString(),
                AccessToken = accesToken,
                TransactionId = transactionId
            });
        }
        catch (Exception ex)
        {
            Console.WriteLine("❌ Exception: " + ex.ToString());
                return StatusCode(500, new { success = false, message = "Internal error", error =
ex.Message });
        }
    }
```

5. Payment Status Verification and Invoice Generation

5.1 Status Check Implementation

API

```
[HttpPost("status/{merchantOrderId}")]

    public async Task<IActionResult> GetPaymentStatus(string merchantOrderId)

    {

        if (string.IsNullOrEmpty(merchantOrderId))

        {

            return BadRequest(new { message = "MerchantTransactionId is required" });

        }


                                                    string apiEndpoint =
$"{_configuration["PhonePeSettings:StatusEndpoint"]}/{merchantOrderId}/status";

        using var client = new HttpClient();

                        client.DefaultRequestHeaders.Add("Authorization", $"O-Bearer {await
_phonePeAuthService.GetAccessTokenAsync()}");


                                                    var response = await
client.GetAsync($"{_configuration["PhonePeSettings:BaseUrl"]}{apiEndpoint}");


        if (response.StatusCode == HttpStatusCode.NoContent)

        {

            return Ok(new { success = false, status = "No content returned from PhonePe" });

        }


        var responseData = await response.Content.ReadAsStringAsync();
```

```
        if (string.IsNullOrWhiteSpace(responseData))
        {
            return Ok(new { success = false, status = "Empty response from PhonePe" });
        }

        var jsonResponse = JsonConvert.DeserializeObject<dynamic>(responseData);
        if (jsonResponse != null && jsonResponse.success == true)
        {
            return Ok(new { success = true, status = jsonResponse.data.state });
        }

        JObject payload = JObject.Parse(responseData);

        if(payload["state"]?.ToString() == "COMPLETED")
        ProcessPaymentSuccess(payload);



        return Ok(responseData);
    }
```

UI

```
const checkPaymentStatus = async (merchantOrderId: any, accessToken: any) => {
    try {
      const response = await fetch(
```

```
  `${apiurlPhonepeAcc}/status/${merchantOrderId}`,
  {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "Authorization": `O-Bearer ${accessToken}`,
    }
  }
);

if (!response.ok) throw new Error("Status check failed");

const data = await response.json();
console.log("Full payment status data:", data);

// Save to localStorage
// localStorage.setItem("paymentResponse", JSON.stringify(data));
localStorage.removeItem("merchantOrderId");
localStorage.removeItem("accessToken");

setPaymentStatus(data.state);

if (data.state === "COMPLETED") {
  if (
```

```
        data.paymentDetails &&
        Array.isArray(data.paymentDetails) &&
        data.paymentDetails.length > 0
      ) {

        const now = new Date();
        const formattedDateTime = now.getFullYear() + '-' +
          String(now.getMonth() + 1).padStart(2, '0') + '-' +
          String(now.getDate()).padStart(2, '0') + ' ' +
          String(now.getHours()).padStart(2, '0') + ':' +
          String(now.getMinutes()).padStart(2, '0') + ':' +
          String(now.getSeconds()).padStart(2, '0');

        console.log(formattedDateTime);

              downloadInvoice("Nan", data.orderId, formattedDateTime, data.metaInfo.udf2,
data.amount)
        toast.toast({
          title: "Success",
          description: `Payment succeeded for ${data.metaInfo.udf2}`,
        });
      } else {
        console.warn("Missing paymentDetails in responseData");
      }
    } else if (data.state === "PENDING") {
```

```
      // Optionally add retry limit or spinner here
      setTimeout(() => checkPaymentStatus(merchantOrderId, accessToken), 5000);
    } else {
      toast.toast({
        title: "Payment Failed",
        description: `State: ${data.state}`,
      });
    }

  } catch (err) {
    console.error("Status check error:", err);
    toast.toast({
      title: "Error",
      description: "Failed to verify payment status",
    });
  }
};
```

5.2 Invoice Generation Implementation

UI:

```
const downloadInvoice = async (
  id: string,
```

```
 paymentId: string,
    transactionDate: string,
    description: string,
    amount: string
  ) => {
    debugger;

    try {
      if (id == "") {
       id = "Nan";
      }

     if (id == "Nan") {
       const logoBase64 =
         "paster your LOGO as base64 string here eg :-

AAAABHNCSVQICAgIfAhkiAAAAAlwSFlzAAAuIwAALiMBeKU. . . . .
  ";

       const htmlContent = `<div style="font-family: Arial, sans-serif; padding: 20px;
max-width: 800px; margin: auto; border: 1px solid #ccc; border-radius: 8px;">

    <!-- Header -->
    <div style="display: flex; justify-content: space-between; align-items: center;
margin-bottom: 30px;">
      <div style="max-width: 65%;">
        <div style="font-size: 22px; font-weight: bold;">AGNO INTEL PRIVATE
LIMITED</div>
        <div style="margin-top: 5px; font-size: 12px; line-height: 1.5;">
          Address: Manasarovar, #1, 9C/10A, Second Floor Second Street,<br />
          Ayodhya Colony, Velachery, Chennai, Tamil Nadu 600042
        </div>
```

```
    </div>
    <div style="max-width: 30%; text-align: right;">
      <img src="${logoBase64}" alt="Company Logo" style="width: 120px; height: auto;" />
    </div>
  </div>

  <!-- Invoice Title -->
  <h1 style="text-align: center; font-size: 26px; margin-bottom: 30px;">Invoice</h1>

  <!-- Invoice Metadata -->
  <div style="font-size: 14px; margin-bottom: 30px;">
    <p style="margin: 5px 0;"><strong>Order ID:</strong> ${paymentId}</p>
    <p style="margin: 5px 0;"><strong>Date of issue:</strong> ${transactionDate}</p>
    <p style="margin: 5px 0;"><strong>Date due:</strong> ${transactionDate}</p>
  </div>

  <!-- Description Table -->
  <table style="width: 100%; border-collapse: collapse; margin-bottom: 25px; font-size: 14px;">
    <thead>
      <tr style="background-color: #f5f5f5;">
        <th style="border: 1px solid #ddd; padding: 10px; text-align: left;">Description</th>
        <th style="border: 1px solid #ddd; padding: 10px; text-align: right;">Amount</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td style="border: 1px solid #ddd; padding: 10px;">${description}</td>
        <td style="border: 1px solid #ddd; padding: 10px; text-align: right;">${(Number(amount) / 100).toLocaleString("en-IN", {
          style: "currency",
```

```
            currency: "INR",
          })}</td>
        </tr>
      </tbody>
    </table>

    <!-- Total Section -->
    <div style="text-align: right; font-size: 15px; line-height: 1.8;">
      <p><strong>Subtotal:</strong> ${(Number(amount) / 100).toLocaleString("en-IN", {
        style: "currency",
        currency: "INR",
      })}</p>
      <p><strong>Total:</strong> ${(Number(amount) / 100).toLocaleString("en-IN", {
        style: "currency",
        currency: "INR",
      })}</p>
      <h2 style="margin-top: 10px;">Amount Due: ${(Number(amount) /
100).toLocaleString("en-IN", {
        style: "currency",
        currency: "INR",
      })}</h2>
    </div>
  </div>
  `;

      // 🛠️ Create a hidden container to render HTML string
      const element = document.createElement("div");
      element.innerHTML = htmlContent;
      document.body.appendChild(element); // Temporarily add to DOM (required for
html2pdf)
```

```
      const opt = {
        margin: 0.5,
        filename: `invoice.pdf`,
        image: { type: "jpeg", quality: 0.98 },
        html2canvas: { scale: 2 },
        jsPDF: { unit: "in", format: "letter", orientation: "portrait" },
      };

      // 📥 Generate PDF from the element
      html2pdf()
      .set(opt)
      .from(element)
      .save()
      .then(() => {
          document.body.removeChild(element); // 🧹 Clean up after saving
        });
    } else {
      // Send GET request to your backend endpoint
      const response = await axios.get(
`${apiUrlAdvAcc}/DownloadInvoicePdf?invoiceId=${id}&paymentId=${paymentId}`,
        {
          responseType: "blob", // Ensure the response is treated as a file
        }
      );

      // Create a Blob from the response
      const blob = new Blob([response.data], { type: "application/pdf" });
      const downloadUrl = window.URL.createObjectURL(blob);

      // Create a link element to initiate download
```

```
      const link = document.createElement("a");
      link.href = downloadUrl;
      link.download = `Invoice_${paymentId}.pdf`;
      document.body.appendChild(link);
      link.click();
      link.remove();
    }
  } catch (error) {
    console.error("Error downloading receipt:", error);
    alert("Failed to download receipt.");
  }
};
```

6. UI Integration

6.1 Payment Page Integration

The PhonePe payment page can be embedded in your UI using the redirect URL returned by the initiate endpoint:

```
// React component example
function PaymentPage({ orderId, redirectUrl }) {
  return (
    <div className="payment-container">
      <h2>Complete Your Payment</h2>
      <iframe
        src={redirectUrl}
        width="100%"
        height="600px"
```

```
      frameBorder="0"
      allow="payment"
    ></iframe>
  </div>
 );
}
```

6.2 Alternative Redirect Approach

Instead of embedding the page, you can redirect the user:

```
window.location.href = redirectUrl;
```

7. Webhook Implementation

7.1 Webhook Controller

```
[ApiController]
[Route("api/webhook/phonepe")]
public class PhonePeWebhookController : ControllerBase
{
   private readonly IDbHandler _dbHandler;
   private readonly ILogger<PhonePeWebhookController> _logger;

                        public PhonePeWebhookController(IDbHandler dbHandler,
ILogger<PhonePeWebhookController> logger)
   {
```

```
        _dbHandler = dbHandler;
        _logger = logger;
    }

    internal static void ProcessPaymentSuccess(string responseData)
    {
        throw new NotImplementedException();
    }

    [HttpPost]
    [AllowAnonymous]
    public async Task<IActionResult> HandleWebhook()
    {
        _logger.LogInformation("Webhook hit: {time}", DateTime.UtcNow);

        // Read body ONCE and reuse
        string requestBody = await new StreamReader(Request.Body).ReadToEndAsync();

                                                _logger.LogInformation("Headers:      {headers}",
JsonConvert.SerializeObject(Request.Headers));
        _logger.LogInformation("Body: {body}", requestBody);

        string fallbackLogPath = @"C:\webHookLogs\log.txt";
        Directory.CreateDirectory(Path.GetDirectoryName(fallbackLogPath));
```

```
                    await System.IO.File.AppendAllTextAsync(fallbackLogPath,
JsonConvert.SerializeObject(Request.Headers));
        await System.IO.File.AppendAllTextAsync(fallbackLogPath, requestBody);

        try
        {
            if (string.IsNullOrWhiteSpace(requestBody))
            {
                _logger.LogError("Received an empty webhook payload.");
                return BadRequest(new { message = "Invalid webhook payload" });
            }

            JObject json;
            try
            {
                json = JObject.Parse(requestBody);
            }
            catch (JsonReaderException jsonEx)
            {
                    _logger.LogError(jsonEx, "Invalid JSON received in PhonePe webhook: {0}",
requestBody);
                return BadRequest(new { message = "Invalid JSON format" });
            }

            string eventType = json["event"]?.ToString();
```

```
var payload = json["payload"];

if (payload == null)
{
    _logger.LogWarning("Missing 'payload' in webhook.");
    return BadRequest(new { message = "Missing 'payload' in webhook." });
}

if (string.IsNullOrWhiteSpace(payload["orderId"]?.ToString()))
{
    _logger.LogWarning("Missing Order ID in payload.");
    return BadRequest(new { message = "Missing Order ID in payload." });
}

try
{
    switch (eventType)
    {
        case "checkout.order.completed":
            await ProcessPaymentSuccess(payload);
            break;
        case "checkout.order.failed":
            _logger.LogWarning($"Payment failed for Order ID: {payload["orderId"]}");
            break;
```

```
                case "pg.refund.failed":
                    _logger.LogWarning($"Refund failed for Order ID: {payload["orderId"]}");
                    break;
                case "pg.refund.completed":
                                        _logger.LogInformation($"Refund completed for Order ID:
{payload["orderId"]}");
                    break;
                case "pg.refund.accepted":
                                         _logger.LogInformation($"Refund accepted for Order ID:
{payload["orderId"]}");
                    break;
                default:
                    _logger.LogWarning($"Unhandled event type: {eventType}");
                    break;
            }
        }
        catch (Exception innerEx)
        {
            _logger.LogError(innerEx, $"Error while processing event type: {eventType}");
                    return StatusCode(500, new { message = $"Error while processing event:
{eventType}", details = innerEx.Message });
        }

        return Ok(new { message = "Webhook processed successfully" });
    }
```

```
        catch (Exception ex)
        {
            _logger.LogError(ex, "Unexpected error in webhook handler");
             return StatusCode(500, new { message = "Internal server error", details = ex.Message
});
        }
    }


    private async Task ProcessPaymentSuccess(JToken payload)
    {
        try
        {
            string procedureName = "InsertPhonepetDetails";

            var parameters = new Dictionary<string, object>
        {
            {"@PaymentId", payload["orderId"]?.ToString() ?? string.Empty },
            //{"@Amount", payload["amount"]?.ToString() ?? "0" },
            { "@Amount", (Convert.ToDecimal(payload["amount"]) / 100).ToString() },
                    {"@Status", payload["state"]?.ToString() == "COMPLETED" ? "Succeeded" :
"Failed" },
            {"@Currency", payload["metaInfo"]?["udf1"]?.ToString() ?? "INR" },
            {"@plan_name", payload["metaInfo"]?["udf2"]?.ToString() ?? "Default Plan" },
            {"@accountid", payload["metaInfo"]?["udf3"]?.ToString() ?? "Unknown" },
```

```
            {"@orderid", payload["paymentDetails"]?[0]?["transactionId"]?.ToString() ?? "" },
            {"@paymenttype", "Phonepe" },
            {"@ReceiptUrl", "" },
            {"@Email", "" },
            {"@productname", "" },
            {"@package_type", payload["metaInfo"]?["udf4"]?.ToString() ?? "Unknown" }
        };

            try
            {
                await Task.Run(() =>
                                        _dbHandler.ExecuteNonQuery(procedureName, parameters,
CommandType.StoredProcedure)
                );
                _logger.LogInformation("Payment details saved to DB successfully.");

                // Optional: write to fallback file (use C:\Logs safely)
                string fallbackLogPath = @"C:\Logs\webhook-log.txt";
                Directory.CreateDirectory(Path.GetDirectoryName(fallbackLogPath));
                await System.IO.File.AppendAllTextAsync(fallbackLogPath, $"{DateTime.Now} -
Inserted to DB: {parameters["@PaymentId"]}\n");
            }
            catch (Exception dbEx)
            {
                _logger.LogError(dbEx, "Database error while inserting PhonePe payment data.");
```

```
            // Fallback log to file
            string errorLogPath = @"C:\Logs\webhook-errors.txt";
            Directory.CreateDirectory(Path.GetDirectoryName(errorLogPath));
                await System.IO.File.AppendAllTextAsync(errorLogPath, $"{DateTime.Now} -
DB Error: {dbEx.Message}\n");

            throw; // Rethrow to trigger main handler
        }

        try
        {
            var generator = new PhonepeInvoiceGenerator();
            byte[] pdfBytes = generator.GenerateInvoice(payload);

            if (pdfBytes != null)
            {
                var insertInvoiceParams = new Dictionary<string, object>
                {
                    { "@PaymentId", payload["orderId"]?.ToString() ?? "" },
                    { "@PdfData", pdfBytes },
                    { "@CreatedAt", DateTime.UtcNow }
                };

                try
```

```
                {
                    await Task.Run(() =>
                                        _dbHandler.ExecuteNonQuery("InsertPhonePeInvoicePdf",
insertInvoiceParams, CommandType.StoredProcedure)
                    );
                        _logger.LogInformation("PDF invoice saved to database for Payment ID:
{0}", payload["orderId"]);
                }
                catch (Exception dbEx)
                {
                    _logger.LogError(dbEx, "Error saving PDF to DB.");
                    throw new ApplicationException("Failed to save PDF invoice to DB", dbEx);
                }
            }
            else
            {
                    _logger.LogWarning("PDF generation returned null for Payment ID: {0}",
payload["orderId"]);
            }
        }
        catch (Exception pdfEx)
        {
                    _logger.LogError(pdfEx, "Error generating PDF for Payment ID: {0}",
payload["orderId"]);
            throw new ApplicationException("Failed to generate PDF", pdfEx);
```

```
            }
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Fatal error in ProcessPaymentSuccess.");
            throw; // Rethrow to propagate to caller
        }
    }
}
```

8. ngrok Setup for Local Webhook Testing

8.1 ngrok Installation and Configuration

1. Download and install ngrok from https://ngrok.com/download
2. Sign up for a free ngrok account and get your auth token

Configure ngrok with your auth token:
ngrok config add-authtoken YOUR_AUTH_TOKEN

8.2 Start ngrok to Expose Local Webhook Endpoint

ngrok http 5000 --domain=your-reserved-domain.ngrok-free.app

Where 5000 is your local application port.

8.3 Configure Webhook URL in PhonePe

Update your payment initiation code to use the ngrok URL as the callback URL:

paymentFlow = new

```
{
    type = "PG_CHECKOUT",
    message = "Payment message used for collect requests",
    merchantUrls = new
    {
        callbackUrl = "https://your-reserved-domain.ngrok-free.app/api/webhook/phonepe",
        redirectUrl = request.RedirectUrl
    }
}
```

9. Environment Switching (UAT vs Production)

9.1 Configuration Setup

Create an appsettings.json file to manage different environments:

```
{
  "PhonePe": {
    "Environment": "UAT", // or "Production"
    "UAT": {
      "BaseUrl": "https://api-preprod.phonepe.com/apis/pg-sandbox",
      "MerchantId": "M223ZQK1QQMIZUAT_2503211",
      "SaltKey": "YzU0NDI5NDgtOWZiMi00NDI4LWJlODktOTIzOTExMDYwMGZi",
      "SaltIndex": "1",
      "TokenUrl": "https://api-preprod.phonepe.com/apis/pg-sandbox/v1/oauth/token"
    },
    "Production": {
```

```
    "BaseUrl": "https://api.phonepe.com/apis/hermes",
    "MerchantId": "[PRODUCTION_MERCHANT_ID]",
    "SaltKey": "[PRODUCTION_SALT_KEY]",
    "SaltIndex": "1",
    "TokenUrl": "https://api.phonepe.com/apis/hermes/v1/oauth/token"
  }
 }
}
```

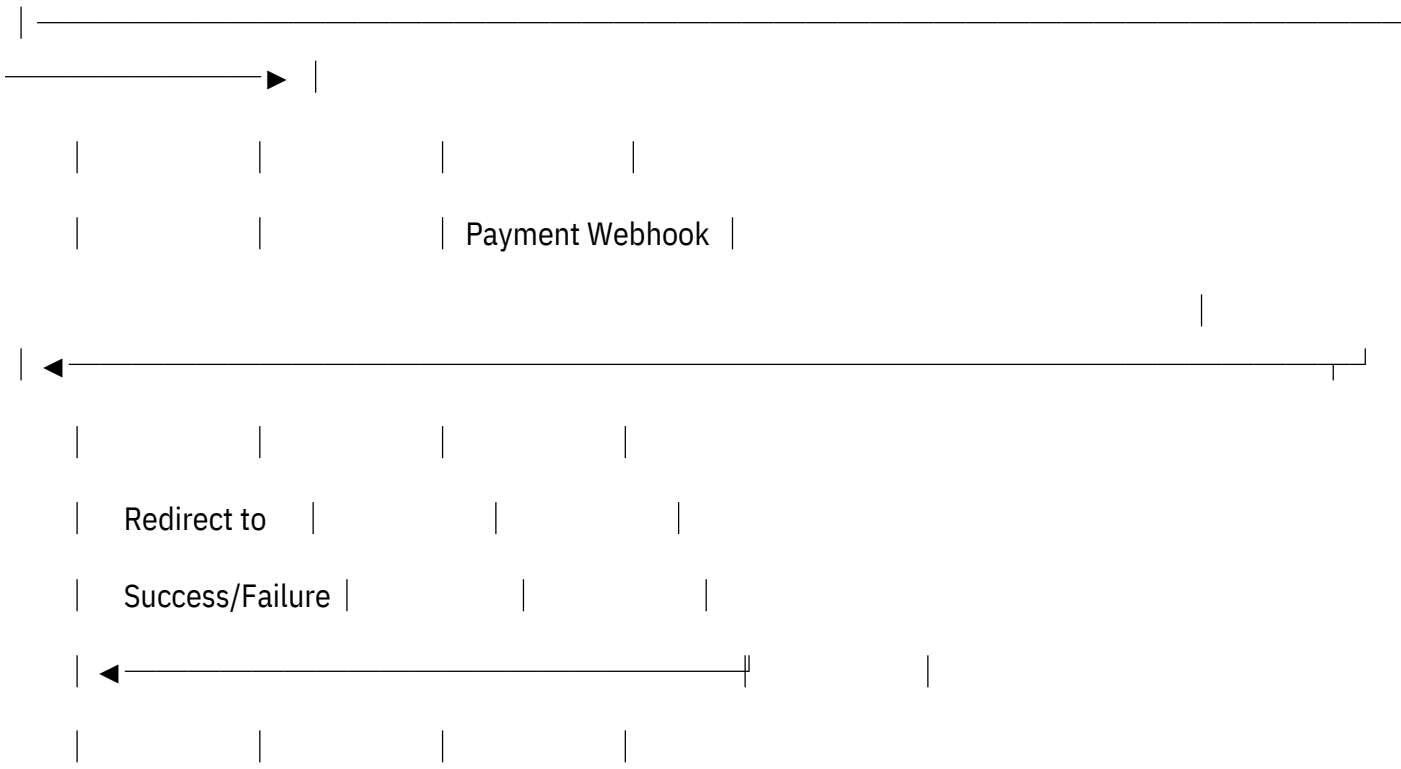
9.2 Configuration Service

```
public class PhonePeConfigService
{
    private readonly IConfiguration _configuration;

    public PhonePeConfigService(IConfiguration configuration)
    {
        _configuration = configuration;
    }

    public string GetBaseUrl()
    {
        string env = _configuration["PhonePe:Environment"];
        return _configuration[$"PhonePe:{env}:BaseUrl"];
    }
```

```
public string GetMerchantId()
{
    string env = _configuration["PhonePe:Environment"];
    return _configuration[$"PhonePe:{env}:MerchantId"];
}

public string GetSaltKey()
{
    string env = _configuration["PhonePe:Environment"];
    return _configuration[$"PhonePe:{env}:SaltKey"];
}

public string GetSaltIndex()
{
    string env = _configuration["PhonePe:Environment"];
    return _configuration[$"PhonePe:{env}:SaltIndex"];
}

public string GetTokenUrl()
{
    string env = _configuration["PhonePe:Environment"];
    return _configuration[$"PhonePe:{env}:TokenUrl"];
}
```

```
}
```

10. Transaction Flow Diagram



11. Testing and Troubleshooting

11.1 Sandbox Testing Credentials

For UAT testing, use the following test card:

- Card Number: 4111 1111 1111 1111
- Expiry: Any future date
- CVV: Any 3 digits
- Name: Any name
- OTP: 123456

11.2 Common Issues and Solutions

Issue	Possible Cause	Solution
Authentication failure	Invalid Merchant ID or Salt Key	Verify credentials in configuration
Checksum mismatch	Incorrect checksum generation	Ensure proper concatenation and encoding
Webhook not received	ngrok tunnel issue	Check ngrok status and logs
Invalid currency	Non-INR currency	Ensure currency conversion function works correctly
Payment failure	Amount too small	PhonePe has minimum amount restrictions (usually ₹1)

11.3 Logging Recommendations

Implement comprehensive logging:

- Log all API requests and responses
- Log webhook payloads
- Track payment status transitions
- Monitor ngrok connection status

12. Security Considerations

12.1 Secure Key Management

- Store Merchant ID and Salt Key in secure vault or environment variables
- Never expose these keys in client-side code
- Rotate keys periodically according to PhonePe policies

12.2 Webhook Security

- Always verify webhook signatures
- Use HTTPS for all communication
- Implement request timeout settings
- Validate incoming payload format

12.3 Transaction Data Protection

- Encrypt sensitive payment data in database
- Implement proper access controls
- Follow PCI-DSS guidelines for payment data

13. Production Deployment Checklist

13.1 Pre-Deployment Steps

- Complete merchant onboarding with PhonePe for production access
- Obtain production Merchant ID and Salt Key
- Update configuration to use production endpoints
- Replace ngrok with proper domain and SSL certificate
- Set up monitoring and alerting

13.2 Post-Deployment Verification

- Conduct end-to-end payment tests with minimal amounts
- Verify webhook processing
- Check transaction reports in PhonePe dashboard
- Monitor error logs

14. References

- PhonePe API Documentation
- ngrok Documentation
- .NET Core HttpClient Documentation
- ASP.NET Core WebHooks Documentation