PartnerHub

Stripe Integration Concept

Frontend & Backend Developer Reference

SEMYOU GmbH · ClubGefühle / PartnerHub

Tables: PartnerHub.Product · PartnerHub.ProviderOrder

Version 1.0 · March 2026

1. Architecture Overview

This document defines the complete data flow between the PartnerHub frontend, the .NET backend, Stripe, and the SQL Server database. It covers all four transactional scenarios and the required Stripe webhook handlers.

1.1 Core Principle: Stripe is the Source of Truth

The database mirrors Stripe state — it never drives it. Every ProviderOrder record is created or updated exclusively in response to Stripe webhook events or direct API calls. The frontend never writes billing state directly to the database.

Rule	Detail
Single Subscription ID	StripeSubscriptionId in ProviderOrder never changes on plan upgrades/downgrades. Only the Stripe price item inside the subscription is swapped.
FREE = €0 Subscription	The FREE plan is created as a €0 Stripe subscription at registration. This ensures all plan-change logic follows the same code path.
ValidTo is the Gate	Feature access is controlled by Status = Active AND (ValidTo IS NULL OR ValidTo > SYSUTCDATETIME()). Never rely on plan name alone.
Limits via Function	MaxActiveTrips and other limits are calculated by fn_GetProviderLimits() — never stored as fixed values. Automatically correct after upgrade/downgrade.
Webhooks update DB	ProviderOrder.Status, ValidTo, AmountPaid and CancelAtPeriodEnd are updated only via Stripe webhook events — never from frontend API calls.

1.2 Table Relationship Summary

Product → ProviderOrder (FK: ProductId). Each ProviderOrder belongs to exactly one TravelProvider and references one Product. For offer-level products (ContentUpgrade, Boost, AppPlacement targeted at offers), RefTravelOfferId is set.

ProductType	OrderType	RecurringType	Stripe Ref	ValidTo behavior
Plan (Free/Adv/Prem)	Provider	Monthly / Annual	`StripeSubscriptionId`	Extended on each invoice.paid
ExtraTrips	Provider	Monthly / Annual	`StripeSubscriptionId`	Extended on each invoice.paid
Boost (Reise/Profil)	Provider / Offer	Monthly	`StripeSubscriptionId`	Extended on each invoice.paid
AppPlacement	Provider / Offer	OneTime	`StripePaymentIntentId`	ValidFrom + 7 days (DurationDays)
Badge	Provider	OneTime	`StripePaymentIntentId`	NULL — permanent
ContentUpgrade	Offer	OneTime	`StripePaymentIntentId`	NULL — permanent

2. Scenario: Subscription Create

Covers both initial FREE plan registration and the first paid plan checkout (Advanced / Premium).

2.1 Registration — FREE Plan

Frontend responsibilities

- Call POST /api/auth/register with provider details.
- On success, show dashboard. No Stripe UI needed — the backend handles everything.
- No redirect, no payment method required.

Backend flow (synchronous, no webhook needed for FREE)

#	Actor	Action
1	Backend	stripe.customers.create() → stores StripeCustomerId on TravelProvider
2	Backend	stripe.subscriptions.create({ customer, items: [{ price: price_free_id }] }) — €0
3	Backend (sp)	sp_portal_Order_Create: INSERT ProviderOrder (ProductId=FREE, Status=Active, StripeSubscriptionId=sub_xxx, ValidFrom=NOW, ValidTo=NULL, AmountPaid=0)
4	Backend	Return 200 OK to frontend — provider is fully active

✓ Design decision
ValidTo = NULL for FREE plan — no expiry. Access gated by Status = Active only.

2.2 First Paid Plan Checkout (Advanced / Premium)

Frontend responsibilities

- Display plan selection with monthly/annual toggle.
- User clicks "Upgrade" → call POST /api/billing/create-checkout-session with { priceId, successUrl, cancelUrl }.
- Redirect to Stripe Checkout URL returned by backend.
- On return to successUrl: show "Processing…" spinner — do NOT assume active yet.
- Poll GET /api/billing/subscription-status every 2s until Status = Active (max ~15s), then update UI.

Backend + Webhook flow

#	Actor	Action
1	Backend	stripe.checkout.sessions.create({ mode: subscription, customer: existing_cus_xxx, line_items: [{ price: price_advanced_monthly }] })
2	Stripe	User completes payment in Stripe Checkout
3	Stripe → Webhook	invoice.paid fired: subscription_id, amount_paid, period_end
4	Backend (sp)	sp_portal_Order_UpdateByStripe: UPDATE existing FREE ProviderOrder → Status=Active, ProductId=Advanced, ValidTo=period_end, AmountPaid=amount
5	Frontend	Polling detects Status=Active → show success state

⚠ Critical
The FREE plan ProviderOrder row is updated in-place — its StripeSubscriptionId does NOT change. ProductId switches from FREE to Advanced/Premium. This is the single code path for all plan changes.

3. Scenario: Subscription Cancel

The provider cancels their paid plan. Access is not revoked immediately — they retain features until the end of the current billing period.

3.1 Frontend responsibilities

- Show "Cancel Plan" button in billing settings.
- Display confirmation dialog showing the exact date access ends (ValidTo from DB).
- Call POST /api/billing/cancel-subscription.
- On 200 response: update UI to show "Cancellation scheduled — access until {ValidTo}".
- Do NOT wait for webhook — the backend confirms synchronously based on Stripe API response.

3.2 Backend + Webhook flow

#	Actor	Action
1	Backend	stripe.subscriptions.update(sub_xxx, { cancel_at_period_end: true })
2	Stripe → Webhook	customer.subscription.updated: cancel_at_period_end = true
3	Backend (sp)	sp_portal_Order_UpdateByStripe: SET CancelAtPeriodEnd = 1 on matching ProviderOrder. Status stays Active. ValidTo stays unchanged.
4	Stripe → Webhook	At period end: customer.subscription.updated (status=canceled) OR customer.subscription.deleted
5	Backend (sp)	sp_portal_Order_UpdateByStripe: SET Status=Cancelled. ValidTo is NOT modified — provider retains access until ValidTo.
6	Stripe → Webhook	After ValidTo: nightly SQL job sets Status=Expired. OR: access check (ValidTo < NOW) naturally gates features.

⚠ Note
CancelAtPeriodEnd = 1 is purely informational for UI display. The actual access gate is ValidTo. Never use CancelAtPeriodEnd as a feature gate.

4. Scenario: Upgrade / Downgrade

The provider switches between plan tiers (e.g. Free → Advanced, Advanced → Premium, Premium → Advanced). The StripeSubscriptionId never changes — only the price item inside the subscription is swapped.

4.1 Frontend responsibilities

- Show current plan with upgrade/downgrade options.
- For upgrades (Free→Paid, Paid→higher): prorated charge may apply — show proration preview via GET /api/billing/proration-preview?priceId=xxx before confirming.
- Call POST /api/billing/change-plan with { newPriceId }.
- Show "Processing…" — update UI after polling confirms new ProductId is reflected.

- For downgrades to FREE: show "Downgrade scheduled — changes apply at period end" based on backend response.

ℹ Proration
Immediate upgrades (Stripe default) charge proration now. Downgrades should be scheduled at period end via cancel_at_period_end pattern or subscription schedule. Backend controls this — frontend only calls change-plan endpoint.

4.2 Backend + Webhook flow — Upgrade

#	Actor	Action
1	Backend	stripe.subscriptions.retrieve(sub_xxx) → get subscription item ID (si_xxx)
2	Backend	stripe.subscriptions.update(sub_xxx, { items: [{ id: si_xxx, price: new_price_id }], proration_behavior: create_prorations })
3	Stripe → Webhook	invoice.paid (proration invoice) fired immediately
4	Backend (sp)	sp_portal_Order_UpdateByStripe: UPDATE ProviderOrder — ProductId=new plan, ValidTo=new period_end, AmountPaid=proration amount
5	Stripe → Webhook	customer.subscription.updated also fires — used to confirm ProductId switch in DB if not already done

4.3 Backend + Webhook flow — Downgrade

#	Actor	Action
1	Backend	stripe.subscriptions.update(sub_xxx, { items: [{ id: si_xxx, price: lower_price_id }], proration_behavior: none, billing_cycle_anchor: unchanged })
2	Stripe → Webhook	customer.subscription.updated: new price reflected
3	Backend (sp)	sp_portal_Order_UpdateByStripe: ProductId = new lower plan. ValidTo unchanged. CancelAtPeriodEnd unchanged.
4	Backend	Call fn_GetProviderLimits() — if active TravelOffers exceed new plan limit: SET GracePeriodEnd = NOW + 30 days on TravelProvider
5	Frontend	Show warning: "Your current offers exceed the new plan limit. You have 30 days to resolve this."

⚠ Downgrade Grace Period
On downgrade: existing TravelOffer records are NOT deactivated. IsOverLimit = 1 is set on offers exceeding the new limit. Providers retain access to data — only new offer creation is blocked until within limits.

5. Scenario: One-Time Order (Badge, ContentUpgrade, AppPlacement)

One-time products use Stripe Payment Intents (not subscriptions). StripePaymentIntentId is stored instead of StripeSubscriptionId. These are mutually exclusive per ProviderOrder row (enforced by CHK_ProviderOrder_StripeRef).

5.1 Frontend responsibilities

- Show add-on shop listing (ProductType: Badge, ContentUpgrade, AppPlacement).
- Call POST /api/billing/create-payment-intent with { productId, refTravelOfferId? }.
- Render Stripe Payment Element using the clientSecret returned.
- On stripe.confirmPayment() success: show "Processing…" and poll order status.
- For AppPlacement (Deal der Woche): show slot availability — only 1 slot per week; display "Sold out" if taken.

5.2 Backend + Webhook flow

#	Actor	Action
1	Backend	stripe.paymentIntents.create({ amount, currency, customer: cus_xxx, metadata: { productId, refTravelOfferId } })
2	Backend	Return { clientSecret } to frontend — no DB write yet
3	Frontend	User enters payment details and confirms via Stripe.js Payment Element
4	Stripe → Webhook	payment_intent.succeeded: payment_intent.id, amount, metadata
5	Backend (sp)	sp_portal_Order_Create: INSERT ProviderOrder (StripePaymentIntentId=pi_xxx, Status=Active, ValidFrom=NOW, ValidTo=NOW+DurationDays OR NULL, AmountPaid)
6	Frontend	Polling detects new order → show activated add-on in UI

5.3 ValidTo Calculation per Product

Product	DurationDays	ValidTo	Notes
AppPlacement (Deal der Woche)	7	ValidFrom + 7 days	Weekly slot — max 1 concurrent
Boost (Reise/Profil)	30	ValidFrom + 30 days	If sold as OneTime (not subscription)
Badge (Verifiziert)	NULL	NULL (permanent)	Stays active indefinitely
ContentUpgrade	NULL	NULL (permanent)	Per-offer; stays active unless refunded

6. Required Stripe Webhooks

All webhooks must be verified using the Stripe-Signature header and the webhook signing secret (whsec_xxx). Register all events in the Stripe Dashboard under Developers → Webhooks.

6.1 Webhook Event Registry

Event	Priority	Purpose
Subscription Events		
`customer.subscription.created`	**HIGH**	New subscription created. Store StripeSubscriptionId, set Status=Active for Free plan.
`customer.subscription.updated`	**HIGH**	Plan change, cancel_at_period_end flag, or status change. Read items[0].price to detect plan switch.

`customer.subscription.deleted`	**HIGH**	Subscription fully cancelled (after period end). Set Status=Cancelled.
Invoice Events		
`invoice.paid`	**HIGH**	Payment confirmed. Update ValidTo = subscription.current_period_end, AmountPaid = invoice.amount_paid.
`invoice.payment_failed`	**HIGH**	Payment attempt failed. Set Status=PastDue. Stripe will retry automatically. Trigger email notification.
`invoice.payment_action_required`	**MEDIUM**	3D Secure / Strong Customer Authentication required. Send email with payment link to provider.
`invoice.upcoming`	LOW	Optional: send renewal reminder email 7 days before next billing. No DB write.
Payment Intent Events (One-Time Products)		
`payment_intent.succeeded`	**HIGH**	One-time payment confirmed. INSERT new ProviderOrder with StripePaymentIntentId, compute ValidTo.
`payment_intent.payment_failed`	**MEDIUM**	One-time payment failed. No DB write needed (no order created yet). Optionally log for support.
Customer Events		
`customer.updated`	LOW	Billing address or VAT ID changed in Stripe. Optionally sync to PartnerHub.BillingAddress table.
Charge / Refund Events		
`charge.refunded`	**MEDIUM**	Refund issued. For one-time products: set Status=Cancelled on matching ProviderOrder. For subscriptions: handled via subscription events.

7. Webhook Payload — Key Fields

For each relevant webhook, the following fields must be extracted and used to update the database.

7.1 invoice.paid

Field Path	Maps to DB
`invoice.subscription`	Look up ProviderOrder by StripeSubscriptionId
`invoice.lines.data[0].period.end`	ProviderOrder.ValidTo (as UTC datetime)
`invoice.amount_paid / 100`	ProviderOrder.AmountPaid
`invoice.currency`	ProviderOrder.Currency

7.2 customer.subscription.updated

Field Path	Maps to DB
`subscription.id`	Look up ProviderOrder by StripeSubscriptionId
`subscription.items.data[0].price.id`	Lookup in Product.StripePriceId → get new ProductId
`subscription.cancel_at_period_end`	ProviderOrder.CancelAtPeriodEnd

`subscription.status`	Map to ProviderOrder.Status (active→Active, past_due→PastDue, canceled→Cancelled)
`subscription.current_period_end`	ProviderOrder.ValidTo — only update here if invoice.paid did not already update it

7.3 payment_intent.succeeded (One-Time)

Field Path	Maps to DB
`payment_intent.id`	ProviderOrder.StripePaymentIntentId
`payment_intent.metadata.productId`	ProviderOrder.ProductId
`payment_intent.metadata.refTravelOfferId`	ProviderOrder.RefTravelOfferId (nullable)
`payment_intent.amount / 100`	ProviderOrder.AmountPaid
`payment_intent.customer`	Look up TravelProviderId via TravelProvider.StripeCustomerId

⚠ Required
Always pass productId and refTravelOfferId in PaymentIntent metadata at creation time. These are needed to create the ProviderOrder on webhook receipt — the webhook handler cannot reconstruct them otherwise.

8. Database Write Rules by Event

Quick reference: which DB fields change on which event.

Stripe Event	Status	ValidTo	ProductId	Other fields
`customer.subscription.created`	**Active**	NULL (Free)	FREE	StripeSubscriptionId set
`invoice.paid`	**Active**	→ period_end	(unchanged)	AmountPaid updated
`invoice.payment_failed`	**PastDue**	unchanged	unchanged	Trigger email
`customer.subscription.updated (plan change)`	**Active**	unchanged	→ new plan	CancelAtPeriodEnd may change
`customer.subscription.updated (cancel flag)`	**Active**	unchanged	unchanged	CancelAtPeriodEnd = 1
`customer.subscription.deleted`	**Cancelled**	unchanged	unchanged	Access until ValidTo
`payment_intent.succeeded`	**Active**	NOW+DurationDays / NULL	from metadata	INSERT new row; StripePaymentIntentId set
`charge.refunded`	**Cancelled**	NOW (if OneTime)	unchanged	One-time orders only

9. Frontend Status Mapping

How to translate ProviderOrder.Status + ValidTo + CancelAtPeriodEnd into user-facing UI states.

Status	ValidTo	CancelAtPeriodEnd	UI Message
Active	NULL	0	Free plan active — no expiry
Active	>= today	0	"Advanced plan — renews {ValidTo}"
Active	>= today	1	"Plan cancelled — access until {ValidTo}"
PastDue	>= today	0	"Payment failed — please update payment method. Access maintained during retry."
Cancelled	>= today	any	"Plan ended — access until {ValidTo}. Upgrade to restore."
Cancelled	< today	any	"No active plan — features restricted to Free."
Expired	< today	any	"Add-on expired on {ValidTo}."
Incomplete	any	any	"Payment not confirmed — please complete checkout."

ℹ Access Gate Query
Access check query: SELECT TOP 1 Status, ValidTo, CancelAtPeriodEnd FROM PartnerHub.ProviderOrder WHERE TravelProviderId=@id AND ProductType='Plan' AND Status IN ('Active','PastDue') AND (ValidTo IS NULL OR ValidTo > SYSUTCDATETIME()). If no row: treat as FREE restricted.

10. Idempotency & Error Handling

10.1 Webhook Idempotency

- Stripe may deliver the same event more than once. Always design webhook handlers to be idempotent.
- Check: if UPDATE affected 0 rows (StripeSubscriptionId not found), log and return 200 — do not error.
- For INSERT (payment_intent.succeeded): use StripePaymentIntentId as unique key. If already exists, skip and return 200.
- Always return HTTP 200 to Stripe — even on internal errors — otherwise Stripe will retry and may cause duplicate processing.

10.2 Webhook Security

- Verify every webhook using EventUtility.ConstructEvent(json, Stripe-Signature header, whsec_xxx).
- Never trust event data without signature verification.
- Use a separate webhook endpoint secret (whsec_xxx) distinct from the Stripe secret key.
- Webhook endpoint should be AllowAnonymous but signature-verified — never require JWT auth.

10.3 Stripe API Key Usage

Key	Where used
`sk_live_xxx (Secret Key)`	Backend only — NEVER expose to frontend
`pk_live_xxx (Publishable Key)`	Frontend only — used to initialize Stripe.js / Payment Element

`whsec_xxx (Webhook Secret)`	Backend webhook handler only — verifies Stripe-Signature header
`price_xxx (Price IDs)`	Stored in Product.StripePriceId — frontend sends priceId to backend, never constructs Stripe calls directly

11. Stored Procedures Reference

All procedures live in the PartnerHub schema. Naming convention: sp_portal_Order_* (CamelCase). Every write procedure is idempotent — re-running with the same Stripe ID is safe.

sp_portal_Order_CreateFree

Creates the €0 FREE plan ProviderOrder synchronously during provider registration. Called once per provider — Stripe subscription already created via API before this SP is called.

When called: POST /api/auth/register → backend creates Stripe Customer + Subscription → calls this SP.

Parameter	Type	Req.	Description
`@TravelProviderId`	BIGINT	✓	ID of the newly registered TravelProvider.
`@StripeSubscriptionId`	NVARCHAR(100)	✓	sub_xxx returned by stripe.subscriptions.create() for the €0 FREE plan.
`@Result OUTPUT`	INT	—	0=success, 1=provider not found, 2=FREE product not found in Product table, 3=subscription already exists (idempotent).

Returns: OrderId, TravelProviderId, ProductId, Status, ValidFrom, ValidTo (NULL), AmountPaid (0.00), StripeSubscriptionId.

sp_portal_Order_Create

Inserts a new ProviderOrder for a ONE-TIME product. Called exclusively from the payment_intent.succeeded webhook handler. ValidTo = ValidFrom + Product.DurationDays, or NULL for permanent products (Badge, ContentUpgrade).

When called: POST /api/webhooks/stripe → payment_intent.succeeded → this SP.

Parameter	Type	Req.	Description
`@TravelProviderId`	BIGINT	✓	Resolved from TravelProvider.StripeCustomerId matching payment_intent.customer.
`@ProductId`	BIGINT	✓	Resolved from Product.Id matching payment_intent.metadata.productId.
`@RefTravelOfferId`	BIGINT	—	NULL for Provider-scope products. Set from payment_intent.metadata.refTravelOfferId for Offer-scope products.
`@StripePaymentIntentId`	NVARCHAR(100)	✓	pi_xxx from payment_intent.id. Used as idempotency key — duplicate calls are silently skipped.
`@AmountPaid`	DECIMAL(10,2)	✓	payment_intent.amount / 100. Stored for audit.

@Currency	CHAR(3)	—	Default EUR. From payment_intent.currency.
@Result OUTPUT	INT	—	0=success, 1=provider not found, 2=product not found/inactive, 3=duplicate pi (idempotent, returns existing row).

⚠ Required
Always set productId and refTravelOfferId in PaymentIntent metadata at creation time — these cannot be recovered at webhook time otherwise.

sp_portal_Order_UpdateBySubscription

Master webhook handler for all subscription lifecycle events. A single SP routing on @EventType keeps webhook handling simple: the .NET controller extracts the right fields per event type and passes them through.

When called: POST /api/webhooks/stripe → any subscription or invoice event → this SP.

Parameter	Type	Req.	Description
@StripeSubscriptionId	NVARCHAR(100)	✓	sub_xxx. Primary lookup key — finds the ProviderOrder to update.
@EventType	NVARCHAR(80)	✓	Stripe event type string. Accepted values: invoice.paid · invoice.payment_failed · customer.subscription.updated · customer.subscription.deleted
@NewValidTo	DATETIME2(7)	—	[invoice.paid only] subscription.current_period_end converted to UTC. Extends ValidTo on every successful renewal.
@AmountPaid	DECIMAL(10,2)	—	[invoice.paid only] invoice.amount_paid / 100. Updates AmountPaid for latest billing cycle.
@NewStripePriceId	NVARCHAR(100)	—	[subscription.updated only] subscription.items.data[0].price.id. Resolved to ProductId via Product.StripePriceId lookup.
@NewCancelAtPeriodEnd	BIT	—	[subscription.updated only] subscription.cancel_at_period_end. Mirrors the Stripe flag for UI display.
@Result OUTPUT	INT	—	0=success or idempotent (no row matched), 1=StripePriceId not found in Product table (product not yet seeded).

EventType value	DB fields written
invoice.paid	Status=Active, ValidTo=@NewValidTo, AmountPaid=@AmountPaid, UpdatedAt
invoice.payment_failed	Status=PastDue, UpdatedAt (only if not already Cancelled/Expired)
customer.subscription.updated	ProductId (resolved from price), CancelAtPeriodEnd, UpdatedAt
customer.subscription.deleted	Status=Cancelled, UpdatedAt — ValidTo NOT modified

sp_portal_Order_UpdateByPaymentIntent

Handles post-creation lifecycle events for one-time orders. Currently handles charge.refunded — sets Status=Cancelled and ValidTo=NOW to immediately revoke access.

When called: POST /api/webhooks/stripe → charge.refunded → this SP.

Parameter	Type	Req.	Description
`@StripePaymentIntentId`	NVARCHAR(100)	✓	pi_xxx. Primary lookup key for one-time ProviderOrder rows.
`@EventType`	NVARCHAR(80)	✓	Currently: charge.refunded. Future events can be routed here.
`@Result OUTPUT`	INT	—	0=success, 1=order not found or already not Active (idempotent).

sp_portal_Order_GetActiveByProvider

Returns all currently active and PastDue orders for a provider, joined with Product metadata. Used by the frontend billing overview page to render active add-ons, expiry dates and cancellation flags.

When called: GET /api/billing/orders → returns full active order list.

Parameter	Type	Req.	Description
`@TravelProviderId`	BIGINT	✓	Provider whose orders should be loaded.

Returns: OrderId, ProductCode, ProductType, ProductTitle, TargetEntity, RecurringType, RefTravelOfferId, OrderType, Status, ValidFrom, ValidTo, CancelAtPeriodEnd, AmountPaid, Currency, StripeSubscriptionId, StripePaymentIntentId, UpdatedAt — ordered by Product.SortOrder.

sp_portal_Order_GetPlanStatus

Lightweight read: returns the single active Plan order for a provider. Called on every authenticated page load to determine feature gates without a Stripe API call. Returns 0 or 1 row.

When called: middleware / auth token refresh → GET /api/billing/plan-status.

Parameter	Type	Req.	Description
`@TravelProviderId`	BIGINT	✓	Provider to check. Returns 0 rows if no active plan — frontend treats this as FREE restricted.

Returns: OrderId, Status, ValidFrom, ValidTo, CancelAtPeriodEnd, AmountPaid, ProductCode, PlanTitle, PlanPrice, RecurringType.

sp_portal_Order_SetCancelFlag

Sets CancelAtPeriodEnd synchronously from the cancel API call — before the Stripe webhook arrives. This allows the frontend to immediately show the cancellation state without waiting for the async webhook. The following customer.subscription.updated webhook will call UpdateBySubscription with the same value — fully idempotent.

When called: POST /api/billing/cancel-subscription (and optionally /api/billing/undo-cancel) → this SP → then returns GetPlanStatus result.

Parameter	Type	Req.	Description
`@TravelProviderId`	BIGINT	✓	Scopes the update to the correct provider — prevents cross-tenant writes.
`@StripeSubscriptionId`	NVARCHAR(100)	✓	sub_xxx. Secondary key ensuring the correct order row is updated.
`@CancelAtPeriodEnd`	BIT	✓	1 = schedule cancellation at period end. 0 = undo scheduled cancellation.
`@Result OUTPUT`	INT	—	0=success, 1=no Active order found for this provider+subscription.

Returns: result of sp_portal_Order_GetPlanStatus for immediate frontend state refresh.

12. SQL Script — All Stored Procedures

Complete T-SQL script for all 7 stored procedures. Execute in SQL Server Management Studio against the target database containing the PartnerHub schema. All procedures use CREATE OR ALTER — safe to re-run.

✓ Execution
Execute order does not matter — no cross-SP dependencies. All 7 can be run in a single batch. Requires PartnerHub.Product and PartnerHub.ProviderOrder tables to exist before execution.

```
-- ============================================================
-- Schema   : PartnerHub
-- Purpose  : Stripe integration — Order lifecycle management
-- Naming   : sp_portal_Order_* (CamelCase after prefix)
-- Version  : 1.0 · March 2026
-- ============================================================
-- Procedures in this file:
--   1. sp_portal_Order_CreateFree          — Registration: INSERT €0 plan order
--   2. sp_portal_Order_Create              — One-time product: INSERT on
payment_intent.succeeded
--   3. sp_portal_Order_UpdateBySubscription — Webhook handler for subscription events
--   4. sp_portal_Order_UpdateByPaymentIntent — Webhook handler for payment_intent events
--   5. sp_portal_Order_GetActiveByProvider  — Load all active orders for a provider
--   6. sp_portal_Order_GetPlanStatus        — Frontend: current plan + status for a provider
--   7. sp_portal_Order_SetCancelFlag        — Set/unset CancelAtPeriodEnd
-- ============================================================



-- ============================================================
-- 1. sp_portal_Order_CreateFree
--    Called during provider registration.
--    Creates the €0 FREE plan ProviderOrder synchronously
--    (no webhook needed for free plan creation).
--    Requires StripeSubscriptionId already created via Stripe API.
-- ============================================================
CREATE OR ALTER PROCEDURE [PartnerHub].[sp_portal_Order_CreateFree]
    @TravelProviderId       BIGINT,
    @StripeSubscriptionId   NVARCHAR(100),    -- sub_xxx from stripe.subscriptions.create
    @Result                 INT OUTPUT        -- 0=success, 1=provider not found, 2=free
product not found, 3=order already exists
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate provider
    IF NOT EXISTS (
        SELECT 1 FROM [PartnerHub].[TravelProvider]
        WHERE Id = @TravelProviderId AND IsDeleted = 0
    )
    BEGIN
        SET @Result = 1;
        RETURN;
    END

    -- Prevent duplicate
    IF EXISTS (
        SELECT 1 FROM [PartnerHub].[ProviderOrder]
        WHERE StripeSubscriptionId = @StripeSubscriptionId
    )
    BEGIN
        SET @Result = 3;
        RETURN;
    END

    -- Resolve FREE plan ProductId
    DECLARE @ProductId BIGINT;
    SELECT TOP 1 @ProductId = Id
    FROM [PartnerHub].[Product]
    WHERE ProductType = 'Plan'
      AND Price = 0.00
      AND IsActive = 1;

    IF @ProductId IS NULL
    BEGIN
        SET @Result = 2;
        RETURN;
    END

    -- Insert FREE plan order
    INSERT INTO [PartnerHub].[ProviderOrder]
    (
        TravelProviderId,
        ProductId,
        RefTravelOfferId,
        OrderType,
        Status,
        OrderDate,
        ValidFrom,
        ValidTo,
        CancelAtPeriodEnd,
        AmountPaid,
        Currency,
        StripeSubscriptionId,
        StripePaymentIntentId,
        CreatedAt,
        UpdatedAt
    )
    VALUES
    (
        @TravelProviderId,
        @ProductId,
        NULL,                            -- Provider-level order
        'Provider',
        'Active',
        SYSUTCDATETIME(),
        SYSUTCDATETIME(),
        NULL,                            -- FREE plan never expires
        0,
        0.00,
        'EUR',
        @StripeSubscriptionId,
        NULL,
        SYSUTCDATETIME(),
        SYSUTCDATETIME()
    );

    SET @Result = 0;
    -- Return created order
    SELECT
        OrderId,
        TravelProviderId,
        ProductId,
        Status,
        ValidFrom,
        ValidTo,
        AmountPaid,
        StripeSubscriptionId
    FROM [PartnerHub].[ProviderOrder]
    WHERE StripeSubscriptionId = @StripeSubscriptionId;
END;
GO



-- ============================================================
-- 2. sp_portal_Order_Create
--    Called on payment_intent.succeeded webhook.
--    Creates a new ProviderOrder for a ONE-TIME product
--    (Badge, ContentUpgrade, AppPlacement, OneTime Boost).
--    ValidTo = ValidFrom + Product.DurationDays (NULL if DurationDays IS NULL).
-- ============================================================
CREATE OR ALTER PROCEDURE [PartnerHub].[sp_portal_Order_Create]
    @TravelProviderId           BIGINT,
    @ProductId                  BIGINT,
    @RefTravelOfferId           BIGINT          = NULL,     -- required when
Product.TargetEntity = 'TravelOffer'
    @StripePaymentIntentId      NVARCHAR(100),              -- pi_xxx
    @AmountPaid                 DECIMAL(10, 2),             -- invoice.amount_paid / 100
    @Currency                   CHAR(3)         = 'EUR',
    @Result                     INT             OUTPUT       -- 0=success, 1=provider not
found, 2=product not found, 3=duplicate pi
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate provider
    IF NOT EXISTS (
        SELECT 1 FROM [PartnerHub].[TravelProvider]
        WHERE Id = @TravelProviderId AND IsDeleted = 0
    )
    BEGIN
        SET @Result = 1;
        RETURN;
    END

    -- Validate product
    DECLARE @DurationDays INT;
    DECLARE @TargetEntity NVARCHAR(20);
    DECLARE @OrderType    NVARCHAR(20);

    SELECT
        @DurationDays = DurationDays,
        @TargetEntity = TargetEntity
    FROM [PartnerHub].[Product]
    WHERE Id = @ProductId AND IsActive = 1;

    IF @TargetEntity IS NULL
    BEGIN
        SET @Result = 2;
        RETURN;
    END

    -- Idempotency: skip if PaymentIntent already processed
    IF EXISTS (
        SELECT 1 FROM [PartnerHub].[ProviderOrder]
        WHERE StripePaymentIntentId = @StripePaymentIntentId
    )
    BEGIN
        SET @Result = 3;
        -- Return existing row — idempotent
```

```sql
        SELECT OrderId, Status, ValidFrom, ValidTo, AmountPaid
        FROM [PartnerHub].[ProviderOrder]
        WHERE StripePaymentIntentId = @StripePaymentIntentId;
        RETURN;
    END

    -- Determine OrderType from TargetEntity
    SET @OrderType = CASE WHEN @TargetEntity = 'TravelOffer' THEN 'Offer' ELSE 'Provider' END;

    DECLARE @ValidFrom DATETIME2(7) = SYSUTCDATETIME();
    DECLARE @ValidTo   DATETIME2(7) = CASE
        WHEN @DurationDays IS NOT NULL
        THEN DATEADD(DAY, @DurationDays, @ValidFrom)
        ELSE NULL   -- permanent (Badge, ContentUpgrade)
    END;

    INSERT INTO [PartnerHub].[ProviderOrder]
    (
        TravelProviderId,
        ProductId,
        RefTravelOfferId,
        OrderType,
        Status,
        OrderDate,
        ValidFrom,
        ValidTo,
        CancelAtPeriodEnd,
        AmountPaid,
        Currency,
        StripeSubscriptionId,
        StripePaymentIntentId,
        CreatedAt,
        UpdatedAt
    )
    VALUES
    (
        @TravelProviderId,
        @ProductId,
        @RefTravelOfferId,
        @OrderType,
        'Active',
        SYSUTCDATETIME(),
        @ValidFrom,
        @ValidTo,
        0,
        @AmountPaid,
        @Currency,
        NULL,                           -- OneTime: no subscription
        @StripePaymentIntentId,
        SYSUTCDATETIME(),
        SYSUTCDATETIME()
    );

    SET @Result = 0;

    SELECT
        OrderId,
        TravelProviderId,
        ProductId,
        RefTravelOfferId,
        OrderType,
        Status,
        ValidFrom,
        ValidTo,
        AmountPaid,
        StripePaymentIntentId
    FROM [PartnerHub].[ProviderOrder]
    WHERE StripePaymentIntentId = @StripePaymentIntentId;
END;
GO
```

```
-- ============================================================
-- 3. sp_portal_Order_UpdateBySubscription
--    Master webhook handler for all subscription-related events.
--    Called by the .NET webhook controller for:
--      - invoice.paid                     → extend ValidTo, AmountPaid, Status=Active
--      - invoice.payment_failed           → Status=PastDue
--      - customer.subscription.updated    → ProductId change, CancelAtPeriodEnd, status sync
--      - customer.subscription.deleted    → Status=Cancelled
--    Idempotent: returns 0 even if no row matched (Stripe may send duplicates).
-- ============================================================
CREATE OR ALTER PROCEDURE [PartnerHub].[sp_portal_Order_UpdateBySubscription]
    @StripeSubscriptionId   NVARCHAR(100),

    -- Event routing — caller passes the Stripe event type string
    @EventType              NVARCHAR(80),
    --   invoice.paid
    --   invoice.payment_failed
    --   customer.subscription.updated
    --   customer.subscription.deleted

    -- invoice.paid
    @NewValidTo             DATETIME2(7)    = NULL,     -- subscription.current_period_end (as
UTC)
    @AmountPaid             DECIMAL(10, 2)  = NULL,     -- invoice.amount_paid / 100

    -- customer.subscription.updated — plan change
    @NewStripePriceId       NVARCHAR(100)   = NULL,     -- subscription.items.data[0].price.id
    @NewCancelAtPeriodEnd   BIT             = NULL,     -- subscription.cancel_at_period_end

    @Result                 INT             OUTPUT      -- 0=success/idempotent, 1=price not
found in Product table
AS
BEGIN
    SET NOCOUNT ON;

    SET @Result = 0;

    -- ── invoice.paid ──────────────────────────────────────────────────────────
    IF @EventType = 'invoice.paid'
    BEGIN
        UPDATE [PartnerHub].[ProviderOrder]
        SET
            Status      = 'Active',
            ValidTo     = ISNULL(@NewValidTo, ValidTo),
            AmountPaid  = ISNULL(@AmountPaid, AmountPaid),
            UpdatedAt   = SYSUTCDATETIME()
        WHERE StripeSubscriptionId = @StripeSubscriptionId;
        -- Idempotent: 0 rows updated is OK
        RETURN;
    END

    -- ── invoice.payment_failed ────────────────────────────────────────────────
    IF @EventType = 'invoice.payment_failed'
    BEGIN
        UPDATE [PartnerHub].[ProviderOrder]
        SET
            Status      = 'PastDue',
            UpdatedAt   = SYSUTCDATETIME()
        WHERE StripeSubscriptionId = @StripeSubscriptionId
          AND Status NOT IN ('Cancelled', 'Expired');
        RETURN;
    END

    -- ── customer.subscription.updated ─────────────────────────────────────────
    IF @EventType = 'customer.subscription.updated'
    BEGIN
        -- Resolve new ProductId from StripePriceId (if plan changed)
        DECLARE @NewProductId BIGINT = NULL;

        IF @NewStripePriceId IS NOT NULL
        BEGIN
```

```
            SELECT @NewProductId = Id
            FROM [PartnerHub].[Product]
            WHERE StripePriceId = @NewStripePriceId
              AND IsActive = 1;

            IF @NewProductId IS NULL
            BEGIN
                -- Price not found — product may not be seeded yet; signal caller
                SET @Result = 1;
                RETURN;
            END
        END

        UPDATE [PartnerHub].[ProviderOrder]
        SET
            ProductId           = ISNULL(@NewProductId, ProductId),
            CancelAtPeriodEnd   = ISNULL(@NewCancelAtPeriodEnd, CancelAtPeriodEnd),
            UpdatedAt           = SYSUTCDATETIME()
        WHERE StripeSubscriptionId = @StripeSubscriptionId;
        RETURN;
    END

    -- ── customer.subscription.deleted ─────────────────────────────────────────
    IF @EventType = 'customer.subscription.deleted'
    BEGIN
        UPDATE [PartnerHub].[ProviderOrder]
        SET
            Status      = 'Cancelled',
            UpdatedAt   = SYSUTCDATETIME()
        WHERE StripeSubscriptionId = @StripeSubscriptionId
          AND Status NOT IN ('Cancelled', 'Expired');
        -- ValidTo is intentionally NOT modified: provider retains access until period end
        RETURN;
    END

END;
GO



-- ============================================================
-- 4. sp_portal_Order_UpdateByPaymentIntent
--    Called on charge.refunded webhook.
--    Sets a one-time order to Cancelled and optionally clears ValidTo.
--    payment_intent.succeeded creates a NEW row via sp_portal_Order_Create —
--    this procedure only handles post-creation lifecycle events.
-- ============================================================
CREATE OR ALTER PROCEDURE [PartnerHub].[sp_portal_Order_UpdateByPaymentIntent]
    @StripePaymentIntentId  NVARCHAR(100),
    @EventType              NVARCHAR(80),
    -- charge.refunded

    @Result                 INT OUTPUT      -- 0=success/idempotent, 1=order not found
AS
BEGIN
    SET NOCOUNT ON;

    IF @EventType = 'charge.refunded'
    BEGIN
        UPDATE [PartnerHub].[ProviderOrder]
        SET
            Status      = 'Cancelled',
            ValidTo     = SYSUTCDATETIME(),     -- revoke access immediately on refund
            UpdatedAt   = SYSUTCDATETIME()
        WHERE StripePaymentIntentId = @StripePaymentIntentId
          AND Status = 'Active';

        SET @Result = CASE WHEN @@ROWCOUNT > 0 THEN 0 ELSE 1 END;
        RETURN;
    END

    SET @Result = 0;
```

```sql
END;
GO


-- ============================================================
-- 5. sp_portal_Order_GetActiveByProvider
--    Returns all currently active orders for a provider.
--    Used by frontend to render active add-ons, plan badge,
--    expiry dates and cancellation flags.
--    Joins Product to include display metadata.
-- ============================================================
CREATE OR ALTER PROCEDURE [PartnerHub].[sp_portal_Order_GetActiveByProvider]
    @TravelProviderId   BIGINT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT
        o.OrderId,
        o.ProductId,
        p.ProductCode,
        p.ProductType,
        p.TargetEntity,
        p.Title             AS ProductTitle,
        p.Headline          AS ProductHeadline,
        p.Price             AS ProductPrice,
        p.RecurringType,
        o.RefTravelOfferId,
        o.OrderType,
        o.Status,
        o.ValidFrom,
        o.ValidTo,
        o.CancelAtPeriodEnd,
        o.AmountPaid,
        o.Currency,
        o.StripeSubscriptionId,
        o.StripePaymentIntentId,
        o.OrderDate,
        o.UpdatedAt
    FROM [PartnerHub].[ProviderOrder] o
    JOIN [PartnerHub].[Product]        p ON p.Id = o.ProductId
    WHERE o.TravelProviderId = @TravelProviderId
      AND o.Status IN ('Active', 'PastDue')
      AND (o.ValidTo IS NULL OR o.ValidTo > SYSUTCDATETIME())
    ORDER BY p.SortOrder, o.ValidFrom DESC;
END;
GO


-- ============================================================
-- 6. sp_portal_Order_GetPlanStatus
--    Lightweight call: returns the current plan order for a provider.
--    Used by frontend on every page load / auth token refresh
--    to determine feature gates without calling Stripe API.
--    Returns exactly 0 or 1 row.
-- ============================================================
CREATE OR ALTER PROCEDURE [PartnerHub].[sp_portal_Order_GetPlanStatus]
    @TravelProviderId   BIGINT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT TOP 1
        o.OrderId,
        o.Status,
        o.ValidFrom,
        o.ValidTo,
        o.CancelAtPeriodEnd,
        o.AmountPaid,
        p.ProductCode,
        p.Title             AS PlanTitle,
        p.Price             AS PlanPrice,
        p.RecurringType
    FROM [PartnerHub].[ProviderOrder] o
    JOIN [PartnerHub].[Product]        p ON p.Id = o.ProductId
    WHERE o.TravelProviderId = @TravelProviderId
      AND p.ProductType      = 'Plan'
      AND o.Status IN ('Active', 'PastDue')
      AND (o.ValidTo IS NULL OR o.ValidTo > SYSUTCDATETIME())
    ORDER BY o.ValidFrom DESC;

    -- If no row returned: provider has no active plan → treat as FREE restricted
END;
GO



-- ============================================================
-- 7. sp_portal_Order_SetCancelFlag
--    Called synchronously from POST /api/billing/cancel-subscription.
--    Sets CancelAtPeriodEnd = 1 BEFORE the webhook arrives
--    so the frontend can reflect the cancellation immediately.
--    The webhook customer.subscription.updated will also fire
--    and call sp_portal_Order_UpdateBySubscription — idempotent.
-- ============================================================
CREATE OR ALTER PROCEDURE [PartnerHub].[sp_portal_Order_SetCancelFlag]
    @TravelProviderId       BIGINT,
    @StripeSubscriptionId   NVARCHAR(100),
    @CancelAtPeriodEnd      BIT,             -- 1 = schedule cancel, 0 = undo cancel
    @Result                 INT OUTPUT       -- 0=success, 1=order not found
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE [PartnerHub].[ProviderOrder]
    SET
        CancelAtPeriodEnd   = @CancelAtPeriodEnd,
        UpdatedAt           = SYSUTCDATETIME()
    WHERE TravelProviderId      = @TravelProviderId
      AND StripeSubscriptionId  = @StripeSubscriptionId
      AND Status                = 'Active';

    SET @Result = CASE WHEN @@ROWCOUNT > 0 THEN 0 ELSE 1 END;

    IF @Result = 0
    BEGIN
        -- Return updated plan status for immediate frontend refresh
        EXEC [PartnerHub].[sp_portal_Order_GetPlanStatus]
            @TravelProviderId = @TravelProviderId;
    END
END;
GO
```

13. Webhook Deep-Dive

For each required Stripe webhook: the full payload structure (abbreviated), which fields we extract, and exactly what action is taken in the database or application layer.

⚠ Security: Verify Before Processing

Always verify the webhook signature first. Reject any request where ConstructEvent() throws — return 400. For all other cases (event processed or intentionally skipped) return 200 so Stripe does not retry.

13.1 invoice.paid

Fired after every successful payment — initial checkout, monthly renewal, annual renewal, and proration invoices on plan upgrades. This is the primary event that keeps ValidTo current.

Stripe Payload (abbreviated)

```
{
  "type": "invoice.paid",
  "data": {
    "object": {
      "id": "in_1Xxx",                          // Invoice ID
      "subscription": "sub_1Xxx",                // → lookup ProviderOrder
      "customer": "cus_1Xxx",
      "amount_paid": 9900,                       // cents → / 100 = 99.00
      "currency": "eur",
      "status": "paid",
      "lines": {
        "data": [{
          "period": {
            "start": 1741000000,                 // Unix UTC
            "end":   1743678400                  // → ValidTo
          }
        }]
      }
    }
  }
}
```

Fields we extract and actions taken

Payload Field	Example Value	What we do with it
`data.object.subscription`	sub_1Xxx	Lookup ProviderOrder by StripeSubscriptionId.
`data.object.lines.data[0].period.end`	1743678400	Convert Unix → UTC datetime → ProviderOrder.ValidTo. This extends the access window for the next billing period.
`data.object.amount_paid / 100`	99.00	ProviderOrder.AmountPaid — updated on every renewal for reporting.
`data.object.currency`	eur	ProviderOrder.Currency (should always be EUR; stored for audit).

SP called: sp_portal_Order_UpdateBySubscription (@EventType = 'invoice.paid')

⚠ FREE Plan Handling
Also fires for €0 FREE plan renewals — update ValidTo to NULL or ignore (FREE has ValidTo=NULL). Backend should detect amount_paid = 0 and skip ValidTo update to preserve NULL.

13.2 invoice.payment_failed

Fired when a recurring payment attempt fails. Stripe will automatically retry (Smart Retries). We set Status=PastDue but do NOT revoke access — the provider retains features during the retry window.

Stripe Payload (abbreviated)

```
{
  "type": "invoice.payment_failed",
  "data": {
    "object": {
      "id": "in_1Xxx",
      "subscription": "sub_1Xxx",               // → lookup ProviderOrder
      "customer": "cus_1Xxx",
      "attempt_count": 1,                       // 1, 2, 3 — Stripe retries up to 4x
      "next_payment_attempt": 1741086400,        // Unix UTC of next retry
      "amount_due": 9900,
      "hosted_invoice_url": "https://invoice.stripe.com/..."  // → send in email
    }
  }
}
```

Fields we extract and actions taken

Payload Field	Example Value	What we do with it
`data.object.subscription`	sub_1Xxx	Lookup ProviderOrder by StripeSubscriptionId → SET Status=PastDue.
`data.object.attempt_count`	1	Used in email subject: "Payment attempt 1 of 4 failed". No DB write.
`data.object.next_payment_attempt`	1741086400	Convert to readable date for email body: "Next retry: 15 March 2026".
`data.object.hosted_invoice_url`	https://invoice.stripe.com/...	Include as "Pay Now" button in payment-failed email to provider.

SP called: sp_portal_Order_UpdateBySubscription (@EventType = 'invoice.payment_failed')

Additional action: trigger transactional email to provider with "Pay Now" link and next retry date.

13.3 invoice.payment_action_required

Fired when Strong Customer Authentication (3D Secure / SCA) is required to complete the payment. Common for European cards. Access is not revoked — provider must re-authenticate.

Stripe Payload (abbreviated)

```
{
  "type": "invoice.payment_action_required",
  "data": {
    "object": {
      "subscription": "sub_1Xxx",
      "payment_intent": "pi_1Xxx",
      "hosted_invoice_url": "https://invoice.stripe.com/...",  // → 3DS link
      "amount_due": 9900
    }
  }
}
```

Fields we extract and actions taken

Payload Field	Example Value	What we do with it

`data.object.subscription`	sub_1Xxx	Lookup ProviderOrder → SET Status=PastDue (same as payment_failed).
`data.object.hosted_invoice_url`	https://invoice.stripe.com/...	Send email: "Your payment requires authentication — click here to complete."

SP called: sp_portal_Order_UpdateBySubscription (@EventType = 'invoice.payment_failed') — same DB action.

13.4 customer.subscription.created

Fired when a new Stripe subscription is created — including the initial FREE plan subscription at registration. For FREE: ProviderOrder is already created synchronously by sp_portal_Order_CreateFree, so this webhook is used only for verification / idempotency check.

Stripe Payload (abbreviated)

```
{
  "type": "customer.subscription.created",
  "data": {
    "object": {
      "id": "sub_1Xxx",                          // StripeSubscriptionId
      "customer": "cus_1Xxx",
      "status": "active",
      "items": {
        "data": [{
          "price": {
            "id": "price_free_v1",                // → resolve ProductId
            "metadata": { "plan_code": "FREE" }
          }
        }]
      },
      "current_period_end": 1743678400
    }
  }
}
```

Fields we extract and actions taken

Payload Field	Example Value	What we do with it
`data.object.id`	sub_1Xxx	Check if ProviderOrder already exists. If yes: no-op (idempotent). If no: run sp_portal_Order_CreateFree as fallback.
`data.object.items.data[0].price.id`	price_free_v1	Verify ProductId in database matches. Log mismatch as warning.

13.5 customer.subscription.updated

The most important subscription event. Fired on: plan upgrade/downgrade, cancel_at_period_end flag change, and status transitions. Always inspect both items[0].price.id and cancel_at_period_end on every event — both may change simultaneously.

Stripe Payload (abbreviated)

```
{
  "type": "customer.subscription.updated",
  "data": {
    "object": {
      "id": "sub_1Xxx",
      "status": "active",                        // active | past_due | canceled | paused
      "cancel_at_period_end": true,              // → CancelAtPeriodEnd
      "current_period_end": 1743678400,
      "items": {
        "data": [{
          "id": "si_1Xxx",                       // subscription item ID (for future
updates)
          "price": {
            "id": "price_advanced_monthly_v1",  // → resolve new ProductId
            "metadata": { "plan_code": "ADVANCED" }
          }
        }]
      }
    },
    "previous_attributes": {
      "items": { ... },                          // previous price — use to detect plan
change
      "cancel_at_period_end": false
    }
  }
}
```

Fields we extract and actions taken

Payload Field	Example Value	What we do with it
`data.object.id`	sub_1Xxx	Lookup ProviderOrder by StripeSubscriptionId.
`data.object.items.data[0].price.id`	price_advanced_monthly_v1	Resolve to ProductId via Product.StripePriceId → update ProviderOrder.ProductId if changed.
`data.object.cancel_at_period_end`	true	ProviderOrder.CancelAtPeriodEnd = 1 or 0.
`data.object.status`	active	Map to ProviderOrder.Status: active→Active, past_due→PastDue, canceled→Cancelled, paused→Paused.
`data.previous_attributes`	{ items: ... }	Compare previous price.id to new price.id. If changed AND it is a downgrade: call fn_GetProviderLimits and set GracePeriodEnd if over limit.

SP called: sp_portal_Order_UpdateBySubscription (@EventType = 'customer.subscription.updated')

ℹ Store si_xxx
Store subscription item ID (si_xxx) from items.data[0].id somewhere accessible — needed when calling stripe.subscriptions.update() for future plan changes. Best stored in TravelProvider.StripeSubscriptionItemId.

13.6 customer.subscription.deleted

Fired when a subscription is fully cancelled — either because cancel_at_period_end was true and the period ended, or because of immediate cancellation (e.g. admin action in Stripe dashboard). ValidTo is NOT modified — the provider retains access until the already-paid period expires.

Stripe Payload (abbreviated)

```
{
  "type": "customer.subscription.deleted",
  "data": {
    "object": {
      "id": "sub_1Xxx",
      "status": "canceled",
      "ended_at": 1743678400,                 // Unix UTC when subscription ended
      "cancel_at_period_end": true,
      "current_period_end": 1743678400
    }
  }
}
```

Fields we extract and actions taken

Payload Field	Example Value	What we do with it
`data.object.id`	sub_1Xxx	Lookup ProviderOrder → SET Status=Cancelled. ValidTo stays as-is.
`data.object.current_period_end`	1743678400	Verify matches existing ValidTo. If mismatch: update ValidTo to this value (safety net).

SP called: sp_portal_Order_UpdateBySubscription (@EventType = 'customer.subscription.deleted')

13.7 payment_intent.succeeded

Fired when a one-time payment is successfully confirmed. This is the trigger for creating ProviderOrder rows for Badge, ContentUpgrade, and AppPlacement products. productId and refTravelOfferId must be in the PaymentIntent metadata.

Stripe Payload (abbreviated)

```
{
  "type": "payment_intent.succeeded",
  "data": {
    "object": {
      "id": "pi_1Xxx",                        // StripePaymentIntentId
      "customer": "cus_1Xxx",                 // → resolve TravelProviderId
      "amount": 4900,                         // cents → / 100 = 49.00
      "currency": "eur",
      "status": "succeeded",
      "metadata": {
        "productId": "42",                    // → ProviderOrder.ProductId
        "refTravelOfferId": "117"             // → ProviderOrder.RefTravelOfferId
(nullable)
      }
    }
  }
}
```

Fields we extract and actions taken

Payload Field	Example Value	What we do with it
`data.object.id`	pi_1Xxx	ProviderOrder.StripePaymentIntentId — also used as idempotency key; duplicate events are silently skipped.
`data.object.customer`	cus_1Xxx	Resolve TravelProviderId via TravelProvider.StripeCustomerId.
`data.object.metadata.productId`	42	ProviderOrder.ProductId. Used to determine OrderType and DurationDays.
`data.object.metadata.refTravelOfferId`	117	ProviderOrder.RefTravelOfferId — NULL for Provider-scope products (Badge).
`data.object.amount / 100`	49.00	ProviderOrder.AmountPaid.

SP called: sp_portal_Order_Create

13.8 payment_intent.payment_failed

Fired when a one-time payment attempt fails. Because no ProviderOrder has been created yet (we only insert on success), there is no DB write. The frontend receives an error from Stripe.js directly and shows an inline error message — no webhook action needed.

Fields we extract and actions taken

Payload Field	Example Value	What we do with it
`data.object.last_payment_error.message`	Your card was declined.	Optional: log to application log for support visibility. No DB write.
`data.object.metadata.productId`	42	Optional: log for diagnostics. No action.

SP called: none. Frontend handles payment failure inline via Stripe.js confirmPayment() error response.

13.9 charge.refunded

Fired when a payment is refunded — typically initiated from the Stripe Dashboard by an admin. For one-time products this immediately revokes access by setting Status=Cancelled and ValidTo=NOW. For subscription refunds, the subscription events handle access — no extra DB action needed here.

Stripe Payload (abbreviated)

```
{
  "type": "charge.refunded",
  "data": {
    "object": {
      "id": "ch_1Xxx",
      "payment_intent": "pi_1Xxx",              // → lookup ProviderOrder (OneTime only)
      "amount_refunded": 4900,
      "refunded": true,
      "metadata": {
        "productId": "42"                       // set at PaymentIntent creation
      }
    }
  }
}
```

Fields we extract and actions taken

Payload Field	Example Value	What we do with it
`data.object.payment_intent`	pi_1Xxx	Lookup ProviderOrder by StripePaymentIntentId. If found and Status=Active: SET Status=Cancelled, ValidTo=NOW. If no row (subscription refund): no-op.
`data.object.amount_refunded`	4900	Log for audit. No DB write to ProviderOrder (AmountPaid stays as charged amount for revenue records).

SP called: sp_portal_Order_UpdateByPaymentIntent (@EventType = 'charge.refunded')

13.10 invoice.upcoming (optional)

Fired 7 days before the next renewal invoice. No DB write required — used purely to send a renewal reminder email. Stripe sends this only if the subscription has an upcoming_invoice configured.

Fields we extract and actions taken

Payload Field	Example Value	What we do with it
`data.object.subscription`	sub_1Xxx	Resolve provider email via ProviderOrder → TravelProvider.
`data.object.amount_due / 100`	99.00	Include in email: "Your plan renews for €99.00 on {date}".
`data.object.next_payment_attempt`	1743678400	Convert to readable date for email body.

SP called: none. Email only.

13.11 Webhook Registration Checklist

Register the following events in Stripe Dashboard → Developers → Webhooks → Add Endpoint. Endpoint URL: https://api.partnerhub.de/api/webhooks/stripe

Event Name	Priority	DB Write	SP Called
`invoice.paid`	**HIGH**	✓	sp_portal_Order_UpdateBySubscription
`invoice.payment_failed`	**HIGH**	✓	sp_portal_Order_UpdateBySubscription + email
`invoice.payment_action_required`	**HIGH**	✓	sp_portal_Order_UpdateBySubscription + email
`customer.subscription.created`	MEDIUM	✓*	sp_portal_Order_CreateFree (fallback only)
`customer.subscription.updated`	**HIGH**	✓	sp_portal_Order_UpdateBySubscription
`customer.subscription.deleted`	**HIGH**	✓	sp_portal_Order_UpdateBySubscription
`payment_intent.succeeded`	**HIGH**	✓	sp_portal_Order_Create
`payment_intent.payment_failed`	MEDIUM	—	none (frontend handles via Stripe.js)
`charge.refunded`	**MEDIUM**	✓	sp_portal_Order_UpdateByPaymentIntent
`invoice.upcoming`	LOW	—	none — renewal reminder email only

⚠ Footnote

* customer.subscription.created: ProviderOrder is normally created synchronously by sp_portal_Order_CreateFree during registration. This webhook serves as a fallback idempotency check only.