Webhook Handler Fix - Payment Record Creation

Date: 2025-12-25

Problem Summary

The Stripe webhook was receiving `payment_intent.succeeded` events successfully, but **Payment records were not being created in the database**. This caused:

- Booking status remained `PENDING` instead of `CONFIRMED`
- No payment records in the database
- Creators couldn't see pending balances
- Payment tracking was broken

Root Causes Identified

1. Insufficient Metadata Validation

The original code extracted metadata but didn't validate it properly:

```
// OLD CODE - Weak validation
const platformFee = Number(paymentIntent.metadata?.platformFee || 0);
const creatorAmount = Number(paymentIntent.metadata?.creatorAmount || 0);
```

Issues:
- If metadata was missing, values defaulted to `0`
- No check for `undefined` or malformed strings
- No early exit on validation failure
- Errors were logged but execution continued

2. Silent Error Handling

The original code caught payment creation errors but continued processing:

```
// OLD CODE - Silent failure
catch (paymentError: any) {
  console.error('ERROR creating payment record:', paymentError);
  // Continue with rest of webhook processing  ← THIS IS THE PROBLEM
}
```

Issues:
- Webhook returned 200 OK even when payment creation failed
- Stripe thought the webhook succeeded
- No retry mechanism triggered
- Errors were hidden in logs

3. Insufficient Logging

The original logging didn't provide enough detail to debug:
- No visibility into metadata extraction
- No type information for parsed values
- No validation checkpoints
- Generic error messages

4. Data Type Concerns

Prisma's `Decimal` type might be strict about accepting plain numbers, but the code didn't handle this explicitly.

Solution Implemented

1. Strict Metadata Validation with Early Exit

```
// NEW CODE - Strict validation
const metadataPlatformFee = paymentIntent.metadata?.platformFee;
const metadataCreatorAmount = paymentIntent.metadata?.creatorAmount;

try {
  if (!metadataPlatformFee) {
    throw new Error('platformFee is missing from metadata');
  }
  platformFee = parseFloat(metadataPlatformFee);
  if (isNaN(platformFee) || platformFee < 0) {
    throw new Error(`Invalid platformFee value: "${metadataPlatformFee}"`);
  }
  // Similar validation for creatorAmount...
} catch (metadataError) {
  console.error('METADATA PARSING ERROR:', metadataError.message);
  return NextResponse.json({ received: true, error: 'Invalid metadata' }, { status: 20
0 });
}
```

Benefits:
- ✅ Validates metadata exists before parsing
- ✅ Uses `parseFloat()` for proper string-to-number conversion
- ✅ Checks for `NaN` and invalid values
- ✅ **Returns early** if validation fails - doesn't try to create invalid payment
- ✅ Prevents database corruption

2. Amount Validation with Business Logic Check

```
// Validate amount from booking
if (!amount || amount <= 0 || isNaN(amount)) {
  console.error('VALIDATION ERROR: Invalid amount from booking');
  return NextResponse.json({ received: true, error: 'Invalid booking amount' }, { stat
us: 200 });
}

// Verify fee calculation (platformFee + creatorAmount should equal amount)
const expectedTotal = platformFee + creatorAmount;
const difference = Math.abs(amount - expectedTotal);
if (difference > 0.02) { // 2 cent tolerance for floating point
  console.warn('WARNING: Fee calculation mismatch!');
  console.warn('Total:', amount, 'EUR vs Expected:', expectedTotal, 'EUR');
}
```

Benefits:
- ✅ Validates booking amount is valid
- ✅ Cross-checks metadata values against booking total
- ✅ Detects calculation errors in payment intent creation
- ✅ Provides early warning of data inconsistencies

3. Critical Error Handling - Throw on Failure

```
// NEW CODE - Throw error to trigger Stripe retry
catch (paymentError: any) {
  console.error('ERROR CREATING PAYMENT RECORD');
  console.error('Full error:', JSON.stringify(paymentError, null, 2));

  // Detailed Prisma error handling
  if (paymentError?.code === 'P2002') {
    console.error('Unique constraint violation');
  } else if (paymentError?.code === 'P2003') {
    console.error('Foreign key constraint failed');
  }
  // ... more error codes

  // CRITICAL: Throw error to trigger Stripe webhook retry
  throw new Error(`Failed to create payment record: ${paymentError?.message}`);
}
```

Benefits:
- ✅ **Throws error** instead of silently continuing
- ✅ Webhook returns error status to Stripe
- ✅ **Stripe automatically retries** the webhook
- ✅ Detailed Prisma error code handling
- ✅ Better debugging information

4. Comprehensive Logging

Added detailed logs at every step:

```
console.log('========================================');
console.log('PAYMENT CREATION - Extracting metadata...');
console.log('Raw metadata:', JSON.stringify(paymentIntent.metadata, null, 2));
console.log('Extracted metadata values (as strings):', {
  platformFee: metadataPlatformFee,
  creatorAmount: metadataCreatorAmount,
  type: `platformFee: ${typeof metadataPlatformFee}, creatorAmount: ${typeof metadataC
reatorAmount}`,
});
console.log('Metadata values parsed successfully:', {
  platformFee: `${platformFee} EUR`,
  creatorAmount: `${creatorAmount} EUR`,
});
console.log('Database values (all in EUR):');
console.log('  - bookingId:', booking.id);
console.log('  - amount:', amount);
console.log('  - platformFee:', platformFee);
// ... more logs
console.log('========================================');
```

Benefits:
- ✅ Complete visibility into metadata extraction
- ✅ Type information for debugging
- ✅ Clear validation checkpoints
- ✅ Easy to trace execution flow
- ✅ Formatted output for readability

5. Variable Scoping Fix

Fixed scoping issues where variables weren't accessible in email templates:

```
// Declare variables at function scope (before if/else)
let amount: number;
let platformFee: number;
let creatorAmount: number;
let payoutReleaseDate: Date;

// Set from existing payment OR from metadata
if (existingPayment) {
  amount = Number(existingPayment.amount);
  // ...
} else {
  amount = Number(booking.totalPrice);
  // ...
}

// Now variables are accessible in email templates below
```

Files Modified

- `app/api/payments/webhook/route.ts` - Complete rewrite of payment creation logic

Testing Recommendations

1. Test Valid Payment Flow

- Create a new booking
- Complete payment via Stripe
- Verify webhook is called
- Check logs for successful payment creation
- Verify Payment record exists in database
- Verify Booking status is CONFIRMED

2. Test Missing Metadata

- Manually trigger webhook with missing `platformFee` in metadata
- Should see “platformFee is missing from metadata” error
- Webhook should return 200 but skip payment creation

3. Test Invalid Metadata

- Send webhook with `platformFee: "invalid"`
- Should see “Invalid platformFee value” error
- Webhook should return 200 but skip payment creation

4. Test Database Errors

- Test with non-existent booking ID
- Should see Prisma P2003 error (foreign key constraint)
- Webhook should throw error and Stripe should retry

5. Test Duplicate Payment

- Send same webhook twice
- Second time should use existing payment
- Should log “Payment record already exists”

Expected Behavior After Fix

✅ **Successful Payment:**
1. Webhook receives `payment_intent.succeeded`
2. Extracts and validates metadata
3. Creates Payment record with:
- amount: 70.00 EUR
- platformFee: 7.00 EUR
- creatorAmount: 63.00 EUR
- payoutStatus: HELD
- payoutReleaseDate: +7 days
4. Updates Booking status to CONFIRMED
5. Sends confirmation emails
6. Returns 200 OK

❌ **Invalid Metadata:**
1. Webhook receives event
2. Metadata validation fails

3. Logs detailed error
4. Returns 200 OK (to prevent retry)
5. No payment record created

❌ **Database Error:**
1. Webhook receives event
2. Metadata validation passes
3. Database operation fails
4. Logs detailed Prisma error
5. **Throws error** - webhook returns 4xx/5xx
6. **Stripe retries webhook**

Key Improvements

Aspect	Before	After
Validation	Weak (0 defaults)	Strict (early exit)
Error Handling	Silent (continue)	Explicit (throw)
Logging	Basic	Comprehensive
Retry Logic	None (always 200)	Automatic (Stripe retry)
Data Integrity	At risk	Protected
Debugging	Difficult	Easy

Monitoring After Deployment

Watch for these log patterns:

✅ Success Pattern:

```
========================================
WEBHOOK: payment_intent.succeeded received
PAYMENT CREATION - Extracting metadata...
✅ Metadata values parsed successfully
Creating payment record in database...
✅✅✅ PAYMENT RECORD CREATED SUCCESSFULLY! ✅✅✅
========================================
```

❌ Metadata Error Pattern:

```
PAYMENT CREATION - Extracting metadata...
❌ METADATA PARSING ERROR: platformFee is missing from metadata
Skipping payment creation for booking: xxx
```

❌ Database Error Pattern:

```
Creating payment record in database...
❌❌❌ ERROR CREATING PAYMENT RECORD ❌❌❌
Error code: P2003
❌ Foreign key constraint failed
```

Next Steps

1. **Deploy the fix** to staging environment
2. **Test with test Stripe keys** first
3. **Monitor logs** for the patterns above
4. **Verify database** - check Payment records are created
5. **Test end-to-end** - complete a real booking
6. **Deploy to production** once validated
7. **Set up alerts** for webhook failures

Related Files

- `lib/stripe.ts` - Payment intent creation (already fixed)
- `app/api/payments/create-intent/route.ts` - Sets metadata (working correctly)
- `prisma/schema.prisma` - Payment model schema (Decimal fields)

Technical Notes

Metadata Format

Stripe metadata stores values as strings:
- `platformFee: "7"` (EUR as string)
- `creatorAmount: "63"` (EUR as string)

Must be parsed with `parseFloat()` before use.

Database Schema

Payment model uses `Decimal @db.Decimal(10, 2)` for currency:
- Stores EUR values (not cents)
- Example: 70.00 EUR, not 7000 cents
- JavaScript numbers are automatically converted by Prisma

Stripe Webhook Retry Logic

- Webhook returns 2xx: Stripe considers it successful, no retry
- Webhook returns 4xx/5xx: Stripe retries with exponential backoff
- Max retries: 3 attempts over ~24 hours

Conclusion

The webhook handler now:
- ✅ Validates all metadata before processing
- ✅ Returns early on validation errors

- ✅ Throws errors on database failures (triggers Stripe retry)
- ✅ Logs comprehensive debugging information
- ✅ Protects data integrity
- ✅ Enables automatic recovery via Stripe retry mechanism

This should completely resolve the issue of payment records not being created.