Cron Jobs Improvements Documentation

This document describes all improvements made to the existing cron jobs and the new cron jobs created for the Call a Star platform.

Table of Contents

Overview

What Changed

All cron jobs now have **mandatory detailed logging** that tracks:
- **Type** of cron job
- **Number of items processed**
- **Execution status** (SUCCESS or ERROR)
- **Duration** (in milliseconds and seconds)
- **Start and end timestamps**
- **Detailed context** (errors, items processed, etc.)

Why These Changes

1. **100% Traceability**: Every cron execution is now logged in the database, making debugging and monitoring easy
2. **Performance Monitoring**: Duration tracking helps identify slow operations
3. **Error Tracking**: All errors are captured with full stack traces
4. **Audit Trail**: Complete history of what was processed and when

---

Improvements to Existing Crons

1. cleanup-logs

File: `/app/api/cron/cleanup-logs/route.ts`

What Was Improved:
- Added `logCronRun()` for successful executions with detailed stats
- Added `logCronError()` for failed executions

- Duration tracking from start to finish
- Removed dependency on `system-logger` in favor of unified `logger` module

Logs Generated:

```
// Success log
{
  cronType: 'cleanup-logs',
  itemsProcessed: 150,  // Total logs deleted
  duration: 2341,       // Milliseconds
  infoDeleted: 100,
  warningDeleted: 30,
  errorDeleted: 20,
  status: 'SUCCESS'
}

// Error log (if failure)
{
  cronType: 'cleanup-logs',
  duration: 156,
  error: 'Database connection failed',
  status: 'ERROR'
}
```

---

2. process-payouts

File: `/app/api/cron/process-payouts/route.ts`

What Was Improved:
- Added `logCronRun()` for successful processing with detailed stats
- Added `logCronError()` for fatal errors
- Comprehensive tracking of succeeded/failed/skipped payouts
- Duration tracking and detailed error reporting
- Removed dependency on `system-logger` in favor of unified `logger` module

Logs Generated:

```
// Success log
{
  cronType: 'process-payouts',
  itemsProcessed: 25,      // Total processed
  duration: 12543,         // Milliseconds
  succeeded: 20,
  failed: 2,
  skipped: 3,
  errors: [
    { creatorId: 'xyz', error: 'Insufficient balance' }
  ],
  status: 'SUCCESS'
}

// Error log (if fatal failure)
{
  cronType: 'process-payouts',
  duration: 1234,
  itemsProcessed: 10,
  error: 'Database transaction failed',
  status: 'ERROR'
}
```

---

3. process-automatic-payouts

File: `/app/api/cron/process-automatic-payouts/route.ts`

What Was Improved:
- Added `logCronRun()` for successful processing
- Added `logCronError()` for fatal errors
- Detailed tracking of processed/skipped/failed payouts
- Duration tracking and error reporting
- Removed dependency on `system-logger` in favor of unified `logger` module

Logs Generated:

```
// Success log
{
  cronType: 'process-automatic-payouts',
  itemsProcessed: 15,        // Total processed
  duration: 8765,            // Milliseconds
  totalCreators: 20,
  skipped: 3,
  failed: 2,
  totalAmount: 5420.50,      // Total paid out
  status: 'SUCCESS'
}

// Error log (if fatal failure)
{
  cronType: 'process-automatic-payouts',
  duration: 2341,
  error: 'Stripe API error',
  status: 'ERROR'
}
```

New Cron Jobs

4. cleanup-daily-rooms ✨ NEW

File: `/app/api/cron/cleanup-daily-rooms/route.ts`

Purpose:
Automatically cleans up Daily.io video rooms for completed or cancelled bookings.

How It Works:
1. Finds all bookings with status `COMPLETED` or `CANCELLED` that still have a `dailyRoomName`
2. Checks if the room exists on Daily.io using `checkDailyRoomExists()`
3. Deletes the room if it exists using `deleteDailyRoom()`
4. Updates the booking to remove `dailyRoomName` and `dailyRoomUrl`
5. Logs each deletion with `logDailyRoomDeleted()` or errors with `logDailyRoomError()`

Why This Is Important:
- Prevents unnecessary charges from Daily.io for unused rooms
- Keeps the Daily.io account clean and organized
- Ensures no old rooms are left hanging after calls end

Logs Generated:

```
// Success log
{
  cronType: 'cleanup-daily-rooms',
  itemsProcessed: 12,          // Total rooms processed
  duration: 3456,              // Milliseconds
  roomsDeleted: 8,             // Successfully deleted
  roomsAlreadyDeleted: 3,      // Already gone
  roomsFailed: 1,              // Failed to delete
  status: 'SUCCESS'
}

// Per-room success log
{
  type: 'DAILY_ROOM_DELETED',
  roomId: 'room-abc-123',
  bookingId: 'clx123456',
  roomUrl: 'https://example.daily.co/room-abc-123',
  bookingStatus: 'COMPLETED',
  deletedBy: 'cron_job',
  status: 'SUCCESS'
}

// Per-room error log
{
  type: 'DAILY_ROOM_ERROR',
  roomId: 'room-xyz-789',
  bookingId: 'clx789012',
  error: 'Room not found',
  operation: 'delete',
  status: 'ERROR'
}
```

Recommended Schedule: Every hour (`0 * * * *`)

5. send-booking-reminders ✨ NEW

File: `/app/api/cron/send-booking-reminders/route.ts`

Purpose:
Sends email reminders to both clients and creators 15 minutes before their scheduled calls.

How It Works:
1. Finds all `CONFIRMED` bookings starting in 15-20 minutes
2. Checks if a reminder has already been sent (`reminderSent = false`)
3. Sends a reminder email to the **client**
4. Sends a reminder email to the **creator**
5. Marks the booking as `reminderSent = true`
6. Logs each email with `logEmailSent()` or errors with `logEmailError()`

Email Details:
- **Language:** English only
- **Template:** `generateBookingReminderEmail()` in `lib/email.ts`
- **Link:** Platform booking page (e.g., `/bookings/[id]`), not Daily.io direct link
- **Recipients:** Both client and creator

Why This Is Important:
- Ensures both parties are reminded before the call
- Increases show-up rates and reduces no-shows
- Professional user experience

Logs Generated:

```
// Success log
{
  cronType: 'send-booking-reminders',
  itemsProcessed: 10,          // Total emails sent
  duration: 4567,              // Milliseconds
  totalBookings: 5,            // Bookings processed
  emailsSent: 10,              // 2 per booking (client + creator)
  emailsFailed: 0,
  timeWindow: {
    minTime: '2024-12-31T14:15:00Z',
    maxTime: '2024-12-31T14:20:00Z'
  },
  status: 'SUCCESS'
}

// Per-email success log (client)
{
  type: 'EMAIL_SENT',
  bookingId: 'clx123456',
  userId: 'user123',
  emailType: 'booking_reminder_client',
  recipientEmail: 'client@example.com',
  recipientName: 'John Doe',
  creatorName: 'Jane Creator',
  callDateTime: '2024-12-31T14:30:00Z',
  status: 'SUCCESS'
}

// Per-email success log (creator)
{
  type: 'EMAIL_SENT',
  bookingId: 'clx123456',
  userId: 'creator456',
  emailType: 'booking_reminder_creator',
  recipientEmail: 'creator@example.com',
  recipientName: 'Jane Creator',
  clientName: 'John Doe',
  callDateTime: '2024-12-31T14:30:00Z',
  status: 'SUCCESS'
}

// Per-email error log
{
  type: 'EMAIL_ERROR',
  bookingId: 'clx123456',
  userId: 'user123',
  emailType: 'booking_reminder_client',
  error: 'SMTP connection failed',
  recipientEmail: 'client@example.com',
  status: 'ERROR'
}
```

Recommended Schedule: Every 5 minutes (`*/5 * * * *`)

Logging System

All cron jobs use the unified logging system from `lib/logger.ts` .

Key Functions

`logCronRun(cronType, itemsProcessed, duration, additionalContext?)`

Logs a successful cron execution.

Example:

```
await logCronRun(
  'cleanup-logs',
  150,  // Items processed
  2341, // Duration in ms
  {
    startTime: '2024-12-31T02:00:00Z',
    endTime: '2024-12-31T02:00:02Z',
    infoDeleted: 100,
    warningDeleted: 30,
    errorDeleted: 20
  }
);
```

`logCronError(cronType, error, additionalContext?)`

Logs a cron execution error.

Example:

```
await logCronError(
  'cleanup-logs',
  error,
  {
    startTime: '2024-12-31T02:00:00Z',
    endTime: '2024-12-31T02:00:02Z',
    duration: 2341
  }
);
```

`logEmailSent(bookingId, userId, emailType, additionalContext?)`

Logs a successful email send.

Example:

```
await logEmailSent(
  'clx123456',
  'user123',
  'booking_reminder_client',
  {
    recipientEmail: 'client@example.com',
    recipientName: 'John Doe',
    creatorName: 'Jane Creator'
  }
);
```

`logEmailError(bookingId, userId, emailType, error, additionalContext?)`

Logs an email sending error.

Example:

```
await logEmailError(
  'clx123456',
  'user123',
  'booking_reminder_client',
  error,
  {
    recipientEmail: 'client@example.com'
  }
);
```

`logDailyRoomDeleted(roomId, bookingId, additionalContext?)`

Logs a successful Daily.io room deletion.

Example:

```
await logDailyRoomDeleted(
  'room-abc-123',
  'clx123456',
  {
    roomUrl: 'https://example.daily.co/room-abc-123',
    bookingStatus: 'COMPLETED',
    deletedBy: 'cron_job'
  }
);
```

`logDailyRoomError(roomId, bookingId, error, additionalContext?)`

Logs a Daily.io room operation error.

Example:

```
await logDailyRoomError(
  'room-abc-123',
  'clx123456',
  error,
  {
    operation: 'delete',
    bookingStatus: 'COMPLETED'
  }
);
```

Database Changes

New Field: `reminderSent` in Booking Model

Migration: `20251231134640_add_reminder_sent_to_booking`

Schema Change:

```
model Booking {
  // ... existing fields ...
  reminderSent Boolean @default(false) // Flag to track if reminder email was sent

  @@index([reminderSent])
}
```

Purpose:
Prevents duplicate reminder emails from being sent if the cron runs multiple times within the 15-20 minute window.

SQL Migration:

```
ALTER TABLE "Booking" ADD COLUMN "reminderSent" BOOLEAN NOT NULL DEFAULT false;
CREATE INDEX "Booking_reminderSent_idx" ON "Booking"("reminderSent");
```

Email Templates

New Template: `generateBookingReminderEmail()`

File: `lib/email.ts`

Purpose:
English-only reminder email template for bookings starting in 15 minutes.

Parameters:

```
{
  userName: string,        // Recipient's name
  creatorName: string,     // Other person's name (creator or client)
  callDateTime: Date,      // Scheduled call time
  bookingUrl: string       // Platform booking URL (not Daily.io)
}
```

Features:
- Clean, professional design
- Clear call-to-action button
- Formatted date and time with timezone
- Platform branding

Example Usage:

```
const emailHtml = generateBookingReminderEmail({
  userName: 'John Doe',
  creatorName: 'Jane Creator',
  callDateTime: new Date('2024-12-31T14:30:00Z'),
  bookingUrl: 'https://callastar.com/bookings/clx123456'
});

await sendEmail({
  to: 'john@example.com',
  subject: '⏰ Reminder: Your call is starting soon!',
  html: emailHtml
});
```

Updated: `checkDailyRoomExists()` Function

File: `lib/daily.ts`

Purpose:
Checks if a Daily.io room exists before attempting to delete it.

Returns: `Promise<boolean>`
- `true` if room exists
- `false` if room doesn't exist (404 response)
- Throws error for other API errors

Example Usage:

```
const roomExists = await checkDailyRoomExists('room-abc-123');
if (roomExists) {
  await deleteDailyRoom('room-abc-123');
}
```

Testing

Manual Testing

You can manually trigger any cron job by making a POST request:

```
# Test cleanup-logs
curl -X POST "http://localhost:3000/api/cron/cleanup-logs" \
  -H "Authorization: Bearer your-cron-secret"

# Test process-payouts
curl -X POST "http://localhost:3000/api/cron/process-payouts" \
  -H "x-cron-secret: your-cron-secret"

# Test process-automatic-payouts
curl -X POST "http://localhost:3000/api/cron/process-automatic-payouts" \
  -H "Authorization: Bearer your-cron-secret"

# Test cleanup-daily-rooms
curl -X POST "http://localhost:3000/api/cron/cleanup-daily-rooms" \
  -H "Authorization: Bearer your-cron-secret"

# Test send-booking-reminders
curl -X POST "http://localhost:3000/api/cron/send-booking-reminders" \
  -H "Authorization: Bearer your-cron-secret"
```

Check Status (GET Endpoint)

```
# Check if cron endpoint is ready
curl "http://localhost:3000/api/cron/cleanup-logs"
```

View Logs

Query the `Log` table in the database:

```
-- View recent cron runs
SELECT * FROM "Log"
WHERE type = 'CRON_RUN'
ORDER BY "createdAt" DESC
LIMIT 20;

-- View recent cron errors
SELECT * FROM "Log"
WHERE type = 'CRON_ERROR'
ORDER BY "createdAt" DESC
LIMIT 20;

-- View logs for a specific cron
SELECT * FROM "Log"
WHERE context->>'cronType' = 'cleanup-logs'
ORDER BY "createdAt" DESC
LIMIT 10;

-- View recent email logs
SELECT * FROM "Log"
WHERE type IN ('EMAIL_SENT', 'EMAIL_ERROR')
ORDER BY "createdAt" DESC
LIMIT 20;

-- View recent Daily.io room logs
SELECT * FROM "Log"
WHERE type IN ('DAILY_ROOM_DELETED', 'DAILY_ROOM_ERROR')
ORDER BY "createdAt" DESC
LIMIT 20;
```

Summary of Changes

Existing Crons Improved (3)

1. ✅ `cleanup-logs` - Added detailed logging with logCronRun/logCronError
2. ✅ `process-payouts` - Added detailed logging with logCronRun/logCronError
3. ✅ `process-automatic-payouts` - Added detailed logging with logCronRun/logCronError

New Crons Created (2)

1. ✨ `cleanup-daily-rooms` - Automatic Daily.io room cleanup
2. ✨ `send-booking-reminders` - Email reminders 15 minutes before calls

New Functions Added

- `logCronRun()` - Log successful cron executions
- `logCronError()` - Log cron errors
- `checkDailyRoomExists()` - Check if Daily.io room exists
- `generateBookingReminderEmail()` - Generate reminder email HTML

Database Changes

- Added `reminderSent` field to `Booking` model
- Added index on `reminderSent` for performance

Documentation Created

- `CRON_SCHEDULE.md` - Recommended schedules and configuration
- `CRON_IMPROVEMENTS.md` - This document

Next Steps

1. **Deploy Database Migration:**
   `bash`
   `npx prisma migrate deploy`

2. **Configure Cron Schedules:**
   - Update `vercel.json` with the recommended schedules
   - Or create GitHub Actions workflows

3. **Set Environment Variables:**
   - Ensure `CRON_SECRET` is set in production
   - Ensure `NEXT_PUBLIC_APP_URL` is set correctly

4. **Monitor Logs:**
   - Check the `Log` table regularly
   - Set up alerts for `CRON_ERROR` entries
   - Monitor email delivery rates

5. **Test in Staging:**
   - Manually trigger all crons
   - Verify logs are created

- Verify emails are sent
- Verify Daily.io rooms are deleted

Questions or Issues?

If you encounter any issues or have questions about the cron system, please:

1. Check the logs in the database
2. Verify environment variables are set correctly
3. Test the cron endpoint manually
4. Review this documentation

For critical issues, contact the development team.

Last Updated: December 31, 2024
Version: 1.0
Author: Call a Star Development Team