📧 Email System Documentation - Call a Star

Overview

The email system in Call a Star is built with **100% traceability** in mind. Every email sent (successful or failed) is automatically logged to the database for audit and debugging purposes.

All email templates are **in English only** to provide a consistent user experience.

🎯 Email Types

1. Payment Confirmation Email

- **Type ID**: `payment_confirmation`
- **Trigger**: Immediately after a successful payment (Stripe webhook `payment_intent.succeeded`)
- **Recipients**: Customer who made the payment
- **Subject**: `✅ Payment Confirmed - Your Call is Booked!`
- **Template Function**: `generatePaymentConfirmationEmail()`
- **Content**:
- Payment confirmation message
- Booking summary (creator, date/time, duration, price)
- Link to booking page on the platform
- Important preparation instructions
- Professional thank you message

Example Log Context:

```json
{
  "bookingId": "clx123456",
  "userId": "user789",
  "emailType": "payment_confirmation",
  "recipientEmail": "user@example.com",
  "subject": "✅ Payment Confirmed - Your Call is Booked!",
  "emailId": "resend_abc123"
}
```

---

2. Booking Reminder Email (Client)

- **Type ID**: `booking_reminder_client`
- **Trigger**: 15 minutes before a confirmed call starts (cron job)
- **Recipients**: Customer who booked the call
- **Subject**: `⏰ Reminder: Your call is starting soon!`
- **Template Function**: `generateBookingReminderEmail()`
- **Content**:
- Friendly reminder that the call is starting in 15 minutes

- Call details (date, time, creator name)
- Link to booking page to join the call
- Encouragement message

Example Log Context:

```
{
  "bookingId": "clx123456",
  "userId": "user789",
  "emailType": "booking_reminder_client",
  "recipientEmail": "user@example.com",
  "recipientName": "John Doe",
  "creatorName": "Jane Creator",
  "callDateTime": "2025-12-31T14:00:00.000Z"
}
```

---

3. Booking Reminder Email (Creator)

- **Type ID**: `booking_reminder_creator`
- **Trigger**: 15 minutes before a confirmed call starts (cron job)
- **Recipients**: Creator hosting the call
- **Subject**: `⏰ Reminder: Your call is starting soon!`
- **Template Function**: `generateBookingReminderEmail()`
- **Content**: Same as client reminder but from the creator's perspective

Example Log Context:

```
{
  "bookingId": "clx123456",
  "userId": "creator456",
  "emailType": "booking_reminder_creator",
  "recipientEmail": "creator@example.com",
  "recipientName": "Jane Creator",
  "clientName": "John Doe",
  "callDateTime": "2025-12-31T14:00:00.000Z"
}
```

🔧 Technical Implementation

`sendEmail()` Function

The core email sending function with automatic verification and logging:

```
export async function sendEmail({
  to,
  subject,
  html,
  bookingId,    // Optional: for logging context
  userId,       // Optional: for logging context
  emailType     // Optional: for logging context
}: SendEmailOptions): Promise<SendEmailResult>
```

Features:
- ✅ Try-catch error handling (never throws)
- ✅ Response verification from Resend
- ✅ Automatic success/failure logging
- ✅ Returns `{ success: boolean, error?: string, emailId?: string }`
- ✅ Non-blocking: email failures never break the main flow

Return Value:

```
interface SendEmailResult {
  success: boolean;   // true if email was sent successfully
  error?: string;     // error message if failed
  emailId?: string;   // Resend email ID if successful
}
```

Example Usage:

```
const emailResult = await sendEmail({
  to: user.email,
  subject: '✅ Payment Confirmed - Your Call is Booked!',
  html: emailHtml,
  bookingId: booking.id,
  userId: user.id,
  emailType: 'payment_confirmation',
});

if (emailResult.success) {
  console.log('Email sent successfully!');
} else {
  console.error('Email failed:', emailResult.error);
}
```

Email Templates

All email templates are **HTML-based** and **responsive** for mobile devices.

1. `generatePaymentConfirmationEmail()`

```
generatePaymentConfirmationEmail({
  userName: string;
  creatorName: string;
  callTitle: string;
  callDateTime: Date;
  callDuration: number;
  totalPrice: number;
  bookingUrl: string;
  currency?: string;  // Default: 'EUR'
})
```

2. `generateBookingReminderEmail()`

```
generateBookingReminderEmail({
  userName: string;
  creatorName: string;
  callDateTime: Date;
  bookingUrl: string;  // IMPORTANT: Use platform URL, NOT Daily.io URL
})
```

📊 Logging System

All emails are logged using the centralized logging system in `lib/logger.ts`.

Success Logs

When an email is sent successfully, `logEmailSent()` is automatically called:

```
await logEmailSent(
  bookingId,
  userId,
  emailType,
  {
    recipientEmail: 'user@example.com',
    subject: '✅ Payment Confirmed',
    emailId: 'resend_abc123'
  }
);
```

Database Entry:
- **Table**: `Log`
- **Type**: `EMAIL_SENT`
- **Status**: `SUCCESS`
- **Context**: Contains all relevant metadata

Error Logs

When an email fails to send, `logEmailError()` is automatically called:

```
await logEmailError(
  bookingId,
  userId,
  emailType,
  error,
  {
    recipientEmail: 'user@example.com',
    subject: '✅ Payment Confirmed'
  }
);
```

Database Entry:
- **Table**: `Log`
- **Type**: `EMAIL_ERROR`

- **Status**: `ERROR`
- **Error**: Full stack trace included

🔍 How to Verify Email Delivery

1. Check Console Logs

All email operations are logged to the console:

```
✅ Success: "Email sent successfully: resend_abc123"
❌ Error: "Error sending email: <error details>"
```

2. Check Database Logs

Query the `Log` table to see all email operations:

```
-- Get all emails sent for a specific booking
SELECT * FROM "Log"
WHERE type IN ('EMAIL_SENT', 'EMAIL_ERROR')
AND context->>'bookingId' = 'clx123456'
ORDER BY "createdAt" DESC;

-- Get all failed emails in the last 24 hours
SELECT * FROM "Log"
WHERE type = 'EMAIL_ERROR'
AND "createdAt" >= NOW() - INTERVAL '24 hours'
ORDER BY "createdAt" DESC;

-- Get email delivery stats
SELECT
  type,
  COUNT(*) as count
FROM "Log"
WHERE type IN ('EMAIL_SENT', 'EMAIL_ERROR')
GROUP BY type;
```

3. Check Resend Dashboard

If Resend is configured, you can also check the Resend Dashboard (https://resend.com/emails) for real-time delivery status.

📈 Example Log Entries

Successful Email Send

```json
{
  "id": "log_abc123",
  "type": "EMAIL_SENT",
  "status": "SUCCESS",
  "message": "Email sent: payment_confirmation for booking clx123456",
  "context": {
    "bookingId": "clx123456",
    "userId": "user789",
    "emailType": "payment_confirmation",
    "recipientEmail": "user@example.com",
    "subject": "✅ Payment Confirmed - Your Call is Booked!",
    "emailId": "resend_abc123"
  },
  "error": null,
  "createdAt": "2025-12-31T10:30:00.000Z"
}
```

Failed Email Send

```json
{
  "id": "log_xyz789",
  "type": "EMAIL_ERROR",
  "status": "ERROR",
  "message": "Email failed: payment_confirmation for booking clx123456",
  "context": {
    "bookingId": "clx123456",
    "userId": "user789",
    "emailType": "payment_confirmation",
    "recipientEmail": "invalid@email",
    "subject": "✅ Payment Confirmed - Your Call is Booked!"
  },
  "error": "Invalid email address: invalid@email\n\nStack trace:\n...",
  "createdAt": "2025-12-31T10:30:00.000Z"
}
```

🚨 Error Handling Best Practices

1. Never Block Main Flow

Email errors should **never** block the main business logic:

```
// ✅ GOOD: Email wrapped in separate try-catch
try {
  const emailResult = await sendEmail({...});
  if (!emailResult.success) {
    console.error('Email failed but booking succeeded');
  }
} catch (error) {
  console.error('Unexpected email error:', error);
}

// Continue with main flow regardless of email status
```

2. Always Provide Context

Always pass `bookingId`, `userId`, and `emailType` for proper logging:

```
// ✅ GOOD: Full context provided
await sendEmail({
  to: user.email,
  subject: 'Email Subject',
  html: emailHtml,
  bookingId: booking.id,
  userId: user.id,
  emailType: 'payment_confirmation',
});

// ❌ BAD: No logging context
await sendEmail({
  to: user.email,
  subject: 'Email Subject',
  html: emailHtml,
});
```

3. Check Return Values

Always check the return value to handle failures gracefully:

```
const emailResult = await sendEmail({...});

if (emailResult.success) {
  console.log('✅ Email sent successfully');
} else {
  console.error('❌ Email failed:', emailResult.error);
  // Optionally: trigger admin notification or retry logic
}
```

🎨 Email Design Guidelines

- **Language**: English only
- **Tone**: Professional but friendly
- **Mobile-first**: All templates are responsive
- **Brand colors**: Purple gradient (#667eea to #764ba2)
- **Clear CTAs**: Prominent buttons for actions

- **Accessibility**: High contrast, readable fonts

🔐 Security & Configuration

Required Environment Variables

```
RESEND_API_KEY=re_xxxxxxxxxxxxx
EMAIL_FROM=Call a Star <noreply@callastar.com>
NEXT_PUBLIC_APP_URL=https://callastar.com
```

Webhook Security

The Stripe webhook that triggers payment confirmation emails is secured with:
- Stripe webhook signature verification
- Idempotency checks (no duplicate processing)

🛠️ Troubleshooting

Problem: Emails not being sent

Possible causes:
1. `RESEND_API_KEY` not configured
- **Solution**: Set the environment variable

1. Invalid recipient email
   - **Solution**: Check the database for valid email addresses

2. Resend API is down
   - **Solution**: Check Resend Status Page (https://status.resend.com/)

Problem: Emails sent but not logged

Possible causes:
1. Missing context parameters (`bookingId` , `userId` , `emailType`)
- **Solution**: Update the sendEmail call to include all context

1. Database connection issue
   - **Solution**: Check Prisma connection

Problem: Duplicate emails being sent

Possible causes:
1. Webhook idempotency not working
- **Solution**: Check `TransactionLog` for duplicate `stripeEventId`

1. Cron job running too frequently
   - **Solution**: Check `reminderSent` flag on bookings

📝 Adding New Email Types

To add a new email type:

1. **Create the template function** in `lib/email.ts`:

```
export function generateNewEmailType({...}) {
  return `<!DOCTYPE html>...`;
}
```

1. **Send the email with logging**:

```
const emailResult = await sendEmail({
  to: recipient.email,
  subject: 'Your Subject',
  html: generateNewEmailType({...}),
  bookingId: booking.id,
  userId: user.id,
  emailType: 'new_email_type',
});
```

1. **Document it** in this file under “Email Types”

2. **Test thoroughly** and verify logs are created

📞 Support

For issues related to the email system:
1. Check console logs
2. Check database `Log` table
3. Check Resend dashboard
4. Contact the development team

Last Updated: December 31, 2025
Version: 1.0.0
Maintainer: Call a Star Development Team