drop-email-system-spec

Drop Transactional Email System — Implementation Spec

Project: Drop (Fintech Payment App) **Task:** MC #1188 **Date:** 2026-02-17 **Author:** John (AI Director)

---

1. Executive Summary

Drop currently has placeholder email templates but no email service layer. This spec defines a production-ready transactional email system covering:

- Email service abstraction (provider-agnostic)
- Token-based workflows (email verification, password reset)
- Transactional emails (welcome, transaction receipt, login alerts)
- Database schema for tokens and logs
- API endpoints for verification flows
- Integration points with existing auth and transaction systems

MVP Recommendation: Resend — Next.js native, simplest setup (5 min vs 30-60 min for SendGrid), generous free tier (3,000 emails/month), React Email integration, automatic DKIM/SPF/DMARC.

---

2. Provider Comparison

Resend (RECOMMENDED for MVP)

Pros:

- Next.js/React native integration (React Email built-in)
- Automatic DKIM/SPF/DMARC setup (5 min vs 30-60 min for SendGrid)

- Free tier: 3,000 emails/month (100/day limit)
- Pro: $20/month for 50,000 emails (no daily limits)
- Modern developer experience
- Simple API (3 lines to send email)

Cons:

- Newer service (launched 2023, less mature than SendGrid)
- Lower volume ceiling (enterprise-scale needs SendGrid)
- Fewer features (no email validation, inbound parsing, visual template editor)

Best for: Drop MVP — we need transactional emails (not marketing campaigns), React templates, fast setup.

SendGrid

Pros:

- Mature platform (better deliverability reputation)
- More features: email validation, inbound parsing, visual templates, marketing campaigns
- Higher volume handling (enterprise-scale)
- Free trial: 100 emails/day for 60 days
- Essentials: $19.95/month for 50,000-100,000 emails

Cons:

- Complex setup (30-60 min for DNS records)
- Separate pricing for marketing campaigns
- Dedicated IPs cost extra
- Heavier integration (more API complexity)

Best for: Post-MVP if we need email validation, marketing campaigns, or >100k emails/month.

AWS SES

Pros:

- Cheapest at scale ($0.10 per 1,000 emails)
- No monthly minimums
- Built into AWS ecosystem (if we use AWS)

Cons:

- Requires AWS account and IAM setup
- Manual DNS configuration

- No template management
- No built-in analytics

Best for: High-volume, low-cost scenario (10M+ emails/year), already on AWS.

MVP Decision: Start with Resend. Migrate to SendGrid if we need email validation or marketing campaigns. All email logic abstracted via `src/lib/email.ts` — provider swap is 10 lines of code.

Sources:

- Resend vs SendGrid (2026) - Developer Email API Comparison | Sequenzy
- Resend vs Sendgrid Comparison (2026)
- Resend vs SendGrid in 2026: Email APIs Compared | DevPick

---

3. Architecture

3.1 Email Service Layer (`src/lib/email.ts`)

Provider-agnostic email abstraction. All email sending goes through this file.

Current state: Skeleton exists at `src/lib/services/email.ts` with demo-mode logging only.
Action: Replace with production implementation.

Module Interface:

```
// src/lib/email.ts

export interface EmailParams {
  to: string;
  subject: string;
  htmlBody: string;
  textBody?: string;
}

export interface EmailResult {
  success: boolean;
  messageId?: string;
  error?: string;
}
```

```
// Core send function
export async function sendEmail(params: EmailParams): Promise<EmailResult>;

// Template-based helpers
export async function sendWelcomeEmail(userId: string): Promise<EmailResult>;
export async function sendVerificationEmail(email: string, token: string):
Promise<EmailResult>;
export async function sendPasswordResetEmail(email: string, token: string):
Promise<EmailResult>;
export async function sendTransactionReceipt(txId: string): Promise<EmailResult>;
export async function sendTransferReceivedEmail(userId: string, txId: string):
Promise<EmailResult>;
export async function sendLoginAlertEmail(userId: string, ip: string, device: string):
Promise<EmailResult>;
```

Env Vars:

```
# Provider selection
EMAIL_PROVIDER=resend       # resend | sendgrid | smtp
EMAIL_FROM="Drop <no-reply@getdrop.no>"

# Resend (recommended)
RESEND_API_KEY=re_xxxxx

# SendGrid (alternative)
SENDGRID_API_KEY=SG.xxxxx

# SMTP (fallback)
SMTP_HOST=smtp.sendgrid.net
SMTP_PORT=587
SMTP_USER=apikey
SMTP_PASS=SG.xxxxx
```

Provider Implementations:

Resend (Primary)

```
// Resend SDK
import { Resend } from 'resend';
```

```
const resend = new Resend(process.env.RESEND_API_KEY);

async function sendViaResend(params: EmailParams): Promise<EmailResult> {
  const { data, error } = await resend.emails.send({
    from: process.env.EMAIL_FROM!,
    to: params.to,
    subject: params.subject,
    html: params.htmlBody,
    text: params.textBody,
  });

  if (error) {
    return { success: false, error: error.message };
  }

  return { success: true, messageId: data?.id };
}
```

SendGrid (Alternative)

```
// SendGrid SDK
import sgMail from '@sendgrid/mail';

sgMail.setApiKey(process.env.SENDGRID_API_KEY!);

async function sendViaSendGrid(params: EmailParams): Promise<EmailResult> {
  const msg = {
    to: params.to,
    from: process.env.EMAIL_FROM!,
    subject: params.subject,
    html: params.htmlBody,
    text: params.textBody,
  };

  try {
    const [response] = await sgMail.send(msg);
    return { success: true, messageId: response.headers['x-message-id'] };
  } catch (error) {
    return { success: false, error: (error as Error).message };
  }
```

```
}
```

SMTP (Fallback)

```
// Nodemailer
import nodemailer from 'nodemailer';

const transporter = nodemailer.createTransport({
  host: process.env.SMTP_HOST,
  port: Number(process.env.SMTP_PORT),
  secure: false, // TLS
  auth: {
    user: process.env.SMTP_USER,
    pass: process.env.SMTP_PASS,
  },
});

async function sendViaSMTP(params: EmailParams): Promise<EmailResult> {
  const info = await transporter.sendMail({
    from: process.env.EMAIL_FROM,
    to: params.to,
    subject: params.subject,
    html: params.htmlBody,
    text: params.textBody,
  });

  return { success: true, messageId: info.messageId };
}
```

Rate Limiting:

- Max 10 emails per user per hour (prevent abuse)
- Use existing `rate_limits` table with key `email:{userId}:{hour}`
- Helper: `async function checkEmailRateLimit(userId: string): Promise<boolean>`

Retry Logic (Fire-and-Forget for MVP):

- No retry for MVP (email service handles retries internally)
- Log failures to `email_log` table for manual review
- Post-MVP: Add job queue (BullMQ/Agenda) for guaranteed delivery

Template Loading:

```
// Load from src/email-templates/*.html
import fs from 'fs';
import path from 'path';

function loadTemplate(name: string): string {
  const templatePath = path.join(process.cwd(), 'src/email-templates', `${name}.html`);
  return fs.readFileSync(templatePath, 'utf-8');
}

// Replace {{placeholders}} with values
function renderTemplate(template: string, data: Record<string, string>): string {
  return template.replace(/\{\{(\w+)\}\}/g, (_, key) => data[key] || '');
}
```

3.2 Email Templates

Existing templates in `src/email-templates/`:

- `welcome.html` — Welcome email (placeholder: `{{verifyUrl}}`)
- `transaction-receipt.html` — Transaction confirmation (placeholders: `{{transactionDate}}`, `{{amount}}`, `{{fromName}}`, `{{toName}}`, etc.)
- `password-reset.html` — Password reset (placeholders: `{{resetUrl}}`, `{{userEmail}}`)

Templates to CREATE:

1. `email-verification.html`

Purpose: Verify email address (sent after registration) **Placeholders:**

- `{{firstName}}` — User's first name
- `{{verifyUrl}}` — Verification link with token (1-hour expiry)
- `{{otpCode}}` — 6-digit OTP code (backup method if link doesn't work)

Content:

```
<h1>Verifiser e-posten din</h1>
<p>Hei {{firstName}},</p>
<p>Klikk på lenken under for å verifisere e-postadressen din:</p>
<a href="{{verifyUrl}}" style="...">Verifiser e-post</a>
<p>Alternativt, skriv inn denne koden: <strong>{{otpCode}}</strong></p>
<p>Lenken utløper om 1 time.</p>
```

2. `transfer-received.html`

Purpose: Notify user they received money **Placeholders:**

- `{{firstName}}` — Recipient first name
- `{{amount}}` — Amount received (e.g., "kr 2 500,00")
- `{{currency}}` — Currency code (e.g., "NOK")
- `{{senderName}}` — Sender's name
- `{{transactionDate}}` — Timestamp
- `{{transactionId}}` — Transaction ID

Content:

```
<h1>Du mottok penger</h1>
<p>Hei {{firstName}},</p>
<p>Du har mottatt <strong>{{amount}} {{currency}}</strong> fra {{senderName}}.</p>
<p>Dato: {{transactionDate}}</p>
<p>Transaksjons-ID: {{transactionId}}</p>
<a href="https://getdrop.no/dashboard" style="...">Åpne Drop</a>
```

3. `login-alert.html`

Purpose: Security alert for new device/location login **Placeholders:**

- `{{firstName}}` — User's first name
- `{{device}}` — Device/browser (e.g., "Chrome on macOS")
- `{{location}}` — IP or location (e.g., "Oslo, Norway")
- `{{timestamp}}` — Login time
- `{{securityUrl}}` — Link to account security settings

Content:

```
<h1>Ny pålogging oppdaget</h1>
<p>Hei {{firstName}},</p>
<p>Vi har oppdaget en pålogging fra en ny enhet:</p>
<ul>
  <li>Enhet: {{device}}</li>
  <li>Plassering: {{location}}</li>
  <li>Tidspunkt: {{timestamp}}</li>
</ul>
<p>Hvis dette ikke var deg, <a href="{{securityUrl}}">endre passordet ditt</a>
umiddelbart.</p>
```

4. `support-ticket-update.html` (Future)

Purpose: Notify user their support ticket has a response **Placeholders:**

- `{{firstName}}`, `{{ticketId}}`, `{{ticketSubject}}`, `{{updateUrl}}`

3.3 Database Schema

New Tables:

`email_verification_tokens`

```
-- SQLite
CREATE TABLE IF NOT EXISTS email_verification_tokens (
  id TEXT PRIMARY KEY,
  user_id TEXT NOT NULL REFERENCES users(id),
  token TEXT UNIQUE NOT NULL,  -- UUID v4
  otp_code TEXT NOT NULL,       -- 6-digit code (backup method)
  expires_at TEXT NOT NULL,     -- ISO timestamp, 1 hour from creation
  used_at TEXT,                 -- NULL until verified
  created_at TEXT DEFAULT (datetime('now'))
);

CREATE INDEX IF NOT EXISTS idx_email_verify_user ON email_verification_tokens(user_id);
CREATE INDEX IF NOT EXISTS idx_email_verify_token ON email_verification_tokens(token);

-- PostgreSQL
CREATE TABLE IF NOT EXISTS email_verification_tokens (
  id TEXT PRIMARY KEY,
  user_id TEXT NOT NULL REFERENCES users(id),
  token TEXT UNIQUE NOT NULL,
  otp_code TEXT NOT NULL,
  expires_at TEXT NOT NULL,
  used_at TEXT,
  created_at TEXT DEFAULT (CURRENT_TIMESTAMP)
);
```

`password_reset_tokens`

```
-- SQLite
CREATE TABLE IF NOT EXISTS password_reset_tokens (
  id TEXT PRIMARY KEY,
```

```
  user_id TEXT NOT NULL REFERENCES users(id),
  token TEXT UNIQUE NOT NULL,  -- UUID v4
  expires_at TEXT NOT NULL,     -- ISO timestamp, 1 hour from creation
  used_at TEXT,                 -- NULL until reset
  created_at TEXT DEFAULT (datetime('now'))
);

CREATE INDEX IF NOT EXISTS idx_pwd_reset_user ON password_reset_tokens(user_id);
CREATE INDEX IF NOT EXISTS idx_pwd_reset_token ON password_reset_tokens(token);

-- PostgreSQL
CREATE TABLE IF NOT EXISTS password_reset_tokens (
  id TEXT PRIMARY KEY,
  user_id TEXT NOT NULL REFERENCES users(id),
  token TEXT UNIQUE NOT NULL,
  expires_at TEXT NOT NULL,
  used_at TEXT,
  created_at TEXT DEFAULT (CURRENT_TIMESTAMP)
);
```

email_log

```
-- SQLite
CREATE TABLE IF NOT EXISTS email_log (
  id TEXT PRIMARY KEY,
  user_id TEXT REFERENCES users(id),  -- NULL for non-user emails
  template TEXT NOT NULL,             -- Template name (e.g., "welcome", "password-reset")
  recipient TEXT NOT NULL,            -- Email address
  subject TEXT NOT NULL,
  status TEXT NOT NULL CHECK(status IN ('sent','failed')),
  message_id TEXT,                    -- Provider message ID
  error TEXT,                         -- Error message if failed
  sent_at TEXT DEFAULT (datetime('now'))
);

CREATE INDEX IF NOT EXISTS idx_email_log_user ON email_log(user_id);
CREATE INDEX IF NOT EXISTS idx_email_log_status ON email_log(status);
CREATE INDEX IF NOT EXISTS idx_email_log_sent_at ON email_log(sent_at);

-- PostgreSQL
```

```
CREATE TABLE IF NOT EXISTS email_log (
  id TEXT PRIMARY KEY,
  user_id TEXT REFERENCES users(id),
  template TEXT NOT NULL,
  recipient TEXT NOT NULL,
  subject TEXT NOT NULL,
  status TEXT NOT NULL CHECK(status IN ('sent','failed')),
  message_id TEXT,
  error TEXT,
  sent_at TEXT DEFAULT (CURRENT_TIMESTAMP)
);
```

Schema Migration:

- Add to `src/lib/db.ts` in `SQLITE_SCHEMA` and `PG_SCHEMA` constants
- Tables auto-create on `initDb()` (existing pattern)

3.4 API Endpoints

POST `/api/auth/verify-email`

Purpose: Verify email with token or OTP code **Request:**

```
{
  "token": "uuid-v4-token",  // From email link
  "code": "123456"           // Optional: OTP code (if user can't click link)
}
```

Response (200):

```
{
  "data": {
    "verified": true,
    "userId": "usr_xxx"
  }
}
```

Errors:

- 400: Missing token/code
- 404: Token not found

- 410: Token expired or already used

Logic:

1. Look up token in `email_verification_tokens`
2. Check expiry (`expires_at < now()` → error 410)
3. Check `used_at IS NOT NULL` → error 410
4. If `code` provided, validate `otp_code` matches
5. Update `used_at = now()`
6. Mark user as verified (add `email_verified` column to `users` table)
7. Log to audit log

File: `src/app/api/auth/verify-email/route.ts`

POST `/api/auth/forgot-password`

Purpose: Request password reset (sends email with token) **Request:**

```
{
  "email": "user@example.com"
}
```

Response (200):

```
{
  "data": {
    "message": "If the email exists, a reset link has been sent."
  }
}
```

Logic:

1. Look up user by email
2. If user not found → return 200 anyway (security: don't leak account existence)
3. Generate UUID token, 1-hour expiry
4. Insert into `password_reset_tokens`
5. Send `password-reset.html` email with `{{resetUrl}}` = `/reset-password?token=xxx`
6. Log to `email_log`

File: `src/app/api/auth/forgot-password/route.ts`

POST `/api/auth/reset-password`

Purpose: Reset password with token **Request:**

```
{
  "token": "uuid-v4-token",
  "newPassword": "NewP@ssw0rd123"
}
```

Response (200):

```
{
  "data": {
    "message": "Password reset successful. You can now log in."
  }
}
```

Errors:

- 400: Invalid password (reuse validation from `/api/auth/register`)
- 404: Token not found
- 410: Token expired or already used

Logic:

1. Look up token in `password_reset_tokens`
2. Check expiry and usage (same as verify-email)
3. Validate new password (8+ chars, 1 uppercase, 1 lowercase, 1 digit, 1 special)
4. Hash new password (`hashPassword()`)
5. Update `users.password_hash`
6. Mark token `used_at = now()`
7. Revoke all user sessions (security: force re-login)
8. Log to audit log

File: `src/app/api/auth/reset-password/route.ts`

POST `/api/auth/resend-verification`

Purpose: Resend verification email (if user didn't receive it) **Request:**

```
{
  "email": "user@example.com"
}
```

Response (200):

```
{
  "data": {
    "message": "If the email exists, a verification email has been sent."
  }
}
```

Logic:

1. Look up user by email
2. If not found → return 200 anyway
3. If already verified → return 200 (idempotent)
4. Invalidate old tokens (`UPDATE email_verification_tokens SET used_at = now() WHERE user_id = ? AND used_at IS NULL`)
5. Generate new token and OTP
6. Send verification email
7. Rate limit: max 3 resends per hour

File: `src/app/api/auth/resend-verification/route.ts`

3.5 Integration Points

A. Register Flow (MODIFY: `src/app/api/auth/register/route.ts`)

Current state: Lines 107-133 generate OTP for SMS (not implemented). **Action:** Add email verification.

Additions:

```
// After user insert (line 92)
import { sendVerificationEmail } from '@/lib/email';
import crypto from 'crypto';

// Generate email verification token
const verifyTokenId = randomId('evt');
const verifyToken = crypto.randomUUID();
const otpCode = String(crypto.randomInt(100000, 1000000)); // 6 digits
const expiresAt = new Date(Date.now() + 60 * 60 * 1000).toISOString(); // 1 hour

await run(
  `INSERT INTO email_verification_tokens (id, user_id, token, otp_code, expires_at)
```

```
   VALUES (?, ?, ?, ?, ?)`,
  [verifyTokenId, id, verifyToken, otpCode, expiresAt]
);

// Send verification email
await sendVerificationEmail(email!, verifyToken);
```

Note: Keep OTP SMS code (lines 109-133) as-is for phone verification. Email verification is separate.

B. Transaction Complete (MODIFY transaction routes)

Files to modify:

- `src/app/api/transactions/remittance/route.ts` (remittance)
- `src/app/api/transactions/qr-payment/route.ts` (QR payment)

After transaction status = 'completed':

```
import { sendTransactionReceipt, sendTransferReceivedEmail } from '@/lib/email';

// Send receipt to sender
await sendTransactionReceipt(txId);

// If remittance, notify recipient (if they have a Drop account)
if (type === 'remittance' && recipientUserId) {
  await sendTransferReceivedEmail(recipientUserId, txId);
}
```

Note: Recipient email only if recipient has a Drop account. Otherwise, recipient gets money via bank transfer (no Drop account = no email notification from Drop).

C. Login Alert (MODIFY: `src/app/api/auth/login/route.ts`)

Current state: Sets auth cookie, no device tracking. **Action:** Add login alert for new devices.

Device fingerprint detection:

```
import { getClientIp } from '@/lib/middleware';
import { sendLoginAlertEmail } from '@/lib/email';
import crypto from 'crypto';
```

```
const ip = getClientIp(request);
const userAgent = request.headers.get('user-agent') || 'Unknown';

// Generate device fingerprint (hash of IP + User-Agent)
const deviceFingerprint = crypto.createHash('sha256')
  .update(`${ip}:${userAgent}`)
  .digest('hex');

// Check if device is new
const existingDevice = await getOne<{ id: string }>(
  "SELECT id FROM sessions WHERE user_id = ? AND device_fingerprint = ?",
  [userId, deviceFingerprint]
);

if (!existingDevice) {
  // New device → send alert
  await sendLoginAlertEmail(userId, ip, userAgent);
}

// Add device_fingerprint to session insert
```

Schema change:

```
-- Add to sessions table
ALTER TABLE sessions ADD COLUMN device_fingerprint TEXT;
CREATE INDEX IF NOT EXISTS idx_sessions_device ON sessions(device_fingerprint);
```

D. Support Ticket Response (FUTURE)

Not in MVP. When support ticket system is built:

- `POST /api/support/tickets/:id/respond` → sends `support-ticket-update.html`

---

4. Dependencies

Add to `package.json`:

```
{
  "dependencies": {
```

```
    "resend": "^4.0.0",           // Resend SDK
    "@sendgrid/mail": "^8.1.0",   // SendGrid SDK (optional, for provider swap)
    "nodemailer": "^6.9.0"        // SMTP fallback
  },
  "devDependencies": {
    "@types/nodemailer": "^6.4.14"
  }
}
```

Install:

```
npm install resend @sendgrid/mail nodemailer
npm install -D @types/nodemailer
```

---

5. Env Vars

Add to `.env.example`:

```
# --- Email Service ---
# Provider: resend (recommended) | sendgrid | smtp
EMAIL_PROVIDER=resend
EMAIL_FROM="Drop <no-reply@getdrop.no>"

# Resend API key (get from resend.com)
RESEND_API_KEY=re_xxxxx

# SendGrid API key (alternative provider)
# SENDGRID_API_KEY=SG.xxxxx

# SMTP fallback
# SMTP_HOST=smtp.sendgrid.net
# SMTP_PORT=587
# SMTP_USER=apikey
# SMTP_PASS=SG.xxxxx
```

Production setup (Resend):

1. Sign up at resend.com
2. Add domain: `getdrop.no`

3. Add DNS records (DKIM, SPF, DMARC) — Resend provides exact records
4. Generate API key
5. Set `RESEND_API_KEY` in production env

---

6. File List

Files to CREATE:

```
src/lib/email.ts                                    # Email service layer
src/email-templates/email-verification.html         # Email verification template
src/email-templates/transfer-received.html          # Transfer received template
src/email-templates/login-alert.html                # Login alert template
src/app/api/auth/verify-email/route.ts              # Email verification endpoint
src/app/api/auth/forgot-password/route.ts           # Password reset request endpoint
src/app/api/auth/reset-password/route.ts            # Password reset endpoint
src/app/api/auth/resend-verification/route.ts       # Resend verification endpoint
```

Files to MODIFY:

```
src/lib/db.ts                                       # Add email_verification_tokens,
password_reset_tokens, email_log tables
src/app/api/auth/register/route.ts                  # Add email verification send
src/app/api/auth/login/route.ts                     # Add login alert for new devices
src/app/api/transactions/remittance/route.ts        # Add transaction receipt email
src/app/api/transactions/qr-payment/route.ts        # Add transaction receipt email
src/lib/services/email.ts                           # DELETE (replaced by src/lib/email.ts)
.env.example                                        # Add EMAIL_* env vars
package.json                                        # Add resend, @sendgrid/mail, nodemailer
deps
```

Total: 8 new files, 7 modified files.

---

7. Acceptance Criteria

Email Service Layer:

- ☐ `sendEmail()` sends via Resend in production

- [ ] `sendEmail()` logs to console in demo mode
- [ ] `sendEmail()` falls back to SMTP if Resend fails
- [ ] All template helpers (`sendWelcomeEmail()`, etc.) render templates correctly
- [ ] Rate limiting blocks >10 emails/user/hour

Database:

- [ ] All tables created on `initDb()`
- [ ] Tokens auto-expire after 1 hour
- [ ] `email_log` table logs all sent/failed emails

API Endpoints:

- [ ] `POST /api/auth/verify-email` verifies token and OTP
- [ ] `POST /api/auth/verify-email` rejects expired tokens (410)
- [ ] `POST /api/auth/forgot-password` sends reset email
- [ ] `POST /api/auth/reset-password` updates password
- [ ] `POST /api/auth/resend-verification` resends email
- [ ] All endpoints rate limited (10 req/min per IP)

Integration:

- [ ] Registration sends verification email
- [ ] Transaction completion sends receipt to sender
- [ ] Transfer completion sends notification to recipient (if Drop user)
- [ ] Login from new device sends security alert

Templates:

- [ ] All templates render with correct placeholders
- [ ] All templates display correctly in Gmail, Outlook, Apple Mail
- [ ] All templates mobile-responsive (375px width)

Provider Setup:

- [ ] Resend domain verified (`getdrop.no`)
- [ ] DKIM, SPF, DMARC records added to DNS
- [ ] API key set in production env
- [ ] Test email sent successfully

8. Implementation Plan

Phase 1: Email Service Layer (2h)

- Create `src/lib/email.ts` with Resend integration
- Add Resend dependency
- Set up env vars
- Test send via Resend dashboard

Phase 2: Database Schema (1h)

- Add 3 tables to `src/lib/db.ts`
- Test `initDb()` creates tables correctly

Phase 3: API Endpoints (3h)

- Create 4 API routes (verify, forgot, reset, resend)
- Add validation and rate limiting
- Test with Postman/curl

Phase 4: Templates (2h)

- Create 3 new templates (email-verification, transfer-received, login-alert)
- Test rendering with placeholders
- Test display in Gmail/Outlook

Phase 5: Integration (2h)

- Modify register route (send verification email)
- Modify transaction routes (send receipts)
- Modify login route (send alert)
- Test end-to-end flows

Phase 6: Production Setup (1h)

- Set up Resend account
- Verify domain (getdrop.no)
- Add DNS records
- Generate API key
- Deploy to staging

Total: 11 hours

9. Rollout Strategy

Staging:

- Deploy to staging environment
- Test all email flows with real email addresses
- Verify DNS records propagated
- Check spam score (mail-tester.com)

Production (Gradual):

- Enable email verification for NEW users only
- Monitor `email_log` for failures
- After 1 week: enable password reset
- After 2 weeks: enable transaction receipts
- After 3 weeks: enable login alerts

Monitoring:

- Daily check of `SELECT * FROM email_log WHERE status = 'failed'`
- Alert if >5% failure rate
- Weekly review of Resend dashboard (bounce rate, spam rate)

---

10. Success Metrics

Week 1:

- ☐ 0 email send failures
- ☐ <1% bounce rate
- ☐ <0.1% spam rate

Month 1:

- ☐ >90% email open rate (verification, password reset)
- ☐ >50% email open rate (transaction receipts)
- ☐ 0 support tickets about missing emails

Quarter 1:

- ☐ Email verification integrated into BankID flow
- ☐ Transaction receipts sent for 100% of transactions

- [ ] Login alerts sent for 100% of new devices

END OF SPEC

Revision #4
Created 2026-02-18 08:44:44 UTC by John
Updated 2026-05-31 20:02:13 UTC by John