External Services Integration

External Services Integration

> **Project:** {{PROJECT_NAME}} **Version:** {{VERSION}} **Date:** {{DATE}}
> **Author:** {{AUTHOR}} **Status:** Draft | In Review | Approved **Reviewers:** {{REVIEWERS}}

Document History

Version	Date	Author	Changes
0.1	{{DATE}}	{{AUTHOR}}	Initial draft

---

1. Integration Inventory

Service	Category	Criticality	SLA	Owner Team	Status
{{Stripe}}	Payments	Critical	99.99%	{{Backend}}	{{Active}}
{{SendGrid}}	Email delivery	High	99.95%	{{Backend}}	{{Active}}
{{Twilio}}	SMS	Medium	99.95%	{{Backend}}	{{Active}}
{{AWS S3}}	File storage	High	99.99%	{{Infrastructure}}	{{Active}}
{{Sentry}}	Error tracking	Low	—	{{DevOps}}	{{Active}}
{{Google Maps API}}	Geocoding	Medium	99.9%	{{Backend}}	{{Active}}
{{NAME}}	{{Category}}	{{Critical/High/Medium/Low}}	{{X.XX%}}	{{Team}}	{{Status}}

Criticality definitions:

- **Critical:** Service outage causes complete feature failure for end users
- **High:** Service outage degrades core functionality

- **Medium:** Service outage affects non-critical features
- **Low:** Monitoring/internal tools — no user impact

---

2. Per-Service Integration

2.1 Stripe (Payments)

Property	Value
Purpose	Payment processing, subscription billing
API docs	`https://stripe.com/docs/api`
Auth method	Secret key (Bearer token)
Credentials	Vault: `stripe/secret-key-{{env}}`
Webhook secret	Vault: `stripe/webhook-secret-{{env}}`
SDK	`stripe` npm package v14.x
API version	`2023-10-16` (pinned)

Key endpoints used:

Operation	Stripe API	Notes
Create customer	`POST /v1/customers`	On user registration
Create payment intent	`POST /v1/payment_intents`	Checkout flow
Confirm payment	`POST /v1/payment_intents/:id/confirm`	After client confirms
Create subscription	`POST /v1/subscriptions`	Subscription plans
Cancel subscription	`DELETE /v1/subscriptions/:id`	User-initiated cancel

Request example:

```
const paymentIntent = await stripe.paymentIntents.create({
  amount: 1000, // in cents
  currency: 'nok',
  customer: customer.stripeId,
  metadata: { orderId, userId },
  automatic_payment_methods: { enabled: true },
});
```

Error handling:

```
try {
  await stripe.paymentIntents.create(params);
} catch (err) {
  if (err instanceof Stripe.errors.StripeCardError) {
    throw new PaymentDeclinedException(err.message);
  }
  if (err instanceof Stripe.errors.StripeRateLimitError) {
    throw new ServiceTemporarilyUnavailableException('Payment service rate limited');
  }
  // Log unexpected errors to Sentry
  this.sentry.captureException(err);
  throw new PaymentServiceException('Unexpected payment error');
}
```

Retry policy: Stripe SDK handles retries on network errors automatically. Business-level failures (card declined) are NOT retried.

Circuit breaker: `{{Yes — breaker trips after 5 consecutive failures, opens for 30s}}`

Fallback: No fallback for payments — fail clearly with user-facing error message.

Webhooks consumed:

Event	Handler	Action
`payment_intent.succeeded`	`PaymentSucceededHandler`	Mark order paid
`payment_intent.payment_failed`	`PaymentFailedHandler`	Notify user, release hold
`customer.subscription.deleted`	`SubscriptionCancelledHandler`	Downgrade user

Rate limits: 100 read requests/s, 100 write requests/s per secret key. **Cost:** Per transaction (see Finance: Stripe billing dashboard).

2.2 SendGrid (Email)

Property	Value
Purpose	Transactional email delivery
API docs	`https://docs.sendgrid.com/api-reference`
Auth method	API key (Authorization: Bearer)

Property	Value
Credentials	Vault: `sendgrid/api-key-{{env}}`
SDK	`@sendgrid/mail` npm package v8.x
From email	`{{noreply@domain.com}}` (verified sender)

Key operations:

Operation	Template	Trigger
Welcome email	`d-XXXX`	User registration
Password reset	`d-XXXX`	Forgot password flow
Order confirmation	`d-XXXX`	Order placed
Invoice	`d-XXXX`	Invoice generated

Request example:

```
await sgMail.send({
  to: user.email,
  from: { email: 'noreply@domain.com', name: '{{APP_NAME}}' },
  templateId: 'd-XXXXXXXXXXXXXXXXXXXXXXXX',
  dynamicTemplateData: {
    firstName: user.name.split(' ')[0],
    orderNumber: order.number,
    orderTotal: formatCurrency(order.total),
  },
});
```

Error handling:

```
try {
  await sgMail.send(message);
} catch (err) {
  if (err.code === 429) {
    // Queue for retry
    await this.emailQueue.add('retry_email', message, { delay: 60000 });
  } else {
    this.logger.error('SendGrid error', { code: err.code, message: err.message });
    // Don't throw — email failure is non-critical for most flows
  }
}
```

Retry policy: 3 retries via BullMQ queue with 60s, 300s, 900s backoff. **Fallback:** `{{Postmark as backup SMTP | Log and alert team — no fallback}}` **Rate limits:** 100 emails/s on Pro plan.

2.3 AWS S3 (File Storage)

Property	Value
Purpose	User file uploads, generated reports, media
Auth method	IAM Role (EC2/ECS) or AWS Access Key
Credentials	IAM role (preferred) / Vault: `aws/s3-access-key-{{env}}`
SDK	`@aws-sdk/client-s3` v3.x
Buckets	See table below

Bucket configuration:

Bucket	Access	Lifecycle	Purpose
`{{company}}-uploads-{{env}}`	Private	90 day expiry for tmp	User uploads
`{{company}}-exports-{{env}}`	Private	7 day expiry	Generated exports/reports
`{{company}}-public-{{env}}`	Public (CDN)	None	Marketing assets, public images

Pre-signed URL pattern:

```
const command = new PutObjectCommand({
  Bucket: process.env.S3_UPLOADS_BUCKET,
  Key: `${userId}/${ulid()}.${extension}`,
  ContentType: mimeType,
  ContentLength: fileSize,
});

const presignedUrl = await getSignedUrl(s3Client, command, { expiresIn: 900 });
```

Retry policy: AWS SDK retries with exponential backoff by default (max 3 retries). **Circuit breaker:** Breaker trips after 10 consecutive failures. **Fallback:** `{{Cloudflare R2 as fallback storage | Abort upload with user error}}`

2.4 {{SERVICE_NAME}}

Property	Value
Purpose	`{{PURPOSE}}`
API docs	`{{URL}}`
Auth method	`{{API Key / OAuth2 / Basic Auth}}`
Credentials	Vault: `{{vault/path}}`
SDK	`{{package@version or "Direct HTTP"}}`

Key endpoints used:

Operation	Endpoint	Notes
`{{Operation}}`	`{{Method}} {{/path}}`	`{{Notes}}`

Request example:

```
// TODO: Add representative request example
```

Error handling:

```
// TODO: Define error handling strategy
```

Retry policy: `{{Exponential backoff: 1s, 2s, 4s, max 3 retries}}` **Circuit breaker:** `{{Yes/No — threshold: X failures in Y seconds}}` **Fallback / degradation:** `{{Define fallback behavior}}` **Rate limits:** `{{X requests per Y}}` **Cost:** `{{Pricing model reference}}` **Monitoring:** `{{Alert name and dashboard link}}`

---

3. SDK vs Direct API Call Decisions

Service	Approach	Rationale
Stripe	SDK	SDK handles retry logic, type safety, webhook verification
SendGrid	SDK	SDK simplifies template rendering, attachment handling
AWS S3	SDK v3	Modular SDK reduces bundle size; handles signing
`{{Service}}`	Direct HTTP	No official SDK, lightweight wrapper sufficient
`{{Service}}`	SDK	`{{Reason}}`

Wrapper pattern — all integrations are wrapped in a service class:

```
// services/stripe.service.ts — abstraction over Stripe SDK
@Injectable()
export class StripeService {
  // Exposes only operations the app actually needs
  // Hides Stripe-specific implementation details
  // Makes testing easier (injectable, mockable)
  async createPaymentIntent(amount: number, currency: string): Promise<PaymentIntent> { ... }
}
```

---

4. Mock / Stub Strategy for Development & Testing

Environment	Strategy
Unit tests	Jest manual mocks (`__mocks__/stripe.ts`)
Integration tests	Nock HTTP interceptors OR test-mode credentials
Local development	Test API keys (Stripe test mode, SendGrid sandbox)
E2E / staging	Live test-mode credentials — real API calls to sandbox
Production	Live production credentials

Mock setup example:

```
// __mocks__/@sendgrid/mail.ts
const sendMock = jest.fn().mockResolvedValue([{ statusCode: 202 }]);
export default { send: sendMock, setApiKey: jest.fn() };

// In tests
import sgMail from '@sendgrid/mail';
expect(sgMail.send).toHaveBeenCalledWith(expect.objectContaining({
  templateId: 'd-XXXXX',
}));
```

Test mode credentials location: `.env.test` (gitignored) — see onboarding guide.

---

5. Vendor Lock-In Assessment

Service	Lock-in Level	Switching Cost	Migration Complexity
Stripe	Medium	High (webhook events, customer IDs)	`{{2-4 weeks}}`
SendGrid	Low	Low (standard SMTP + template export)	`{{1-2 days}}`
AWS S3	Medium	Medium (URL changes, S3-compatible APIs)	`{{1 week}}`

Mitigation strategy: All integrations wrapped in service classes with defined interfaces. Swapping provider = rewrite service class, not application logic.

---

6. Migration Plan (Switching Providers)

Stripe ? Alternative Payment Provider

Trigger conditions: Pricing increase > 30%, reliability < 99.9%, compliance issues.

Migration steps:

1. Select alternative (Adyen, Braintree, etc.) and obtain test credentials
2. Implement new `PaymentService` adapter behind feature flag
3. Test in staging with full payment flow
4. Migrate new customers to new provider
5. Migrate existing subscription customers (requires customer consent in some jurisdictions)
6. Deprecate Stripe integration (keep webhooks active until all subscriptions migrated)

Data to migrate: Customer IDs (map old → new), subscription IDs.

Approval

Role	Name	Date	Signature
Author			
Backend Lead			
Finance / Legal (payment integrations)			
Security Reviewer			

Revision #7
Created 2026-02-23 12:05:29 UTC by John

Updated 2026-05-25 07:33:36 UTC by John