Bridgelet

Integration Guide

> **Document Version:** 0.1 (MVP)
> **Last Updated:** January 2025
> **Target Audience:** Backend Developers, DevOps Engineers
> **Prerequisites:** Getting Started Guide completed

1. Integration Overview

This guide covers production-ready integration of Bridgelet into your existing payment infrastructure. We'll cover API authentication, webhook handling, error management, and deployment strategies.

2. API Reference

2.1 Authentication

```
 // Initialize with API key
const bridgelet = new BridgeletSDK({
  apiKey: process.env.BRIDGELET_API_KEY,
  network: 'mainnet'
});

// Or use OAuth2 for user-specific operations
const bridgelet = new BridgeletSDK({
  clientId: process.env.CLIENT_ID,
  clientSecret: process.env.CLIENT_SECRET,
  network: 'mainnet'
});
```

2.2 Core Methods

Method	Parameters	Returns
`createEphemeralAccount()`	amount, asset, recipientEmail, expiresIn	Account object with claimUrl
`getAccountStatus()`	accountId	Status, balance, claimed state
`claimFunds()`	accountId, destinationKey, verificationCode	Transaction result
`recoverExpiredAccount()`	accountId	Recovery transaction
`listAccounts()`	filters, pagination	Array of accounts

3. Production Setup

3.1 Environment Configuration

```
 # Production .env
NODE_ENV=production
STELLAR_NETWORK=mainnet
STELLAR_HORIZON_URL=https://horizon.stellar.org

# Bridgelet Configuration
BRIDGELET_API_KEY=prod_key_xxx
BRIDGELET_API_URL=https://api.bridgelet.org

# Your organization wallet (use HSM in production!)
ORGANIZATION_SECRET_KEY=ENCRYPTED_KEY_FROM_HSM
ORGANIZATION_PUBLIC_KEY=GXXXXXXXXXXXXXXX

# Database
```

```
DATABASE_URL=postgresql://user:pass@host:5432/bridgelet

# Monitoring
SENTRY_DSN=https://xxx@sentry.io/xxx
LOG_LEVEL=info
```

3.2 Database Schema

```
 -- Store ephemeral account metadata
CREATE TABLE ephemeral_accounts (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  stellar_public_key VARCHAR(56) NOT NULL UNIQUE,
  amount DECIMAL(20, 7) NOT NULL,
  asset VARCHAR(12) NOT NULL,
  recipient_email VARCHAR(255),
  recipient_phone VARCHAR(20),
  claim_code VARCHAR(64) UNIQUE,
  status VARCHAR(20) NOT NULL, -- created, funded, claime
  expires_at TIMESTAMP NOT NULL,
  claimed_at TIMESTAMP,
  created_at TIMESTAMP DEFAULT NOW(),
  metadata JSONB,
  CONSTRAINT valid_status CHECK (status IN
    ('created', 'funded', 'claimed', 'expired', 'recovere
);

CREATE INDEX idx_status ON ephemeral_accounts(status);
CREATE INDEX idx_expires_at ON ephemeral_accounts(expires
CREATE INDEX idx_recipient_email ON ephemeral_accounts(re
```

4. Webhook Integration

4.1 Setup Webhooks

```
 // Register webhook endpoint
await bridgelet.registerWebhook({
  url: 'https://yourapp.com/webhooks/bridgelet',
  events: ['account.created', 'account.funded', 'account.
  secret: process.env.WEBHOOK_SECRET
});
```

4.2 Webhook Handler

```
 const express = require('express');
const crypto = require('crypto');

app.post('/webhooks/bridgelet', express.json(), async (re
  // Verify signature
  const signature = req.headers['x-bridgelet-signature'];
  const payload = JSON.stringify(req.body);
  const expectedSignature = crypto
    .createHmac('sha256', process.env.WEBHOOK_SECRET)
    .update(payload)
    .digest('hex');

  if (signature !== expectedSignature) {
    return res.status(401).json({ error: 'Invalid signatu
  }

  // Process event
  const { event, data } = req.body;

  switch (event) {
    case 'account.created':
      await handleAccountCreated(data);
      break;
    case 'account.claimed':
      await handleAccountClaimed(data);
      break;
    case 'account.expired':
```

```
      await handleAccountExpired(data);
      break;
  }

  res.json({ received: true });
});
```

5. Batch Operations

5.1 Bulk Account Creation

```
async function createBulkPayments(recipients) {
  const accounts = await bridgelet.createBulkAccounts(
    recipients.map(r => ({
      amount: r.amount,
      asset: 'USDC',
      recipientEmail: r.email,
      expiresIn: '7d',
      metadata: { employeeId: r.id }
    }))
  );

  // Send claim links via your email service
  for (const account of accounts) {
    await sendClaimEmail(account.recipientEmail, account.
  }

  return accounts;
}
```

5.2 CSV Import Example

```
const csv = require('csv-parser');
const fs = require('fs');

async function processCsvPayments(filePath) {
  const recipients = [];

  await new Promise((resolve) => {
    fs.createReadStream(filePath)
      .pipe(csv())
```

```
      .on('data', (row) => {
        recipients.push({
          email: row.email,
          amount: row.amount,
          id: row.employee_id
        });
      })
      .on('end', resolve);
  });

  return await createBulkPayments(recipients);
}
```

6. Error Handling

6.1 Common Errors

Error Code	Description	Resolution
`INSUFFICIENT_BALANCE`	Organization wallet lacks funds	Add more XLM to your wallet
`ACCOUNT_EXPIRED`	Claim attempted on expired account	Funds must be recovered first
`ALREADY_CLAIMED`	Account already claimed	Inform user funds were claimed
`INVALID_DESTINATION`	Recipient wallet address invalid	Validate wallet before claiming
`RATE_LIMIT_EXCEEDED`	Too many requests	Implement exponential backoff

6.2 Retry Logic

```
async function createAccountWithRetry(params, maxRetries
  for (let i = 0; i < maxRetries; i++) {
    try {
      return await bridgelet.createEphemeralAccount(param
    } catch (error) {
      if (error.code === 'RATE_LIMIT_EXCEEDED' && i < max
        const delay = Math.pow(2, i) * 1000; // Exponenti
        await new Promise(resolve => setTimeout(resolve,
        continue;
      }
      throw error;
```

```
    }
  }
}
```

7. Monitoring & Logging

7.1 Key Metrics to Track

- Account creation rate
- Claim success rate
- Average time to claim
- Expiration rate
- Failed claim attempts
- API error rates

7.2 Logging Example

```
 const winston = require('winston');

const logger = winston.createLogger({
  level: process.env.LOG_LEVEL || 'info',
  format: winston.format.json(),
  transports: [
    new winston.transports.File({ filename: 'error.log',
    new winston.transports.File({ filename: 'combined.log
  ]
});

// Log account creation
logger.info('Account created', {
  accountId: account.id,
  amount: account.amount,
  asset: account.asset,
  expiresAt: account.expiresAt
});

// Log errors with context
logger.error('Claim failed', {
  accountId,
  error: error.message,
```

```
    stack: error.stack
  });
```

8. Security Best Practices

8.1 API Key Management

- Store API keys in environment variables or secret management systems (AWS Secrets Manager, HashiCorp Vault)
- Rotate keys regularly (every 90 days recommended)
- Use different keys for development, staging, and production
- Never commit keys to version control
- Implement IP allowlisting for production API access

8.2 Webhook Security

```
// Always verify webhook signatures
function verifyWebhookSignature(payload, signature, secre
  const expectedSignature = crypto
    .createHmac('sha256', secret)
    .update(payload)
    .digest('hex');

  return crypto.timingSafeEqual(
    Buffer.from(signature),
    Buffer.from(expectedSignature)
  );
}
```

8.3 Claim Link Security

- Use HTTPS for all claim link delivery
- Include expiration timestamps in claim codes
- Implement rate limiting on claim attempts
- Log all claim attempts for audit trails
- Consider additional verification (email OTP) for high-value claims

9. Testing

9.1 Unit Tests

```
 const { BridgeletSDK } = require('@bridgelet/sdk');
const nock = require('nock');

describe('Bridgelet Integration', () => {
  let sdk;

  beforeEach(() => {
    sdk = new BridgeletSDK({
      apiKey: 'test_key',
      network: 'testnet'
    });
  });

  it('should create ephemeral account', async () => {
    nock('https://api.bridgelet.org')
      .post('/accounts')
      .reply(200, {
        id: 'acc_123',
        publicKey: 'GXXX...',
        claimUrl: 'https://claim.bridgelet.org/acc_123'
      });

    const account = await sdk.createEphemeralAccount({
      amount: '10',
      asset: 'XLM',
      recipientEmail: 'test@example.com',
      expiresIn: '7d'
    });

    expect(account.id).toBe('acc_123');
  });
});
```

10. Deployment Checklist

10.1 Pre-Production

- □ Complete security audit
- □ Set up monitoring and alerting
- □ Configure production environment variables
- □ Test failover and disaster recovery

- □ Document runbooks for common issues
- □ Set up automated backups
- □ Configure rate limits
- □ Test webhook delivery and retries

10.2 Production Launch

- □ Start with limited beta users
- □ Monitor error rates closely
- □ Have rollback plan ready
- □ Gradually increase traffic
- □ Collect user feedback
