Bridgelet

Architecture Overview

Document Version: 0.1 (MVP)
Last Updated: January 2025
Status: Early Development
Target Phase: Q1 2026

1. Overview

Bridgelet is an infrastructure SDK that enables organizations to send payments to recipients without requiring them to have crypto wallets or understand seed phrases. It creates secure, single-use Stellar accounts that automatically bridge recipients into permanent wallets when they claim funds.

1.1 Problem Statement

Mass payment systems (payroll, aid distribution, airdrops) face a critical barrier: recipients often lack crypto wallets or the technical knowledge to manage them. Traditional solutions require recipients to:

- Understand blockchain concepts
- Securely manage seed phrases
- Navigate complex wallet interfaces
- Perform setup before receiving funds

1.2 Solution Approach

Bridgelet creates temporary "bridge" accounts that hold funds until recipients are ready to claim them. The system automatically sweeps funds to recipients' permanent wallets upon claim, abstracting away the complexity of blockchain operations.

2. System Architecture

2.1 Core Components

🔧 Bridgelet Core (Smart Contracts)

Repository: `bridgelet-core`

Technology: Soroban (Rust)

Responsibilities:

- On-chain account restrictions and validation
- Automatic sweep logic implementation
- Fund expiration and recovery mechanisms
- Single-use account enforcement

⚙️ Bridgelet SDK (Backend)

Repository: `bridgelet-sdk`

Technology: NestJS (TypeScript)

Responsibilities:

- Account lifecycle management
- Claim authentication and verification
- Integration with payment platforms
- API endpoints for organizations

🎨 Bridgelet UI (Future)

Repository: `bridgelet-frontend`

Technology: Next.js (TypeScript)

Responsibilities:

- Reference implementation for claim flows
- User-friendly claiming interface
- Wallet connection management
- Status tracking and notifications

2.2 Data Flow

Payment Flow Diagram

```
1. Organization → Bridgelet SDK
   └ Create ephemeral account request

2. Bridgelet SDK → Stellar Network
   └ Deploy smart contract
   └ Create temporary account

3. Bridgelet SDK → Organization
   └ Return claim link/code

4. Organization → Recipient
   └ Send claim link (email/SMS/etc)

5. Recipient → Bridgelet UI
   └ Click claim link
   └ Connect permanent wallet

6. Bridgelet SDK → Smart Contract
   └ Verify claim
   └ Trigger auto-sweep

7. Smart Contract → Recipient Wallet
   └ Transfer funds
   └ Close ephemeral account
```

3. Technical Stack

Component	Technology	Purpose
Blockchain Layer	Stellar (Soroban)	Smart contract platform
Smart Contracts	Rust	On-chain logic and restrictions
Backend SDK	NestJS (TypeScript)	Account management API
Frontend (Future)	Next.js (TypeScript)	User interface for claiming
Database	PostgreSQL	Account metadata storage
Authentication	JWT + OAuth2	Secure claim verification

4. Key Design Decisions

4.1 Why Stellar/Soroban?

- **Low transaction costs:** Essential for micropayments and mass distribution
- **Fast finality:** 3-5 second confirmations for better UX
- **Smart contract support:** Soroban enables complex account restrictions
- **Built-in asset support:** Native multi-currency capabilities

4.2 Ephemeral vs. Custodial

Bridgelet uses **ephemeral accounts** (non-custodial) rather than a custodial wallet approach because:

- Reduced regulatory burden for organizations
- Recipients maintain true ownership
- No single point of failure
- Transparent on-chain verification

4.3 Single-Use Design

Each ephemeral account is designed for exactly one payment claim cycle to:

- Minimize attack surface
- Simplify state management
- Reduce on-chain storage costs
- Prevent account reuse vulnerabilities

5. Integration Points

5.1 For Payment Platforms

```
 // Example integration
import { BridgeletSDK } from '@bridgelet/sdk';

const sdk = new BridgeletSDK({
  network: 'testnet',
  apiKey: process.env.BRIDGELET_API_KEY
});

// Create ephemeral account for recipient
const account = await sdk.createEphemeralAccount({
  amount: '100',
  asset: 'USDC',
  recipientEmail: 'user@example.com',
  expiresIn: '7d'
});

// Send claim link to recipient
await sendEmail(account.claimUrl);
```

5.2 For Recipients

Recipients interact through a simple claiming interface:

1. Receive claim link via email/SMS
2. Click link to open claiming interface
3. Connect existing wallet OR create new wallet
4. Confirm claim
5. Funds automatically transferred to their wallet

6. Scalability Considerations

6.1 Current Limitations (MVP)

- Single asset per account
- One claim per ephemeral account

- Manual expiration handling
- Limited claim authentication methods

6.2 Future Enhancements

- Multi-asset support per account
- Batched account creation
- Advanced claim verification (KYC, biometrics)
- Automated expiration processing
- Cross-chain bridge support

7. Development Roadmap

Phase	Timeline	Deliverables
MVP (v0.1)	Q1 2026	Core contracts, SDK, basic claiming
Beta (v0.2)	Q2 2026	UI reference, enhanced security, testing
Production (v1.0)	Q3 2026	Mainnet deployment, audits, documentation