DSI Exodus 2.0 Widget: Technical Specification (Updated)

1. System Overview

1.1 Purpose

The DSI Exodus 2.0 Widget is a decentralized social organization tool that enables: - Formation of P2P trust networks - Management of mutual assistance processes - Organization of direct cooperation - Integration with AI for neutral analysis

1.2 Core Principles

- Trust-free architecture
- Decentralized operation
- Empathy-driven connections
- Minimal data storage
- Complete transparency

1.3 Architectural Concept

The Exodus 2.0 Widget adopts a thin-client architecture. The widget itself acts as a lightweight shell, responsible for displaying a minimal core UI and handling user interactions. The bulk of data storage and processing resides in the Google Cloud. Data is dynamically fetched from the cloud via API requests based on user queries and interactions, allowing for a simplified client-side implementation and improved scalability.

2. Functional Requirements

2.1 P2P Registry Formation

Key Functions:

- Automatic P2P connection creation
- Handshake chain tracking
- Connection verification
- Registry maintenance

Data Fetching:

- On widget load: Fetch basic user information (ID, connections count).
- On user interaction (e.g., clicking on "My Connections"): Fetch detailed connection information (IDs, timestamps).

User Stories:

``` As a new participant I want to join through invitation So that I'm connected to the trusted network

As a participant I want to see my connection paths So that I understand my network position ```

## 2.2 Mutual Assistance Organization

**Key Functions:**

- SOS request management with immediate crowdfunding
- Regular assistance tracking with periodic crowdfunding
- Obligation recording
- Fulfillment verification
- Dynamic progress display for all crowdfunding types

**Data Fetching:**

- On SOS request: Fetch the list of participants in the contributor's circle.
- On progress update: Fetch the latest status of contributions.

**User Stories:**

``` As a participant in need I want to create an SOS request So that network members can assist me automatically and dynamically

As a helper I want to commit to obligations So that I can support other participants ```

## 2.3 Cooperation System

**Key Functions:**

- Initiative creation with unique crowdfunding campaigns
- Crowdfunding management with unique campaign codes
- Need-capability matching
- Obligation clearing

**Data Fetching:**

- On initiative creation: Fetch existing similar initiatives.
- On need-capability matching: Fetch potential collaborators' profiles and history.

**User Stories:**

``` As an initiator I want to start a collaborative project So that others can join and contribute

As a participant I want to find matching needs/capabilities So that I can engage in direct cooperation ```

## 2.4 AI Integration

**Key Functions:**

- Data analysis
- Recommendation generation
- Pattern recognition
- Opportunity identification
- Selective campaign distribution and recipient filtering

  **Important Note:** The AI operates exclusively within the registry data provided by participants. For example, it facilitates finding collaborations by matching participants' needs and excess capabilities. All actual execution of collaborations (including any financial transactions or exchange of goods/services) occurs outside the widget, using external communication channels.

**AI Queries:**

- **Need/Capability Matching:** Send user's needs and capabilities (from Cooperation data) to Gemini AI API. Expect a list of potential matches with confidence scores.
- **Network Growth Analysis:** Request aggregated network data (user count over time) from the API and send it to Gemini AI for analysis and trend prediction.

**User Stories:**

```
As a participant I want AI-powered insights So that I can find optimal cooperation opportunities and manage targeted communications
```

# 3. Technical Architecture

## 3.1 Core Data Structures

```typescript interface Action { type: 'OBLIGATIONFULFILLED' | 'OBLIGATIONMISSED' | 'INITIATIVECREATED' | 'INITIATIVESUPPORTED'; timestamp: number; relatedId?: string; // Unique identifier of the related obligation/initiative amount?: number; // Amount associated with the action (e.g., help provided) description?: string; // Additional details about the action }

interface Connection { participantId: string; // Unique identifier of the connected participant timestamp: number; // Timestamp when the connection was established }

interface ReputationScore { score: number; // Reputation score lastUpdated: number; // Last update timestamp positiveActionsCount: number; // Count of positive actions negativeActionsCount: number; // Count of negative actions actionsLog: Action[]; // Log of all actions affecting reputation }

interface Participant { id: string; // Unique identifier (UUID) invitedBy: string; // ID of the inviter handshakePath: string[]; // Sequence of handshake IDs timestamp: number; // Timestamp of registration connections: Connection[]; // Fetched on demand reputation: ReputationScore; // Fetched on demand }

interface Obligation { id: string; // Unique identifier (UUID) type: 'SOS' | 'REGULAR'; amount: number; // Amount requested requesterId: string; // ID of the participant requesting assistance participantId: string; // ID of the participant fulfilling the obligation status: 'PENDING' | 'FULFILLED'; externalLink?: string; // Optional external discussion link timestamp: number; // Timestamp of creation }

interface Initiative { id: string; // Unique identifier (UUID) creatorId: string; // ID of the creator description: string; // Initiative description goal: number; // Fundraising goal currentAmount: number; // Current amount raised status: 'ACTIVE' | 'COMPLETED' | 'CANCELLED'; participants: string[]; // List of participant IDs externalLink?: string; // Optional external link for more information }

interface Cooperation { id: string; // Unique identifier (UUID) type: 'NEED' | 'OFFER'; participantId: string; // ID of the participant description: string; // Description of the need or offer tags: string[]; // Tags for categorization amount?: number; // Optional amount associated status: 'ACTIVE' | 'MATCHED' | 'COMPLETED'; }

interface Crowdfunding { id: string; // Unique identifier (UUID) type: 'SOS' | 'REGULAR' | 'INITIATIVE'; creatorId: string; // ID of the creator goal: number; // Fundraising goal currentAmount: number; // Current amount raised participants: string[]; // List of participant IDs uniqueCode: string; // Unique code for sharing the campaign status: 'ACTIVE' | 'COMPLETED' | 'CANCELLED'; externalLink?: string; // Optional external link for more information } ```

## 3.2 API Endpoints

```typescript // Registry Management POST /api/participants/register // POST: Register a new participant GET /api/participants/{id}/connections // GET: Get participant's connections GET /api/participants/{id}/handshakes // GET: Get participant's handshake path

// Assistance Management POST /api/assistance/sos // POST: Create an SOS request POST /api/assistance/regular // POST: Create a regular assistance request PUT /api/obligations/{id}/fulfill // PUT: Mark an obligation as
```

fulfilled GET /api/obligations/{id}/status // GET: Get the status of an obligation

// Cooperation Management POST /api/initiatives/create // POST: Create a new initiative POST /api/initiatives/{id}/join // POST: Join an existing initiative GET /api/cooperation/matches // GET: Find matching cooperation opportunities POST /api/cooperation/clear // POST: Clear a cooperation entry

// Crowdfunding Management GET /api/crowdfunding/{id}/status // GET: Get the status of a crowdfunding campaign POST /api/crowdfunding/{id}/share // POST: Generate a shareable link for a crowdfunding campaign ```

3.5 API Design

Base URL: `/api/v1`

Endpoints: - **GET /participants/{id}/connections**: - **Request:** None - **Response (JSON):** `json [ { "participantId": "user123", "timestamp": 1678886400 }, { "participantId": "user456", "timestamp": 1678972800 } ]` - **HTTP Status Codes:** 200 OK, 404 Not Found

- **POST /api/assistance/sos**:
  - **Request (JSON):** `json { "amount": 1000, "externalLink": "https://example.com/help" }`
  - **Response (JSON):** `json { "id": "sos123", "status": "ACTIVE" }`
  - **HTTP Status Codes:** 201 Created, 400 Bad Request ```

4.4 Data Handling

- Data fetched from APIs is dynamically displayed in the widget and is not permanently stored locally.
- Minimal caching may be implemented for frequently accessed data, such as user profiles, to minimize API requests and improve performance.
- All sensitive operations rely on secure API endpoints.