**Collaborative Editor Quick Win Features**

# Executive Summary

This specification details five priority features for the collaborative editor that can be implemented independently without external dependencies. These features provide the foundation for PromiseGrid integration while delivering immediate user value. The features are ordered by implementation priority and designed to be developed sequentially over 3-4 weeks.

# Priority Features Overview

## Foundation Features

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Priority** | **Effort** | **Purpose** |
| Document Identity & Fingerprinting | High | Low | Foundation for content addressing and version tracking |
| Database Schema Extensions | High | Low | Enable collaboration data storage and retrieval |
| Message Queue Management | High | Low | Critical offline functionality and message reliability |

## Extension Features

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Priority** | **Effort** | **Purpose** |
| Network Status Display | Medium | Low | Quick win improving user experience and transparency |
| Enhanced PromiseGrid Message Types | High | Medium | Core protocol extension enabling collaboration features |

# Feature 1: Document Identity & Fingerprinting

## Overview

Implement content-based document identity and fingerprinting system that generates unique identifiers for document content and tracks content evolution. This feature provides the foundation for content addressing, version tracking, and future PromiseBase integration.

## Technical Requirements

### Core Components

DocumentIdentity Class

class DocumentIdentity {

generateFingerprint(content: string): string

trackContentChanges(delta: YDelta): void

createDocumentCID(): string

verifyDocumentIntegrity(): boolean

getContentHistory(): ContentSnapshot[]

}

### Implementation Details

**•** Hash Algorithm: Use existing WASM hash functions (SHA-256) for content fingerprinting

**•** Content Normalization: Normalize whitespace and formatting before hashing to ensure consistent fingerprints

**•** Change Tracking: Monitor Yjs document changes and generate new fingerprints for significant content modifications

**•** CID Generation: Generate IPFS-compatible Content Identifiers for future PromiseBase integration

**•** Integrity Verification: Provide methods to verify document content matches its fingerprint

### Data Structures

interface ContentSnapshot {

fingerprint: string

timestamp: number

contentLength: number

changeSummary: string

cid?: string

}

### Integration Points

**•** Yjs Integration: Hook into Yjs document update events to trigger fingerprint generation

**•** WASM Functions: Extend existing WASM module with document fingerprinting functions

**•** IndexedDB Storage: Store fingerprint history using database schema extensions

**•** Message Queue: Generate fingerprint change messages for PromiseGrid distribution

# Feature 2: Database Schema Extensions

## Overview

Extend the existing IndexedDB schema to support collaboration-specific data storage including PromiseGrid messages, document metadata, user information, and peer relationships. This provides the data foundation for all collaboration features.

## Schema Extensions

### New Object Stores

const COLLABORATION\_STORES = {

promiseGridMessages: {

keyPath: 'id',

indexes: ['timestamp', 'document\_id', 'type', 'sender']

},

documentMetadata: {

keyPath: 'cid',

indexes: ['title', 'author', 'created', 'modified', 'fingerprint']

},

userProfiles: {

keyPath: 'id',

indexes: ['public\_key', 'reputation', 'last\_seen']

},

peerConnections: {

keyPath: 'id',

indexes: ['endpoint', 'last\_contact', 'trust\_level']

},

messageQueue: {

keyPath: 'id',

indexes: ['priority', 'created', 'retry\_count', 'status']

}

};

### Data Models

interface PromiseGridMessage {

id: string

timestamp: number

document\_id: string

type: string

sender: string

payload: Uint8Array

signature?: Uint8Array

}

### Database Manager

class CollaborationDB {

async storeMessage(message: PromiseGridMessage): Promise<void>

async getMessagesByDocument(docId: string): Promise<PromiseGridMessage[]>

async storeDocumentMetadata(metadata: DocumentMetadata): Promise<void>

async getUserProfile(userId: string): Promise<UserProfile | null>

async updatePeerConnection(peer: PeerConnection): Promise<void>

async queueMessage(message: QueuedMessage): Promise<void>

async getPendingMessages(): Promise<QueuedMessage[]>

}

**Feature 3: Message Queue Management**

## Overview

Implement a robust message queuing system that handles PromiseGrid message delivery, offline functionality, message prioritization, and network state transitions. This ensures reliable collaboration even during network interruptions.

## Core Components

### Message Queue Manager

class PromiseGridQueue {

async queueMessage(message: PromiseGridMessage, priority: number = 5): Promise<void>

async flushPendingMessages(): Promise<FlushResult>

handleNetworkStateChange(online: boolean): void

prioritizeMessages(): void

retryFailedMessages(): Promise<void>

getQueueStatus(): QueueStatus

}

### Message Prioritization

**•** Priority 1 (Critical): Document edit operations, user presence updates

**•** Priority 3 (High): Comments, annotations, permission changes

**•** Priority 5 (Normal): Document metadata updates, user status changes

**•** Priority 7 (Low): Analytics events, background synchronization

**•** Priority 10 (Lowest): Historical data, cleanup operations

### Network State Management

class NetworkManager {

detectNetworkState(): NetworkState

managePeerConnections(): void

handleReconnection(): Promise<void>

optimizeMessageDelivery(): void

onNetworkChange(callback: (online: boolean) => void): void

}

### Queue Data Structures

interface QueuedMessage {

id: string

message: PromiseGridMessage

priority: number

created: number

retry\_count: number

status: 'pending' | 'sending' | 'sent' | 'failed'

last\_attempt?: number

error?: string

}

# Feature 4: Network Status Display

## Overview

Create a comprehensive network status display system that provides real-time feedback about collaboration network health, peer connections, message queue status, and overall system state. This feature improves user experience and system transparency.

## UI Components

### Network Status Widget

class NetworkStatusUI {

updatePeerCount(count: number): void

showNetworkHealth(metrics: NetworkMetrics): void

displayMessageQueue(queueSize: number, pending: number): void

showConnectionStatus(status: ConnectionStatus): void

displayLatencyIndicator(latency: number): void

renderStatusIcon(state: NetworkState): HTMLElement

}

### Visual Indicators

**•** Connection Status Icons: Green (connected), Yellow (limited), Red (offline), Gray (disconnected)

**•** Peer Count Badge: Real-time display of connected collaborators

**•** Message Queue Indicator: Shows pending messages with priority visualization

**•** Latency Meter: Visual representation of network response times

**•** Sync Status: Indicates document synchronization state

### Status Data Models

interface NetworkMetrics {

peerCount: number

connectedPeers: number

averageLatency: number

messagesSent: number

messagesReceived: number

queuedMessages: number

lastSyncTime: number

}

# Feature 5: Enhanced PromiseGrid Message Types

## Overview

Design and implement enhanced CBOR message types that extend the existing PromiseGrid protocol to support comprehensive collaborative editing features including presence, comments, annotations, permissions, and document lifecycle management.

## Message Type Categories

### Presence Messages

function createPresenceMessage(

userId: string,

cursorPosition: number,

selection: {start: number, end: number} | null,

status: 'online' | 'typing' | 'idle' | 'away'

): Uint8Array

### Comment & Annotation Messages

function createCommentMessage(

docId: string,

position: number,

comment: string,

userId: string,

threadId?: string

): Uint8Array

### Permission Messages

function createPermissionMessage(

docId: string,

userId: string,

permissions: Permission[],

grantor: string

): Uint8Array

### Document Lifecycle Messages

function createDocumentMetadata(

title: string,

authors: string[],

created: number,

modified: number,

tags: string[]

): Uint8Array

### CBOR Schema Definitions

// Message envelope structure

interface MessageEnvelope {

version: number // Protocol version

type: string // Message type identifier

timestamp: number // Unix timestamp

sender: string // User ID

document\_id: string // Target document

payload: any // Type-specific payload

signature?: Uint8Array // Optional signature

}

### WASM Integration

**•** Extend existing WASM module with new message generation functions

**•** Maintain compatibility with existing 'grid' tag (0x67726964) for CBOR encoding

**•** Implement message validation and parsing functions

**•** Provide JavaScript bindings for all new message types

# Implementation Guidelines

## Development Sequence

### Week 1: Foundation Features

1. Document Identity & Fingerprinting - Implement core content hashing and fingerprint generation using existing WASM functions

2. Database Schema Extensions - Extend IndexedDB with new object stores and data models

3. Message Queue Management - Build offline message queuing system with priority handling

### Week 2-3: Extension Features

4. Network Status Display - Create UI components for real-time network status visualization

5. Enhanced PromiseGrid Message Types - Design and implement new CBOR message formats with WASM integration

## Integration Requirements

**•** Maintain backward compatibility with existing Yjs collaboration system

**•** Preserve existing IndexedDB data and migration paths

**•** Ensure graceful degradation when PromiseGrid features are unavailable

**•** Integrate seamlessly with existing WASM module and build system

**•** Maintain performance standards for real-time collaboration

## Success Criteria

**•** All features function independently without external dependencies

**•** Document fingerprinting generates consistent identifiers across sessions

**•** Message queue handles offline/online transitions gracefully

**•** Network status display provides accurate real-time information

**•** Enhanced message types integrate seamlessly with existing PromiseGrid protocol

**•** No performance degradation in existing collaboration features

# Conclusion

These five priority features establish the foundation for comprehensive PromiseGrid integration while providing immediate value to users. Each feature can be implemented independently, allowing for incremental development and testing. The implementation sequence ensures that foundational capabilities are established first, followed by user-facing improvements and protocol extensions.

By focusing on these specific features, the collaborative editor will gain essential distributed collaboration capabilities without requiring coordination with external development teams. This approach enables rapid progress while maintaining system stability and user experience quality.