CLEAR MIND v2.0

Replit Agent Optimized Product Requirements Document

Enterprise-Grade Dual-Interface Idea Management Platform

EXECUTIVE SUMMARY

Product Vision

Clear Mind v2.0 is a sophisticated dual-interface platform that bridges the gap between creative ideation and structured task execution through an innovative bidirectional synchronization architecture. Built specifically for Replit Agent development, this platform enables seamless transition between visual canvas and structured task management while maintaining data integrity through advanced conflict resolution.

Market Positioning

Primary Differentiator: Bidirectional Staging Architecture with Conscious Conflict Resolution

- Traditional tools: Linear workflow (idea → task)
- Clear Mind: Fluid bidirectional workflow with intelligent sync management

Target Market

- **Primary**: Senior project managers and creative directors managing complex multi-phase projects
- **Secondary**: Product teams requiring visual brainstorming and structured execution
- **Validated Use Cases**: Software development planning, content strategy, product roadmaps

Success Metrics (MVP v2.0)

- 30-day consecutive daily usage by internal team (5 users minimum)
- Zero data loss incidents during sync operations
- 95% user satisfaction with conflict resolution UX
- Average time from idea creation to task completion: <48 hours

---

REPLIT PLATFORM SPECIFICATIONS

Technology Stack (Replit Agent Compatible)

```
Frontend: React 18+ with TypeScript
Backend: Express.js with Node.js
```

Database: PostgreSQL (Replit integrated)
Authentication: Replit Auth (built-in)
Styling: Tailwind CSS
Deployment: Replit hosting (.replit.app domain)
Environment: FULLSTACK_JS template

Replit Agent Development Advantages

- **Instant Environment Setup**: Zero configuration required
- **AI-Driven Scaffolding**: Agent creates complete project structure
- **Integrated Database**: PostgreSQL automatically provisioned
- **Built-in Authentication**: Replit Auth handles user management
- **One-Click Deployment**: Immediate production hosting
- **Real-time Collaboration**: Built-in team development features

---

TECHNICAL ARCHITECTURE (Replit Optimized)

System Overview (Restaurant Process)

```
Frontend (Waiter/Menu) - React + TypeScript
├── Canvas Interface: Infinite zoom, drag-drop, visual grouping
├── TodoList Interface: Grid-based task management with modals
├── Navigation System: React Router with persistent state
└── Sync Indicators: Real-time visual feedback components

Backend (Kitchen) - Express.js + Node.js
├── Staging API: RESTful endpoints for change management
├── Sync Controller: State coordination between interfaces
├── Conflict Resolver: Automated detection with guided resolution
└── Export Service: Configurable list generation

Database (Storage) - PostgreSQL
├── Production Tables: Authoritative data with versioning
├── Staging Tables: Temporary change buffers
├── Conflict Logs: Complete audit trail
└── User Management: Replit Auth integration

Deployment (Service) - Replit Platform
├── Auto-scaling hosting on .replit.app
├── Environment variable management
├── SSL certificates and custom domains
└── Integrated monitoring and logging
```

DATABASE SCHEMA (PostgreSQL Optimized)

Production Tables

sql

```sql
-- Users table (integrates with Replit Auth)
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    replit_user_id VARCHAR(255) UNIQUE NOT NULL, -- Replit Auth ID
    username VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL,
    display_name VARCHAR(100),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Primary entity representing unified idea/task concept
CREATE TABLE ideas (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    title VARCHAR(255) NOT NULL,
    description TEXT,
    container_id UUID REFERENCES idea_containers(id) ON DELETE CASCADE,
    priority_level priority_enum DEFAULT 'medium',
    section_id UUID REFERENCES sections(id) ON DELETE SET NULL,
    order_position INTEGER NOT NULL DEFAULT 0,
    canvas_position_x INTEGER NULL,
    canvas_position_y INTEGER NULL,
    is_deleted BOOLEAN DEFAULT FALSE,
    created_from interface_enum NOT NULL,
    last_modified_from interface_enum NOT NULL,
    version_number INTEGER DEFAULT 1,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE
);

-- Container representing groups/todolists
CREATE TABLE idea_containers (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    color VARCHAR(7) NOT NULL, -- hex color
    sync_status sync_status_enum DEFAULT 'synced',
    canvas_position_x INTEGER NULL,
    canvas_position_y INTEGER NULL,
    requires_user_action BOOLEAN DEFAULT FALSE,
    last_canvas_update TIMESTAMP WITH TIME ZONE,
    last_todolist_update TIMESTAMP WITH TIME ZONE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE
);
```

```sql
-- Sections for todolist organization
CREATE TABLE sections (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    container_id UUID REFERENCES idea_containers(id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL,
    order_position INTEGER NOT NULL DEFAULT 0,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE
);

-- Custom enums
CREATE TYPE priority_enum AS ENUM ('low', 'medium', 'high', 'critical');
CREATE TYPE interface_enum AS ENUM ('canvas', 'todolist');
CREATE TYPE sync_status_enum AS ENUM ('synced', 'pending', 'conflict', 'resolving');
CREATE TYPE conflict_type_enum AS ENUM ('content_diff', 'delete_vs_modify', 'structural');
```

Staging Architecture

sql

```sql
-- Canvas staging for conflict-free modifications
CREATE TABLE ideas_staging_canvas (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    original_idea_id UUID REFERENCES ideas(id) ON DELETE CASCADE,
    staged_title VARCHAR(255),
    staged_description TEXT,
    staged_canvas_x INTEGER,
    staged_canvas_y INTEGER,
    staged_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    conflict_with_todolist BOOLEAN DEFAULT FALSE,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE
);

-- TodoList staging for conflict-free modifications
CREATE TABLE ideas_staging_todolist (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    original_idea_id UUID REFERENCES ideas(id) ON DELETE CASCADE,
    staged_title VARCHAR(255),
    staged_description TEXT,
    staged_priority priority_enum,
    staged_section_id UUID,
    staged_order_position INTEGER,
    staged_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    conflict_with_canvas BOOLEAN DEFAULT FALSE,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE
);

-- Comprehensive conflict tracking
CREATE TABLE conflict_resolution_log (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    idea_id UUID REFERENCES ideas(id) ON DELETE CASCADE,
    conflict_type conflict_type_enum NOT NULL,
    canvas_version JSONB,
    todolist_version JSONB,
    resolution_chosen interface_enum,
    resolved_by UUID REFERENCES users(id),
    resolved_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    metadata JSONB
);
```

REPLIT AGENT COMPONENT ARCHITECTURE

Frontend Components (React + TypeScript)

typescript

```tsx
// Main App Component with Routing
interface AppProps {}

const App: React.FC<AppProps> = () => {
  const { user, loading } = useReplitAuth();

  if (loading) return <LoadingSpinner />;
  if (!user) return <LoginScreen />;

  return (
    <BrowserRouter>
      <div className="min-h-screen bg-gray-50">
        <Header user={user} />
        <main className="container mx-auto px-4 py-8">
          <Routes>
            <Route path="/" element={<Dashboard />} />
            <Route path="/canvas" element={<CanvasView />} />
            <Route path="/todolist" element={<TodoListView />} />
          </Routes>
        </main>
      </div>
    </BrowserRouter>
  );
};

// Replit Auth Integration Hook
const useReplitAuth = () => {
  const [user, setUser] = useState<ReplitUser | null>(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    // Integrate with Replit Auth system
    const checkAuth = async () => {
      try {
        const replitUser = await window.replitAuth.getCurrentUser();
        if (replitUser) {
          // Sync with local user table
          const localUser = await syncUserWithDatabase(replitUser);
          setUser(localUser);
        }
      } catch (error) {
        console.error('Auth check failed:', error);
      } finally {
        setLoading(false);
      }
    };
```

```
    checkAuth();
  }, []);

  return { user, loading };
};

// Canvas Component with Drag & Drop
const CanvasView: React.FC = () => {
  const { containers } = useContainers();
  const { ideas, updateIdeaPosition } = useIdeas();
  const { syncState, resolveSyncConflict } = useSyncState();

  const handleDragEnd = async (ideaId: string, newPosition: Position) => {
    await updateIdeaPosition(ideaId, newPosition);
  };

  return (
    <div className="canvas-container h-screen flex">
      <Sidebar>
        <CreateIdeaButton />
        <ContainersList
          containers={containers}
          syncStates={syncState}
          onResolveSyncConflict={resolveSyncConflict}
        />
      </Sidebar>

      <InfiniteCanvas onDragEnd={handleDragEnd}>
        {ideas.map(idea => (
          <DraggableIdeaCard
            key={idea.id}
            idea={idea}
            position={idea.canvasPosition}
          />
        ))}
      </InfiniteCanvas>
    </div>
  );
};

// TodoList Grid Component
const TodoListView: React.FC = () => {
  const { containers } = useContainers();
  const [selectedContainer, setSelectedContainer] = useState<string | null>(null);

  return (
```

```
    <div className="todolist-view">
      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
        <CreateTodoListCard />
        {containers.map(container => (
          <TodoListCard
            key={container.id}
            container={container}
            onClick={() => setSelectedContainer(container.id)}
          />
        ))}
      </div>

      {selectedContainer && (
        <TodoListModal
          containerId={selectedContainer}
          onClose={() => setSelectedContainer(null)}
        />
      )}
    </div>
  );
};
```

Backend API Architecture (Express.js)

typescript

```
// Main Express Server Setup
import express from 'express';
import cors from 'cors';
import helmet from 'helmet';
import { Pool } from 'pg';

const app = express();
const port = process.env.PORT || 3000;

// Database connection (Replit PostgreSQL)
const pool = new Pool({
  connectionString: process.env.DATABASE_URL,
  ssl: process.env.NODE_ENV === 'production' ? { rejectUnauthorized: false } : false
});

// Middleware
app.use(helmet());
app.use(cors());
app.use(express.json());
app.use(express.static('public'));

// Authentication middleware (Replit Auth integration)
const authenticateUser = async (req: Request, res: Response, next: NextFunction) => {
  try {
    const authHeader = req.headers.authorization;
    if (!authHeader) {
      return res.status(401).json({ error: 'No authorization header' });
    }

    const token = authHeader.split(' ')[1];
    const replitUser = await verifyReplitAuthToken(token);

    // Get or create user in local database
    const user = await getOrCreateUser(replitUser);
    req.user = user;
    next();
  } catch (error) {
    res.status(401).json({ error: 'Authentication failed' });
  }
};

// API Routes
app.use('/api/ideas', authenticateUser, ideasRouter);
app.use('/api/containers', authenticateUser, containersRouter);
app.use('/api/sync', authenticateUser, syncRouter);
app.use('/api/export', authenticateUser, exportRouter);
```

```
app.listen(port, () => {
  console.log(`Clear Mind API running on port ${port}`);
});

// Ideas Controller with Staging
class IdeasController {
  static async createIdea(req: Request, res: Response) {
    try {
      const { title, description, containerId, createdFrom } = req.body;
      const userId = req.user.id;

      const result = await pool.query(`
        INSERT INTO ideas (title, description, container_id, created_from, last_modified_from, user_id)
        VALUES ($1, $2, $3, $4, $4, $5)
        RETURNING *
      `, [title, description, containerId, createdFrom, userId]);

      res.status(201).json(result.rows[0]);
    } catch (error) {
      console.error('Error creating idea:', error);
      res.status(500).json({ error: 'Failed to create idea' });
    }
  }

  static async stageCanvasChanges(req: Request, res: Response) {
    try {
      const { ideaId } = req.params;
      const { title, description, canvasX, canvasY } = req.body;
      const userId = req.user.id;

      // Check for existing todolist staging
      const todolistStaging = await pool.query(
        'SELECT * FROM ideas_staging_todolist WHERE original_idea_id = $1 AND user_id = $2',
        [ideaId, userId]
      );

      const hasConflict = todolistStaging.rows.length > 0;

      await pool.query(`
        INSERT INTO ideas_staging_canvas
        (original_idea_id, staged_title, staged_description, staged_canvas_x, staged_canvas_y, conflict_with_todolis
        VALUES ($1, $2, $3, $4, $5, $6, $7)
        ON CONFLICT (original_idea_id, user_id)
        DO UPDATE SET
          staged_title = EXCLUDED.staged_title,
          staged_description = EXCLUDED.staged_description,
```

```
        staged_canvas_x = EXCLUDED.staged_canvas_x,
        staged_canvas_y = EXCLUDED.staged_canvas_y,
        staged_at = NOW(),
        conflict_with_todolist = EXCLUDED.conflict_with_todolist
      `, [ideaId, title, description, canvasX, canvasY, hasConflict, userId]);

      if (hasConflict) {
        await pool.query(
          'UPDATE idea_containers SET sync_status = $1, requires_user_action = TRUE WHERE id = (SELECT contain
          ['conflict', ideaId]
        );
      }

      res.json({ success: true, hasConflict });
    } catch (error) {
      console.error('Error staging canvas changes:', error);
      res.status(500).json({ error: 'Failed to stage changes' });
    }
  }
}

// Sync Controller for Conflict Resolution
class SyncController {
  static async getContainerSyncStatus(req: Request, res: Response) {
    try {
      const { containerId } = req.params;
      const userId = req.user.id;

      const container = await pool.query(
        'SELECT * FROM idea_containers WHERE id = $1 AND user_id = $2',
        [containerId, userId]
      );

      if (container.rows.length === 0) {
        return res.status(404).json({ error: 'Container not found' });
      }

      const conflicts = await SyncService.detectConflicts(containerId, userId);

      res.json({
        container: container.rows[0],
        conflicts,
        requiresUserAction: conflicts.length > 0
      });
    } catch (error) {
      console.error('Error getting sync status:', error);
      res.status(500).json({ error: 'Failed to get sync status' });
```

```
    }
  }

  static async resolveConflict(req: Request, res: Response) {
    try {
      const { conflictId } = req.params;
      const { resolution } = req.body; // 'canvas' or 'todolist'
      const userId = req.user.id;

      await SyncService.applyConflictResolution(conflictId, resolution, userId);

      res.json({ success: true });
    } catch (error) {
      console.error('Error resolving conflict:', error);
      res.status(500).json({ error: 'Failed to resolve conflict' });
    }
  }
}
```

12-PHASE REPLIT AGENT DEVELOPMENT ROADMAP

Phase 1: Project Foundation & Database Setup (Replit Agent Prompts)

Prompt 1.1: Project Initialization

Create a new full-stack web application using the FULLSTACK_JS template with the following specifications:

Project Name: Clear Mind v2.0
Description: Dual-interface idea management platform with canvas and todolist views

Technology Stack:
- Frontend: React 18+ with TypeScript
- Backend: Express.js with Node.js
- Database: PostgreSQL
- Authentication: Replit Auth
- Styling: Tailwind CSS
- Package Manager: npm

Initial Project Structure:
- /src (React frontend)
- /server (Express.js backend)
- /database (PostgreSQL schema files)
- /public (static assets)

Create the basic project scaffolding with proper TypeScript configuration, Tailwind setup, and Express server initialization. Include package.json with all necessary dependencies for both frontend and backend.

Prompt 1.2: Database Schema Implementation

Set up the complete PostgreSQL database schema for the idea management system:

Create these tables with proper relationships:
1. users (id, replit_user_id, username, email, display_name, timestamps)
2. idea_containers (id, name, color, sync_status, canvas_positions, user_id)
3. ideas (id, title, description, container_id, priority_level, section_id, order_position, canvas_position_x, canvas_position_y, created_from, version_number, user_id)
4. sections (id, container_id, name, order_position, user_id)
5. ideas_staging_canvas (staging table for canvas changes)
6. ideas_staging_todolist (staging table for todolist changes)
7. conflict_resolution_log (audit trail for conflict resolutions)

Include these custom ENUM types:
- priority_enum ('low', 'medium', 'high', 'critical')
- interface_enum ('canvas', 'todolist')
- sync_status_enum ('synced', 'pending', 'conflict', 'resolving')
- conflict_type_enum ('content_diff', 'delete_vs_modify', 'structural')

Add proper foreign key constraints, indexes for performance, and populate with sample data for testing (3 users, 5 containers, 15 ideas with various states).

Phase 2: Authentication & Basic Backend (Week 1)

Prompt 2.1: Replit Auth Integration

```
Implement Replit authentication system with the following requirements:

Backend (Express.js):
- Create authentication middleware that verifies Replit Auth tokens
- Implement getOrCreateUser function that syncs Replit users with local users table
- Add protected route middleware for all API endpoints
- Create user profile endpoints for getting/updating user information

Frontend (React):
- Create useReplitAuth custom hook that manages authentication state
- Implement login/logout functionality using Replit Auth
- Create protected route components that redirect to login if not authenticated
- Add user profile display in header component

Include proper error handling, loading states, and TypeScript interfaces for all authentication-related types.
```

Prompt 2.2: Core API Endpoints

```
Create the foundational REST API endpoints for the application:

Ideas API (/api/ideas):
- GET /api/ideas - Get all ideas for authenticated user
- POST /api/ideas - Create new idea
- PUT /api/ideas/:id - Update existing idea
- DELETE /api/ideas/:id - Soft delete idea
- POST /api/ideas/:id/stage-canvas - Stage canvas changes
- POST /api/ideas/:id/stage-todolist - Stage todolist changes

Containers API (/api/containers):
- GET /api/containers - Get all containers for user
- POST /api/containers - Create new container
- PUT /api/containers/:id - Update container
- DELETE /api/containers/:id - Delete container and associated ideas

Include proper request validation, error handling, database transactions, and TypeScript interfaces for all request/response types. Add comprehensive logging for debugging.
```

Phase 3: Static UI Foundation (Week 2)

Prompt 3.1: React Component Structure

Create the complete React component structure with static UI and mock data:

Main Components:
1. App component with React Router setup (/, /canvas, /todolist routes)
2. Header component with logo "Clear Mind", tab navigation, user profile
3. CanvasView component with infinite canvas area, sidebar, and draggable cards
4. TodoListView component with grid layout and modal system
5. Dashboard component with overview statistics

Sidebar Components:
- CreateIdeaButton component
- ContainersList component with sync status indicators
- SyncIndicator component (green/amber/red dots)

Use Tailwind CSS for all styling with responsive design (mobile-first approach). Include mock data for 3 containers and 10+ ideas with various states. All components should be TypeScript with proper interfaces.

NO functionality yet - this is purely static UI with mock data and routing.

Prompt 3.2: Modal System Implementation

Implement a comprehensive modal system for the application:

Create these modal components:
1. CreateIdeaModal - Form for creating new ideas (title, description, container selection)
2. TodoListModal - Complex modal for managing todolist (tasks, sections, priorities)
3. ConflictResolutionModal - Interface for resolving data conflicts
4. ExportModal - Options for exporting todolists with priority filters

Modal Features:
- Full viewport coverage with overlay
- Perfect centering in viewport
- Prevent body scroll when modal is open
- ESC key closes modals
- Click outside to close
- Support for nested modals
- Mobile responsive (320px to 1920px)
- Smooth enter/exit animations

Include TypeScript interfaces for all modal props and state management with React hooks.

Phase 4: Canvas Interface Development (Week 3)

Prompt 4.1: Infinite Canvas with Drag & Drop

Implement the infinite canvas interface with advanced drag and drop functionality:

Canvas Features:
- Infinite scrollable area with zoom controls (50% to 200%)
- Drag and drop for idea cards with smooth animations
- Multi-select capability with Ctrl/Cmd + click
- Canvas position persistence (remember zoom level and pan position)
- Grid snapping (optional) for organized layouts

IdeaCard Component:
- Draggable cards with title and description
- Color-coded by container with visual grouping
- Hover states and selection indicators
- Context menus for card actions (edit, delete, change container)

Technical Implementation:
- Use React DnD or implement custom drag system
- Implement virtualization for performance with 100+ cards
- Canvas state management with React Context or Redux
- Smooth animations using Framer Motion or CSS transitions

Connect to staging API - when user drags cards, automatically stage changes in ideas_staging_canvas table.

Prompt 4.2: Canvas Sidebar and Grouping

Implement the canvas sidebar with idea creation and container management:

Sidebar Components:
1. CreateIdeaButton - Opens modal for new idea creation
2. ContainersList - Shows all user's containers with sync indicators
3. Container context menus - Edit, delete, "Update TodoList" options
4. Sync status indicators - Visual feedback for container sync states

Container Management:
- Create new containers with color selection
- Rename existing containers inline editing
- Delete containers with confirmation dialog
- "Update TodoList" button that creates/updates corresponding todolist

Visual Grouping:
- Ideas automatically group by color/container on canvas
- Visual connection lines between related ideas (optional)
- Container name labels on canvas groups
- Ability to select entire groups for bulk operations

Integrate with backend API for real container CRUD operations and sync status updates.

Phase 5: TodoList Interface Development (Week 4)

Prompt 5.1: TodoList Grid and Card System

Create the TodoList interface with grid layout and interactive cards:

Grid Layout:
- Responsive grid of TodoList cards (1-4 columns based on screen size)
- "Create New TodoList" card with plus icon
- Each TodoList card shows: title, task count, color indicator, last modified

TodoList Card Features:
- Click to open detailed modal
- Hover effects with subtle animations
- Context menu with options: Edit, Export, Delete
- Sync status indicators (same green/amber/red system as canvas)
- Loading states while fetching data

TodoList Modal (Complex):
- Header with editable title and close button
- Main area with sections and tasks
- Tasks with checkboxes, drag-and-drop reordering
- Priority indicators (high, medium, low, critical) with color coding
- "Create new section" input with + button
- "Create new task" input at bottom
- Export button with priority filtering options

Implement full CRUD operations connecting to backend API with proper error handling and optimistic updates.

Prompt 5.2: Advanced TodoList Features

Implement advanced TodoList management features:

Section Management:
- Create sections within TodoLists for organization
- Drag and drop tasks between sections
- Reorder sections with drag and drop
- Delete sections (move tasks to unsectioned area)

Task Features:
- Priority levels (critical, high, medium, low) with visual indicators
- Context menu per task: Edit, Change Priority, Move to Section, Delete
- Task completion with strikethrough and move to bottom
- Bulk operations: Select multiple tasks, bulk priority change

Export System:
- Export modal with filtering options
- Filter by priority level (all, high only, critical only, etc.)
- Include/exclude descriptions checkbox
- Include/exclude priority indicators checkbox
- Generate formatted text output for copying/printing
- Download as .txt file option

Stage all changes using ideas_staging_todolist table and trigger conflict detection when changes conflict with canvas staging.

Phase 6: Bidirectional Synchronization System (Week 5-6)

Prompt 6.1: Staging System Implementation

Implement the core bidirectional staging system for conflict-free synchronization:

Staging Service (Backend):
- StagingService class with methods for canvas and todolist staging
- Automatic conflict detection when staging changes
- Database transactions for atomic staging operations
- Cleanup methods for clearing staging data after resolution

Frontend Hooks:
- useSyncState hook for monitoring container sync status
- useStaging hook for staging changes from components
- Real-time sync status updates (polling every 3 seconds)
- Optimistic UI updates with rollback on conflict

Conflict Detection:
- Detect content differences (title, description changes)
- Detect delete vs modify conflicts
- Detect structural conflicts (section/position changes)
- Generate conflict objects with comparison data

API Endpoints:
- POST /api/ideas/:id/stage-canvas
- POST /api/ideas/:id/stage-todolist
- GET /api/containers/:id/sync-status
- POST /api/sync/resolve-conflict/:conflictId

Implement comprehensive error handling and logging for all staging operations.

Prompt 6.2: Conflict Resolution Interface

Create the conflict resolution system with step-by-step wizard interface:

ConflictResolutionModal Component:
- Wizard-style interface for resolving conflicts one by one
- Progress indicator (Conflict 1 of 3)
- Side-by-side comparison of Canvas vs TodoList versions
- Clear visual differences highlighting
- "Use Canvas Version" / "Use TodoList Version" buttons
- Bulk resolution options ("Use all Canvas versions")

Conflict Types:
1. Content Conflicts - Show title/description differences with timestamps
2. Delete vs Modify - Special handling for items deleted on one side, modified on other
3. Structural Conflicts - Position/section changes with visual indicators

Resolution Process:
- Apply selected version to production ideas table
- Clear staging data for resolved conflicts
- Update container sync_status to 'synced' when all conflicts resolved
- Log all resolution decisions in conflict_resolution_log table

Auto-merge capability for non-conflicting changes (different fields modified).

Phase 7: Export and Advanced Features (Week 7)

Prompt 7.1: Export System Implementation

Build comprehensive export functionality for TodoLists:

ExportService (Backend):
- Generate formatted text output from TodoList data
- Support priority filtering (all, high+critical only, etc.)
- Organize by sections with proper formatting
- Include/exclude descriptions and priority indicators
- Generate downloadable .txt files

Export Modal (Frontend):
- Priority filter dropdown (All, Critical Only, High+Critical, etc.)
- Include descriptions checkbox
- Include priority indicators checkbox
- Live preview of generated output
- Copy to clipboard button
- Download as file button

Export Formats:
- Plain text with bullet points and section headers
- Numbered lists with priority indicators
- Clean format for printing
- Include container name and export date

File naming convention: "ContainerName_YYYY-MM-DD.txt"

Prompt 7.2: Performance Optimization and Polish

Optimize the application for production use:

Performance Optimizations:
- Implement React.memo for expensive components
- Add virtualization for large lists (100+ ideas/tasks)
- Optimize database queries with proper indexing
- Add caching for frequently accessed data
- Lazy load modal components

User Experience Improvements:
- Loading states for all async operations
- Skeleton screens while loading data
- Optimistic updates with error rollback
- Smooth animations for all interactions
- Keyboard shortcuts (Ctrl+N for new idea, ESC for close modal)

Error Handling:
- Comprehensive error boundaries
- User-friendly error messages
- Retry mechanisms for failed requests
- Offline detection and graceful degradation

Production Readiness:
- Environment variable validation
- Proper logging and monitoring
- Security headers and CORS configuration
- Rate limiting for API endpoints

Phase 8-12: Advanced Features & Production (Week 8-12)

Phase 8: Real-time Collaboration

- Multi-user real-time editing with WebSockets
- User presence indicators on canvas
- Real-time conflict resolution with other users
- Collaborative cursors and live changes

Phase 9: Advanced Integrations

- Export to external tools (Notion, Trello, Linear)
- Import from common formats (CSV, JSON)
- API webhooks for external integrations
- Zapier/Integromat compatibility

Phase 10: Analytics and Insights

- User activity tracking and analytics
- Productivity metrics (ideas to completed tasks ratio)
- Usage patterns and optimization suggestions
- Team collaboration analytics

Phase 11: Production Deployment

- Replit deployment optimization
- Custom domain configuration
- Environment variable management
- Production monitoring and alerting

Phase 12: Testing and Quality Assurance

- Comprehensive end-to-end testing
- Conflict resolution scenario testing
- Performance testing under load
- Security audit and penetration testing

VALIDATION CRITERIA FOR REPLIT DEVELOPMENT

Phase 6 Validation (Core Features Complete):

- [ ] Canvas drag & drop works smoothly with 50+ ideas
- [ ] TodoList CRUD operations handle all edge cases
- [ ] Sync system detects conflicts within 5 seconds
- [ ] Export generates properly formatted outputs
- [ ] Replit Auth integration works seamlessly

Phase 6 Validation (Sync System Complete):

- [ ] Bidirectional staging prevents all data loss scenarios
- [ ] Conflict resolution handles all identified conflict types
- [ ] Visual indicators accurately reflect sync status
- [ ] User can resolve conflicts without technical knowledge
- [ ] System maintains data integrity under stress testing

Production Readiness on Replit:

- [ ] Application deploys successfully to .replit.app domain

- [ ] PostgreSQL database performs under concurrent user load
- [ ] All environment variables properly configured in Replit
- [ ] SSL certificates and security headers configured
- [ ] Custom domain integration working (if required)

REPLIT-SPECIFIC IMPLEMENTATION NOTES

Development Environment Setup

1. Create new Repl using FULLSTACK_JS template
2. Configure package.json with all required dependencies
3. Set up environment variables in Replit Secrets:
   - DATABASE_URL (automatically provided by Replit)
   - NODE_ENV=production
   - JWT_SECRET (for session management)
4. Configure .replit file for proper start commands
5. Set up database initialization scripts in /database folder

Deployment Configuration

yaml

```yaml
# .replit file configuration
modules = ["nodejs-20", "postgresql-15"]

[deployment]
run = ["sh", "-c", "npm run build && npm start"]
deploymentTarget = "gce"

[[ports]]
localPort = 3000
externalPort = 80

[env]
NODE_ENV = "production"
```

Database Initialization Script

sql

```sql
-- /database/init.sql
-- Run this in Replit PostgreSQL console to set up tables

-- Enable UUID extension
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Create custom types first
CREATE TYPE priority_enum AS ENUM ('low', 'medium', 'high', 'critical');
CREATE TYPE interface_enum AS ENUM ('canvas', 'todolist');
CREATE TYPE sync_status_enum AS ENUM ('synced', 'pending', 'conflict', 'resolving');
CREATE TYPE conflict_type_enum AS ENUM ('content_diff', 'delete_vs_modify', 'structural');

-- Users table with Replit Auth integration
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    replit_user_id VARCHAR(255) UNIQUE NOT NULL,
    username VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL,
    display_name VARCHAR(100),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Create indexes for performance
CREATE INDEX idx_users_replit_id ON users(replit_user_id);
CREATE INDEX idx_ideas_container ON ideas(container_id);
CREATE INDEX idx_ideas_user ON ideas(user_id);
CREATE INDEX idx_containers_user ON idea_containers(user_id);

-- Insert sample data for testing
INSERT INTO users (replit_user_id, username, email, display_name) VALUES
('sample_user_1', 'testuser', 'test@example.com', 'Test User');
```

SUCCESS METRICS AND KPIs

Development Phase Metrics

- **Phase Completion Rate**: Target 100% within timeline
- **Bug Detection Rate**: <3 critical bugs per phase
- **Code Quality Score**: >85% (TypeScript coverage, ESLint compliance)
- **Performance Benchmarks**: <2s page load, <500ms API response

User Experience Metrics (Post-Launch)

- **Task Completion Rate**: Ideas → Completed tasks >80%
- **Sync Conflict Resolution Success**: >95% resolved without data loss
- **User Satisfaction Score**: >4.5/5.0 (based on feedback)
- **Daily Active Usage**: >90% of registered users

Technical Performance Metrics

- **System Uptime**: >99.5% availability
- **Database Performance**: <100ms average query time
- **Concurrent Users**: Support 50+ simultaneous users
- **Data Integrity**: Zero data loss incidents

Business Metrics

- **User Retention**: >85% after 30 days
- **Feature Adoption**: Sync system used by >95% of active users
- **Support Volume**: <2% of user base requiring support monthly
- **Development Velocity**: Maintain 1 feature per week post-MVP

RISK MITIGATION STRATEGIES

Technical Risks

Risk: Replit platform limitations for complex WebSocket connections **Mitigation**: Use HTTP polling (3-5 second intervals) as primary sync method, implement WebSocket as enhancement

Risk: PostgreSQL performance with complex staging queries **Mitigation**: Implement database indexing strategy, use query optimization, regular performance monitoring

Risk: React performance with large datasets (100+ ideas) **Mitigation**: Implement virtualization using react-window, lazy loading, component memoization

Platform-Specific Risks

Risk: Replit deployment limitations or downtime **Mitigation**: Maintain GitHub backup of all code, document migration process to alternative platforms

Risk: Replit Auth service dependencies **Mitigation**: Implement graceful fallback authentication, user session persistence

User Experience Risks

Risk: Conflict resolution complexity overwhelming users **Mitigation**: Implement auto-merge for simple conflicts, provide "Smart Auto-Resolve" options

Risk: Canvas performance on mobile devices **Mitigation**: Mobile-optimized touch interactions, reduced animations on mobile, responsive breakpoints

---

COMPETITIVE ANALYSIS

Clear Mind vs. Existing Solutions

vs. Miro/FigJam:

- **Advantage**: Direct idea-to-task conversion, structured productivity focus
- **Disadvantage**: Less advanced visual collaboration tools
- **Differentiation**: Bidirectional sync between creative and structured modes

vs. Notion/Obsidian:

- **Advantage**: Visual ideation with drag-drop, faster idea capture
- **Disadvantage**: Less advanced knowledge management features
- **Differentiation**: Canvas-first approach with seamless task integration

vs. Trello/Asana:

- **Advantage**: Visual brainstorming before task structuring
- **Disadvantage**: Less mature project management features
- **Differentiation**: Creative ideation integrated with task management

Unique Value Propositions

1. **Bidirectional Sync Architecture**: Only tool that maintains data integrity between visual and structured modes
2. **Conscious Conflict Resolution**: User-controlled merge decisions prevent data loss
3. **Canvas-to-Task Pipeline**: Seamless flow from creative ideation to executable tasks
4. **Zero Setup Complexity**: Replit-native deployment eliminates infrastructure overhead

---

POST-MVP ROADMAP

Version 2.1 (Month 2-3)

- Real-time collaborative editing with WebSockets

- Mobile app development (React Native or PWA)
- Advanced export formats (PDF, JSON, integration APIs)
- Team workspaces with role-based permissions

Version 2.2 (Month 4-6)

- AI-powered idea categorization and suggestion system
- Integration marketplace (Notion, Linear, Trello, Slack)
- Advanced analytics dashboard with productivity insights
- Voice-to-text idea capture

Version 3.0 (Month 7-12)

- Enterprise features (SSO, advanced security, audit logs)
- Template marketplace for common project types
- Advanced automation with workflow triggers
- White-label solution for enterprise clients

CONCLUSION

This Replit-optimized PRD provides a comprehensive roadmap for building Clear Mind v2.0 using Replit Agent's strengths while maintaining the sophisticated architecture we designed. The specification addresses:

Technical Compatibility: Full alignment with Replit's FULLSTACK_JS capabilities (React + Express + PostgreSQL + Replit Auth)

Agent-Optimized Development: Each phase includes specific prompts designed for Replit Agent's natural language interface

Production-Ready Architecture: Enterprise-grade bidirectional synchronization system that scales with user needs

Risk Mitigation: Comprehensive fallback strategies and platform-specific considerations

Clear Success Path: Detailed validation criteria and metrics that ensure project success

The key differentiator remains the bidirectional staging architecture with conscious conflict resolution - a sophisticated system that can be built incrementally using Replit Agent's capabilities while delivering enterprise-grade reliability.

This PRD transforms your career-defining vision into an executable, technically sound implementation plan that leverages Replit's platform advantages while maintaining architectural sophistication.