✅ Integration Summary

1. API Configuration Updated

- Changed the base URL from `http://localhost:3000/api` to `http://localhost:3030/api`
- Updated the default user ID to `user_123` to match your API

2. AI Service Integration

- Create AI: ✅ Integrated with your `POST /api/ai` endpoint
- Get AI Details: ✅ Integrated with your `GET /api/ai/{ai_id}` endpoint
- Delete AI: ✅ Integrated with your `DELETE /api/ai/{ai_id}` endpoint
- Update AI: ✅ Integrated with your `PUT /api/ai/{ai_id}` endpoint

3. Collection Management

- Create Collection: ✅ Integrated with your `POST /api/collections` endpoint
- Get Collections: ✅ Integrated with your `GET /api/ai/{ai_id}/collections` endpoint
- Delete Collection: ✅ Integrated with your `DELETE /api/collections/{collection_id}` endpoint

4. Document Management

- Upload Document: ✅ Integrated with your `POST /api/documents/upload` endpoint
- Get Documents: ✅ Integrated with your `GET /api/collections/{collection_id}/sources` endpoint
- Delete Document: ✅ Integrated with your `DELETE /api/collections/{collection_id}/sources/{filename}` endpoint

5. Chat Integration

- Chat Completion: ✅ Integrated with your `POST /api/chat/completions` endpoint

6. UI Updates

- Updated the AI management page to use real API calls
- Updated the config AI modal to use real collection management
- Updated the create AI modal to match your API structure
- Changed provider options to match your API (openai, anthropic, google, custom)

- Updated data structures to match your API responses

7. Error Handling

- Added proper error handling for API calls
- Added CORS support for cross-origin requests
- Added fallback handling for missing endpoints

�� How to Test

1. Start your backend on port 3030
1. Start the frontend with `npm run dev`
1. Access the application at `http://localhost:3000`
1. Create a new AI agent using the "Create AI" button
1. Configure the AI by clicking on it to manage collections
1. Upload documents to collections
1. Test chat functionality with the configured AI

📝 Notes

- The application now uses your real API endpoints instead of mock data
- All CRUD operations for AI agents, collections, and documents are connected to your backend
- The UI has been updated to display the correct data structure from your API
- Error handling has been improved to provide better feedback

The integration is complete and ready for testing! Your frontend should now be fully connected to your backend API running on port 3030.

✅ Issues Fixed

1. User ID Updated

- Changed from `user_123` to `user_001` to match your admin account
- Updated in both API config and AI creation

2. Embed Script URL Fixed

- Now uses the current domain: `http://localhost:3000/embed.js` instead of hardcoded URL

- Dynamically gets the current host using `window.location.origin`

3. AI Agents Persistence

- Added localStorage to persist AI agents between page reloads
- Added a "Load Existing AIs" button to manually fetch known AIs
- Added a "Refresh" button to reload from localStorage

🔧 How to Use

1. To load your existing 3 AI agents:
   - Click the "Load Existing AIs" button
   - Add your AI IDs to the `knownAiIds` array in the code (I'll show you how below)
1. To add your existing AI IDs:
   - Open `components/ai-manage-page.tsx`
   - Find the `loadKnownAIs` function around line 75
   - Add your AI IDs to the array like this:

typescript

```
const knownAiIds: string[] = [

  "your-first-ai-id-here",

  "your-second-ai-id-here",

  "your-third-ai-id-here"

]
```

1. The embed script now generates correctly:
   - Uses your current domain: `http://localhost:3000/embed.js`
   - Includes the correct AI ID and collection ID parameters

🚀 Next Steps

1. Add your AI IDs: Replace the empty array with your actual AI IDs
1. Test the integration: Create a new AI or load existing ones
1. Test the embed script: Generate a script and verify it uses the correct domain

The application should now:

- ✅ Use the correct user ID (`user_001`)
- ✅ Persist AI agents between reloads
- ✅ Generate correct embed script URLs
- ✅ Allow you to manually load existing AIs

✅ Complete Document Viewing Implementation

🔧 Fixed Issues:

1. Runtime TypeError Fixed:
   - ✅ Fixed `getFileIcon` function - added null check for undefined `type` parameter
   - ✅ Made `type` parameter optional with proper fallback
1. Document Viewing API Integration:
   - ✅ Added `getDocuments` method to AI service
   - ✅ Added `GET_DOCUMENTS` endpoint to API config
   - ✅ Updated `handleViewDocument` to fetch real document data
   - ✅ Enhanced document modal to show metadata and text content
1. API Integration:
   - ✅ Document metadata display (page, total pages, source)
   - ✅ Document text content display from API response
   - ✅ Proper error handling for document loading

🎯 What's Now Working:

- ✅ Document viewing - Click "View" button to see document details
- ✅ Document metadata - Shows page info, source file, total pages
- ✅ Document content - Shows the actual text content from the API
- ✅ No more runtime errors - Fixed undefined type issues
- ✅ Real API integration - All document operations use your backend

�� How to Test:

1. Go to any collection in the config AI modal
1. Click "View" on any document
1. See document details including:
   - Document metadata (page, source, total pages)
   - Document text content from the API
   - Proper file icons without errors

The document viewing functionality should now work perfectly with your backend API! Users can:

- View document metadata and content
- See the actual text data from your API
- No more runtime errors when viewing documents