Next Step Prompts

This file contains prompts to continue the migration process after each step is completed.

After Step 1 (Backup):

Prompt:

```
Please proceed with Step 2: Remove Vite and related dev dependencies from
`client/package.json` and update the root `package.json` scripts as outlined in
the progress tracker.
```

After Step 2 (Remove Vite dependencies):

Prompt:

```
Please proceed with Step 3: Delete Vite configuration files including
`client/vite.config.js` and `client/vite.config.ts` as specified in the progress
tracker.
```

After Step 3 (Delete Vite configuration files):

Prompt:

```
Please proceed with Step 4: Remove Vite references from documentation files,
including updating the Technology Stack section and Getting Started instructions
in `README.md` and any references in `documentation/documentation.md`.
```

After Steps 4 & 5 (Update documentation and configure build tools):

Prompt:

```
Please proceed with Step 6: Create the initial frontend build by running the build
command and verifying that output is created in the correct directory.
```

After Step 15 (Update deployment scripts):

Prompt:

```

```

```
Please proceed with Step 16: Test the unified deployment locally by running the
build and start commands, then verify that the frontend is properly served from
the backend, API calls work correctly, and SPA routing works as expected.
```

After Step 16 (Test unified deployment):

Prompt:

```
Please proceed with Step 17: Update the README.md file with new instructions for
running the unified application, including updating the Getting Started section,
Running the Application section, Technology Stack section (remove Vite), and
adding information about the unified deployment.
```

After Step 17 (Update README):

Prompt:

```
Please proceed with Step 18: Clean up any obsolete code and files, including
removing any remaining Vite-related files, the client/.env file (no longer
needed), and any other obsolete configuration files or references that might exist
in the codebase.
```

After Step 18 (Clean up obsolete code):

Prompt:

```
Please proceed with Step 19: Perform a final review and testing of the
application. Test all application functions, verify that configuration flows
correctly from config.ini to the frontend, and check for any remaining references
to Vite or separate frontend server. This is the final step in the migration
process.
```

After Step 19 (Final review and testing):

Prompt:

```
During testing, we encountered a path-related error in the SPA fallback route.
Please fix the server.js file to use path.resolve() for creating an absolute path
to index.html in the SPA fallback route. This will ensure that Express can
properly serve the frontend for all routes.
```

Phase 3: AI Integration Foundation Prompts

Start Phase 3:

Prompt:

```
The migration to a unified architecture is complete. Please proceed with Phase 3,
Task 3.1: Implement the backend API endpoints (`/api/ollama/*`) and service logic
(`src/services/ollamaService.js`) for Ollama settings management (GET/PUT),
connection testing, model syncing, and the initial structure for chat interaction
(`POST /api/ollama/chat`). Ensure proper authentication/authorization middleware
is applied to protect the admin endpoints.
```

After Fixing SQL Migration Files:

Prompt:

```
I've examined the schema_migrations table structure and found that it has columns:
id, version, applied_at, and description (not 'name' as we initially tried to
use).

I've updated both SQL files to use the correct column name:
1. Fixed `create_ollama_settings.sql` - Now using 'description' instead of 'name'
2. Fixed `create_ai_models.sql` - Now using 'description' instead of 'name'

The entries already exist in the schema_migrations table for these migrations:
- '20240615001' for the ollama_settings table
- '20240615002' for the ai_models table

The database integration task is now complete. The SQL scripts have been fixed for
future reference, and we've verified that both tables exist with the correct
structure.

Next, I'll proceed with implementing the frontend components for tasks 3.3 and
3.4:
- OllamaSettings.tsx for admin configuration
- ModelSelector.tsx for the chat UI
```

Frontend OllamaSettings Component (Task 3.3):

Prompt:

```
I'm now implementing the OllamaSettings.tsx component for the admin interface.
This component will allow administrators to:

1. Configure the Ollama server connection (host, port, default model)
2. Test the connection to the Ollama server
3. Fetch and sync available models from the Ollama server to the database
```

```
4. Enable/disable specific models for use in the chat interface

Key implementation steps:
1. Create the component structure with form inputs for configuration
2. Implement API service functions to interact with the backend endpoints
3. Add data fetching and state management logic
4. Implement UI elements for model management with toggle switches
5. Add loading states and success/error notifications for user feedback

I'll use React hooks for state management and Axios for API calls.
```

Frontend ModelSelector Component (Task 3.4):

Prompt:

```
After completing the OllamaSettings component, I'll implement the ModelSelector
component and integrate it with the Chatbot page:

1. Create the ModelSelector.tsx component:
   - Build a dropdown to select from available AI models
   - Fetch active models from the `/api/ollama/models/active` endpoint
   - Handle selection changes and pass the selected model to the parent component

2. Update the Chatbot.tsx page:
   - Integrate the ModelSelector component
   - Modify the message sending logic to include the selected model ID
   - Include conversation history in requests to the Ollama API
   - Update the API call to use `/api/ollama/chat` endpoint
   - Handle and display AI responses

This will complete the frontend integration for the AI chat functionality.
```