BinBot Implementation Plan

Overview

This plan breaks down the BinBot implementation into small, testable phases. Each phase builds incrementally and can be tested independently. UI remains minimal until the final phase.

Phase 1: Foundation Setup (Core Infrastructure)

Goal: Basic project structure with minimal working FastAPI server

Tasks:

1. **Create project structure**
   - Create all required directories and files
   - Set up basic `requirements.txt` (include openai or google-generativeai)
   - Create minimal `config.yaml` with external LLM configuration
   - Create basic `api_schemas.py` with Pydantic models
2. **Basic FastAPI server**
   - Minimal `app.py` with FastAPI initialization
   - Basic `/health` endpoint that returns system status
   - Configuration loading from `config.yaml`
   - Environment variable support for API keys
   - Basic error handling structure
3. **Docker setup**
   - Create `Dockerfile` for Python 3.11 base
   - Create `start.sh` script
   - Test container builds and runs
4. **Minimal frontend**
   - Basic `index.html` with no styling
   - Single text input and submit button
   - JavaScript to call `/health` endpoint

Testing:

- Container builds successfully
- FastAPI server starts and responds to `/health`
- Frontend loads and can call backend
- Configuration loads correctly

Phase 2: ChromaDB Integration (Database Layer)

Goal: ChromaDB working with basic collections and persistence

Tasks:

1. **ChromaDB initialization**
    - Add ChromaDB to requirements
    - Initialize persistent ChromaDB client
    - Create `inventory` and `audit_log` collections
    - Add embedding model configuration
2. **Basic data models**
    - Define item schema in ChromaDB
    - Test adding/retrieving simple documents
    - Implement basic error handling for DB operations
3. **Health check enhancement**
    - Add ChromaDB status to `/health` endpoint
    - Test database connectivity

Testing:

- ChromaDB initializes and persists data
- Collections are created successfully
- Basic documents can be added and retrieved
- Health check reports database status

Phase 3: Basic Search Functionality (Core Feature)

Goal: Simple text-based search without LLM integration

Tasks:

1. **Search endpoint**
    - Implement `GET /search` endpoint
    - Use simple text matching (no embeddings yet)
    - Return standardized response format
    - Add basic pagination
2. **Frontend search**
    - Add search form to frontend
    - Display search results in plain text list
    - Handle empty results gracefully

3. **Test data**
    - Create script to populate test inventory data
    - Add various items across different bins

Testing:

- Search endpoint returns results
- Frontend displays search results
- Pagination works correctly
- Empty searches handled properly

Phase 4: LLM Integration (AI Layer)

Goal: Connect to external LLM service (OpenAI/Gemini) for embedding generation and search

Tasks:

1. **LLM client setup**
    - Add OpenAI or Google AI client to backend
    - Implement API key configuration via environment variables
    - Add error handling for API rate limits and failures
    - Test basic LLM connectivity
2. **Embedding generation**
    - Implement embedding generation using external API (e.g., OpenAI text-embedding-ada-002)
    - Update search to use vector similarity
    - Add embedding model version tracking
    - Implement caching to reduce API calls
3. **Enhanced search**
    - Replace text matching with vector search
    - Implement similarity scoring
    - Add confidence thresholds

Testing:

- External LLM API connection works
- Embeddings are generated successfully via API
- Vector search returns relevant results
- Error handling works for API failures and rate limits
- Embedding caching reduces redundant API calls

Phase 5: Add Items Functionality (CRUD Operations)

Goal: Users can add items to inventory

Tasks:

1. **Add endpoint**
   - Implement `POST /add` endpoint
   - Handle single item additions
   - Generate embeddings for new items
   - Store in ChromaDB with metadata
2. **Frontend add form**
   - Add simple form with item name and bin number
   - Handle form submission
   - Display success/error messages
3. **Audit logging**
   - Implement basic audit log entries
   - Track item additions with timestamps

Testing:

- Items can be added successfully
- Embeddings are generated and stored
- Audit log captures additions
- Frontend form works correctly

Phase 6: Remove Items Functionality

Goal: Users can remove items from inventory

Tasks:

1. **Remove endpoint**
   - Implement `POST /remove` endpoint
   - Search for items to remove
   - Handle item deletion from ChromaDB
2. **Basic disambiguation**
   - Return multiple matches when found
   - Simple list format for user selection
3. **Frontend remove interface**
   - Add remove form
   - Display disambiguation options as simple list

    - Handle confirmation flow

Testing:

- Items can be removed successfully
- Disambiguation works for multiple matches
- Audit log captures removals
- Frontend handles disambiguation flow

Phase 7: Move Items Functionality

Goal: Users can move items between bins

Tasks:

1. **Move endpoint**

    - Implement `POST /move` endpoint
    - Update bin_id metadata
    - Handle disambiguation like remove

2. **Frontend move interface**

    - Add move form with source/target bins
    - Reuse disambiguation interface

Testing:

- Items can be moved between bins
- Bin metadata updates correctly
- Audit log captures moves
- Disambiguation works for moves

Phase 8: Context Management (Session State)

Goal: Multi-turn conversations with context awareness

Tasks:

1. **Session management**

    - Implement in-memory session store
    - Track current bin_id per session
    - Add session timeout handling

2. **Context-aware operations**

    - Modify add/remove/move to use context
    - Auto-apply bin_id from context

- Handle context reset commands

3. **Frontend session handling**

    - Track session state in JavaScript
    - Display current context to user

Testing:

- Context is maintained across requests
- Operations use context appropriately
- Context resets work correctly
- Frontend shows current context

Phase 9: Bulk Operations & Transactions

Goal: Handle multiple items in single operations

Tasks:

1. **Bulk add functionality**

    - Parse multiple items from single input
    - Implement transaction management
    - Add rollback on failures

2. **Transaction handling**

    - Generate bulk transaction IDs
    - Atomic commit/rollback operations
    - Enhanced audit logging for bulk ops

Testing:

- Multiple items can be added at once
- Transactions rollback on failures
- Audit log tracks bulk operations
- Performance is acceptable for bulk ops

Phase 10: Natural Language Text Interface

Goal: Create a simple text-based interface for natural language inventory commands

Tasks:

1. **Natural language command parser**

    - Create LLM-based command interpretation
    - Parse commands like "add bolts to bin 3"

   - Extract action, items, and locations
   - Handle context and follow-up commands

2. **Command processing engine**

   - Map parsed commands to API calls
   - Implement context retention ("also add nuts")
   - Support multiple command types (add, remove, move, search)
   - Add command validation and confirmation

3. **Simple text interface frontend**

   - Create chat-like interface for command input
   - Display command results and confirmations
   - Show command history
   - Add help and examples

4. **Context-aware processing**

   - Remember previous bin/item context
   - Support follow-up commands
   - Handle ambiguous references
   - Maintain conversation state

Commands to Support:

- "add bolts to bin 3" → Parse item, action, location
- "also add nuts" → Use previous context (bin 3)
- "remove wires from bin 2" → Remove operation
- "move screws from bin 1 to bin 5" → Move operation
- "search for electronics" → Search operation
- "what's in bin 7?" → List bin contents
- "undo last command" → Rollback functionality

Testing:

- Test basic command parsing accuracy
- Test context retention between commands
- Test error handling for ambiguous commands
- Test all supported command types
- Test complex multi-step operations

Phase 11: Undo Functionality

Goal: Users can undo recent operations

Tasks:

1. **Undo endpoint**

    - Implement `POST /undo` endpoint
    - Restore previous states from audit log
    - Handle bulk operation undos
2. **Frontend undo interface**
    - Add prominent undo button
    - Show what will be undone
    - Confirm undo operations

Testing:

- Single operations can be undone
- Bulk operations can be undone atomically
- Undo creates proper audit entries
- Frontend undo interface works

Phase 12: Image Support (Multimodal)

Goal: Users can attach images to items

Tasks:

1. **Image upload handling**
    - Add file upload to add endpoint
    - Store images in persistent volume
    - Generate image paths in metadata
2. **Basic image display**
    - Show images in search results
    - Handle missing images gracefully
3. **Frontend image support**
    - Add file input to add form
    - Display images in results

Testing:

- Images can be uploaded and stored
- Images display in search results
- File paths are stored correctly
- Missing images handled gracefully

Phase 13: Voice Interface (Web Speech API)

Goal: Voice input for natural interaction

Tasks:

1. **Frontend voice integration**

   - Add microphone button
   - Implement Web Speech API
   - Convert speech to text for existing endpoints

2. **Voice feedback**

   - Add text-to-speech for responses
   - Handle voice command errors

Testing:

- Voice input works in supported browsers
- Speech is converted to text accurately
- Voice commands trigger correct actions
- Fallback works for unsupported browsers

Phase 14: Advanced Features & Polish

Goal: Complete remaining advanced features

Tasks:

1. **Advanced disambiguation**

   - Improve similarity scoring
   - Better disambiguation UI
   - Confidence score display

2. **Enhanced error handling**

   - Comprehensive error messages
   - Retry mechanisms
   - Better fallback strategies

3. **Performance optimization**

   - Embedding caching
   - Query optimization
   - Response time improvements

Testing:

- All error scenarios handled gracefully
- Performance meets requirements
- Advanced features work reliably

Phase 15: UI Enhancement & Final Polish

Goal: Improve user interface and experience

Tasks:

1. **UI styling**
   - Add Tailwind CSS
   - Improve layout and design
   - Add responsive design
2. **UX improvements**
   - Better loading states
   - Improved feedback messages
   - Keyboard shortcuts
3. **Final testing**
   - End-to-end testing
   - Performance testing
   - User acceptance testing

Testing:

- UI is polished and responsive
- All features work together seamlessly
- Performance is acceptable
- User experience is smooth

Testing Strategy for Each Phase

Unit Testing

- Test individual functions and endpoints
- Mock external dependencies (Ollama, ChromaDB)
- Validate request/response schemas

Integration Testing

- Test API endpoints with real ChromaDB
- Test frontend-backend integration
- Test Docker container functionality

Manual Testing

- Test user workflows manually
- Verify error handling

- Test edge cases

Acceptance Criteria

Each phase must pass all tests before proceeding to the next phase.