This is a comprehensive roadmap designed for an **Agentic Coding** workflow (specifically using tools like Cline/Roo-Code).

The strategy here is **"Vertical Slices with a Strong Foundation."** We will first establish the full backend infrastructure, then build the frontend foundation, and finally iterate through features feature-by-feature.

Pre-requisite: Context Loading

Before starting Step 1, you must upload the Spec.pdf to your Agent's context window.

Initial Prompt to Agent:

> "I am uploading a specification for the 'Purchase Organizer Application'. Please read it thoroughly. We will be building this strictly according to this document using Python/FastAPI for the backend and React/Vite/Tailwind for the frontend. Do not write code yet, just acknowledge you have understood the architecture and stack."

Phase 1: Project Initialization & Infrastructure

Step 1: Scaffolding and Directory Structure

Goal: Create the folder structure and configuration files.

Prompt:

> "Create the project directory structure. We need a root folder containing:
>
> 1. backend/: For the FastAPI application.
> 2. frontend/: For the React application.
> 3. deployment/: For scripts.
> 4. config.yaml: Create the file based on the template in **Section 13** of the spec.
> 5. Create a README.md listing these folders."

Step 2: Backend Setup (FastAPI & Dependencies)

Goal: Initialize the Python environment.

Prompt:

> "In the backend/ folder:

1. Create a requirements.txt with: fastapi, uvicorn, sqlalchemy, psycopg2-binary, python-multipart, pyyaml, python-jose[cryptography], passlib[bcrypt], opencv-python-headless, pytesseract, mistralai.

2. Create main.py that loads the config.yaml and initializes a basic FastAPI app with CORS middleware (allow localhost:5173).

3. Create a database.py file with the connection logic described in **Section 9.4**. Use SQLAlchemy. Support both SQLite and PostgreSQL based on the config."

Step 3: Frontend Setup (Vite + Tailwind)

Goal: Initialize the React environment.

Prompt:

"In the frontend/ folder:

1. Initialize a new React app using Vite (npm create vite@latest . -- --template react).

2. Install dependencies: npm install axios react-router-dom tailwindcss postcss autoprefixer lucide-react recharts zustand clsx tailwind-merge.

3. Initialize Tailwind CSS (npx tailwindcss init -p).

4. Configure tailwind.config.js to include the specific color palette defined in **Section 15.1** (Deep Blue, Light Gray, Vibrant Green, Alert Red, Charcoal Gray)."

Phase 2: Backend Core (Database & Auth)

Step 4: Database Models

Goal: Implement the Schema.

Prompt:

"In backend/models.py, create the SQLAlchemy models based strictly on the Database Structure in Section 11 of the spec.

Include models for: User, Purchase, Item, Contributor, Category, PurchaseLog, FriendlyName, and PasswordResetToken.

Ensure all Foreign Keys and constraints (like ON DELETE CASCADE where implied) are correct."

Step 5: Database Logic Layer (CRUD)

Goal: Encapsulate DB access.

Prompt:

"Update backend/database.py to implement the specific functions listed in Section 9.4.1 (User Management) and Section 9.4.2 (Purchase Management).

Implement: create_user, get_user_by_name, create_purchase, get_purchase_by_id.

Ensure you use the database.py module pattern as requested in the spec."

Step 6: Authentication System

Goal: Login and Security.

Prompt:

"Create backend/auth.py and backend/routers/auth.py.

1. Implement password hashing using passlib.
2. Implement JWT token creation and validation.
3. Create a /token endpoint that validates users against the DB and returns a JWT.
4. Implement the 'Login' workflow described in **Section 6.8**."

Phase 3: Frontend Foundation & Navigation

Step 7: Global State & API Client

Goal: Connect Front to Back.

Prompt:

"In the frontend:

1. Create an api/axios.js instance configured with the base URL of the backend.
2. Create a Zustand store (store/authStore.js) to manage the authentication state (token, user info, login/logout actions).
3. Implement the login action that calls the backend and stores the JWT."

Step 8: Login Page

Goal: The entry point.

Prompt:

"Create the Login Page (pages/LoginPage.jsx) following the design in **Section 5.1**.

1. Use the Card-based layout (centered on desktop, full width on mobile).
2. Include the 'User name' and 'Password' (with visibility toggle) fields.
3. Style the 'Log In' button with the Primary Deep Blue color.
4. Integrate with the authStore to handle submission and error states."

Step 9: Persistent Navigation

Goal: The App Shell.

Prompt:

"Create a components/Layout.jsx that includes the **Persistent Navigation Bar** described in **Section 5.4.1**.

1. Left side: Logo + Welcome message.
2. Right side: Hamburger menu (icon) that opens a dropdown with links: Home, Purchases, Dashboard, Settings, Logout.
3. Ensure the layout is responsive."

Phase 4: Purchase Management (Manual Creation)

Step 10: Backend - Items & Categories

Goal: Support for complex purchase structures.

Prompt:

> "In the backend, implement the logic for **Section 9.4.3 (Item Functions)** and **Section 9.4.4 (Category Functions)**.
>
> 1. Implement add_item_to_purchase, update_item, delete_item.
> 2. Implement the **Friendly Name Mapping Logic** (Pseudocode in **Section 9.1**) inside a new service backend/services/mapping_service.py."

Step 11: Purchase Creation Page (UI Layout)

Goal: The complex form.

Prompt:

> "Create pages/PurchaseEditor.jsx for Section 5.6 (Item Review & Editing Page).
>
> This page is used for both manual creation and reviewing scans.
>
> 1. **Section 1:** Purchase Metadata (Name, Date, Payer).
> 2. **Section 2:** The Interactive Item List. Use a card view for items as described in **Section 5.6.1**. Implement the drag-and-drop handle UI (no logic yet) and the 3-level Category dropdowns.
> 3. **Section 3:** Actions Footer (Confirm/Delete)."

Step 12: Purchase Creation Logic

Goal: Wiring up the form.

Prompt:

> "Connect PurchaseEditor.jsx to the backend.
>
> 1. Implement the 'Add Item' button to add a blank card.

2. Implement the Category logic: Category 3 is disabled if 2 is empty, etc. (**Section 10.2**).

3. Implement the 'Confirm Purchase' action that sends the full JSON payload to the backend create_purchase endpoint."

Step 13: Main Page (Recent Activity)

Goal: Dashboard landing.

Prompt:

"Create pages/MainPage.jsx following **Section 5.4**.

1. **Action Hub:** Large buttons for 'Scan Receipt' and 'Create Purchase'.

2. **Recent Activity:** Fetch the 5 most recent purchases from the backend and display them as cards."

Phase 5: Receipt Scanning (AI Integration)

Step 14: Image Processing Service (Backend)

Goal: OCR and Cleanup.

Prompt:

"Implement the Image Processing pipeline in backend/services/ocr_service.py.

1. Copy the preprocess_image code exactly from **Section 9.3**.

2. Implement extract_information using pytesseract.

3. Create a FastAPI endpoint /upload that accepts images, runs this pipeline, and returns extracted text."

Step 15: Mistral AI Analysis (Backend)

Goal: Extract structured data.

Prompt:

"Extend backend/services/ocr_service.py to include the analyze_information function using the Mistral API.

1. Use the exact system message provided in **Source 557**.
2. Ensure the API Key is read from environment variables/config.
3. Update the /upload endpoint to return the JSON structure of items/prices."

Step 16: Scan Receipt Page (Frontend)

Goal: Upload UI.

Prompt:

"Create pages/ScanReceiptPage.jsx based on **Section 5.5**.

1. **Step 1:** Drag & Drop zone for images.
2. **Step 2:** Image Staging (Grid view with Rotate/Crop buttons).
3. **Step 3:** 'Scan Receipt' button that sends images to the backend.
4. On success, redirect to the PurchaseEditor page pre-filled with the extracted data."

Phase 6: Advanced Features

Step 17: Purchase Review & Filtering

Goal: The Searchable Archive.

Prompt:

"Create pages/PurchaseList.jsx for **Section 5.7**.

1. Implement the Search Filter (by title) and Sort Dropdown.
2. Display the Grid of Purchase Cards.
3. Ensure the cards show the Payer, Total Cost, and Contributor Avatars."

Step 18: Dashboard & Analytics

Goal: Data Visualization.

Prompt:

"Create pages/DashboardPage.jsx based on **Section 5.8**.

1. **Left Column:** Filters (Time, Category 1/2/3).
2. **Right Column:**
    - KPI Cards (Total Spending, Count, Average).
    - Use recharts to implement the Scatter Plot (Time vs Cost).
    - (Optional) Implement the Sankey diagram if a suitable library is available, otherwise focus on the Scatter Plot."

Step 19: User Management (Admin)

Goal: Admin Controls.

Prompt:

"Create pages/UserManagement.jsx (**Section 5.10**).

1. Protect this route: only accessible if user.is_admin is true.
2. Display all users in a Grid Card view.
3. Implement the 'Manage' modal to Revoke/Grant Admin rights and Delete users."

Step 20: Settings Page

Goal: User Profile.

Prompt:

"Create pages/SettingsPage.jsx (**Section 5.9**).

1. Display User Name (read-only).
2. Implement the 'Change Password' form.
3. Connect to a backend endpoint update_user_password."

Phase 7: Final Polish & Deployment

Step 21: UI Polish

Goal: Design Consistency.

Prompt:

> "Review the entire application against the Design guidelines in **Section 15**.
>
> 1. Ensure all buttons use the specific Color Palette.
> 2. Check font sizes and spacing.
> 3. Ensure mobile responsiveness (hamburger menus, card stacking) works as described."

Step 22: Dockerization

Goal: Deployable Artifact.

Prompt:

> "Create a Dockerfile and docker-compose.yml in the root.
>
> 1. **Backend:** Python 3.10 image, install requirements, run uvicorn.
> 2. **Frontend:** Node builder image to npm run build, then serve static files using Nginx or a simple python server.
> 3. **Database:** PostgreSQL service (optional, if selected in config)."