# Purchase Organizer Application

Comprehensive Project Specification

# 1. High-Level Objective

To create a full-stack, containerized web application that allows users to digitize, manage, and analyze purchase receipts for personal and collaborative expense tracking. The application must be deployable in a private, self-hosted environment via LXC containers.

# 2. Core Concepts & Terminology

- **Purchase:** A single transaction record, which contains metadata (name, date, creator) and a list of associated items.
- **Item:** An individual product or service within a purchase, having the properties: extracted name, friendly name, price, quantity, categories, and payers.
- **User:** Any registered user, using the application.
- **Creator:** The user who originally created a specific purchase.
- **Payer:** The user that made the initial payment for a purchase.
- **Contributor:** Any user assigned to pay an equal part of the item's cost.
- **Friendly Name:** A user-friendly, standardized name for a product (e.g., "Milk") mapped from a more cryptic name found on a receipt (e.g., "FRSHMLK 1.5%").
- **Extracted Name:** The name of an item that was extracted from the receipt (e.g., "FRSHMLK 1.5%").

# 3. Environment

The application is deployable on a Linux server. This is a self-hosted environment that is exposed to the internet. Only configured users have access.

The website is accessible from a common browser either from a stationary computer or from a mobile device.

# 4. User Roles

## 4.1 User

A standard account. Can create their own purchases, view/edit/delete purchases they are a contributor on.

## 4.2 Administrator

A privileged account. Has all the capabilities of a standard User, plus:

- Full read/write access to ALL users, purchases, and data in the system.
- Access to a user management page.
- Ability to add, manage and delete users.

# 5. Pages

## 5.1 Login Page

The primary objective of this page is to provide a clean, secure, and user-friendly gateway to the application. The design will be minimal to focus the user's attention on the single task of logging in.

### 5.1.1 Visual Layout & Design (Desktop View)

The desktop view will use a modern, centered, card-based layout to create a sense of focus and professionalism.

- **Background**: A subtle, light-gray background (#F8F9FA) to provide a neutral canvas.
- **Login Card**: A central card with a soft shadow and rounded corners will contain all the interactive elements. This card will have a maximum width (e.g., 400px) to maintain a tight, organized look on wider screens.
- **Header**:
    - Inside the card, at the top, the application's logo will be displayed.
    - Below the logo, a clear heading <h1> will state "**Log in to your account**".
- **Form Fields**: The form is the core of the page.
    - **User name Input**: A text field clearly labeled "User name".
    - **Password Input**: A password field labeled "Password". A key feature will be an icon on the right side of the field to toggle password visibility, improving user experience by helping prevent typos.
- **Action Button**:
    - The "**Log In**" button will be the primary call-to-action. It will use the **Primary (Deep Blue)** color, span the full width of the card, and have bold, white text to make it highly prominent.

### 5.1.2 Responsiveness (Mobile View)

The design will be fully responsive and optimized for mobile use.

- **Layout**: The card-based layout is maintained, but the card will expand to use the full width of the screen, with minimal horizontal padding.
- **Font Sizes**: Font sizes will be adjusted to ensure optimal legibility on smaller screens.
- **Tap Targets**: Buttons and input fields will adhere to mobile accessibility guidelines, ensuring they are large enough to be easily tapped.

## 5.4 Main Page

The Main Page serves as the primary launchpad for all user activities. Its objective is to provide immediate access to the most common actions, offer a quick overview of recent activity, and establish the application's primary navigation structure. It embodies the "Clarity in Focus" philosophy by presenting information in a clean, organized, and actionable manner.

### 5.4.1 Visual Layout & Design

The layout is clean and structured, with a clear visual hierarchy to guide the user. It introduces the application's persistent navigation header.

- **Persistent Navigation Bar (Header)**: A slim header will be present at the top of this page and all other authenticated pages.
    - **Left Side**: The application logo and a welcoming message, such as "Welcome, [User's Friendly Name]!".
    - **Right Side**: An icon-driven menu. This includes:
        - A "Logout" button.
        - A navigation menu icon (e.g., a "hamburger" icon) that, when clicked, opens a drop-down menu with links to “Home”, "Purchases", "Dashboard" and "Settings".
- **Primary Action Hub**: The main content area will lead with a prominent section dedicated to the application's core function: adding a purchase.
    - This section will feature a large, centric "**Scan Receipt**" button and a “Create Purchase” button, styled with the **Primary (Deep Blue)** color to draw the user's attention.
    - **Interaction**: Clicking this button will open a simple modal dialog with two clear choices:
        - "**Scan Receipt**" button.
        - "**Create Purchase**" button.
    - This approach keeps the main page clean while providing direct access to both purchase creation methods.
- **Recent Activity Feed**: Below the action hub, a section titled "**Recent Purchases**" will provide an at-a-glance summary.
    - This will display the 3-5 most recent purchases in a vertical list of cards.
    - Each **Purchase Card** will be clickable and display essential information:
        - Purchase Name (purchase_name).
        - Purchase Date (purchase_date).
        - Total amount of the purchase.
    - At the bottom of the list, a "View All Purchases" link will navigate the user to the full "Purchase Review Screen."

### 5.4.2 Responsiveness (Mobile View)

The design will be fully responsive to ensure a seamless mobile experience.

- **Navigation Bar**: The welcome message may be hidden to save space. The navigation links will be accessible exclusively through the hamburger menu icon to keep the header clean.

- **Layout**: The content will be organized into a single, scrollable column. The "Primary Action Hub" will remain at the top, followed by the "Recent Activity" feed.
- **Cards**: Purchase cards will expand to the full width of the screen (with appropriate padding) to maximize readability and ensure tap targets are large enough.

## 5.5 Scan Receipt Page

The objective of this page is to provide a simple, step-by-step process for users to upload, prepare, and submit their receipt images for automated data extraction.

### 5.5.1 Visual Layout & Design

The page will maintain the Persistent Navigation Bar for consistency. The layout will guide the user through a clear, linear process from top to bottom.

- **Header**: A prominent <h1> heading at the top will clearly label the page: "**Scan Your Receipt**".
- **Step 1: Upload Area**: The first section will be a large, clearly defined drop zone for images.
    - **Instructional Text**: It will feature text like "Drag & drop receipt images here, or click to browse". It will also specify the accepted formats and limits: "You can upload up to 5 images (JPEG, PNG, HEIC)".
    - **Button**: A large "**Upload Images**" button will be centered within this area for users who prefer clicking to Browse files.
- **Step 2: Image Staging & Preparation**: Once images are uploaded, they will appear in this section, which is initially hidden.
    - **Header**: A subheading <h2> will label this section "**Prepare Your Images**".
    - **Image Grid**: The uploaded images will be displayed as thumbnails in a responsive grid . Each thumbnail will feature:
        - An 'X' icon to remove the image.
        - Icons for **Rotate** and **Crop**, which will open a simple editing tool in a modal window when clicked.
    - **Action Bar**: Below the image grid, a final action bar will appear. It will contain a single, primary action button labeled "**Scan Receipt**". This button will be disabled until at least one image has been uploaded.

## 5.6 Item Review & Editing Page

This page serves as the central hub for managing the details of a single purchase. It has two primary states:

1) **Review Mode**: Entered after a receipt scan, the page is pre-filled with extracted data for the user to verify, correct, and finalize.
2) **Creation Mode**: Entered via the "Create Manually" workflow, the page is a blank canvas for the user to build a purchase record from scratch.

This is a detailed view for managing a single purchase. It displays all purchase details, including any receipt images, and a table of all its items. On this page, users can edit, add, or delete items. It also shows a summary of the total cost and the cost per contributor.

### 5.6.1 Visual Layout & Design

The page will use the Persistent Navigation Bar and a structured, three-section layout to keep the dense information organized and easy to manage.

**Section 1: Purchase Details Header**

- This top section contains the metadata for the entire purchase.
- **Purchase Name**: An input field for the purchase's title (e.g., "Weekly Groceries").
- **Purchase Date**: A date picker, defaulting to the current date.
- **Payer:** A single-select dropdown menu with assignable users. The creator of the purchase is by default selected.
- **Receipt Images**: If created from a scan, small, clickable image previews are displayed here. Clicking an image opens it in a full-size modal view.
- **Global Options**: Checkboxes for "Add Tax" and "Apply discounts" are present here, unticked by default.
- **Summary View**: For already saved purchases, a summary section displays the "total price for the purchase" and the "total price for the purchase per contributor".

**Section 2: Interactive Item List (Card View)**

This section is now a dynamic vertical list of item cards instead of a table.

- **List Controls**: Above the item list, a small control bar will be present:
  - Add Item button: Creates a new, blank item card and adds it to the top of the list.
- **The Item Card**: Each item is represented by a distinct card with a clear structure.
  - **Drag Handle**: On the far left of each card, a "drag handle" icon (e.g., six small dots) will visually indicate that the card can be moved.
  - **Card Layout**: The content within the card is organized for clarity:
    - **Top Row**: The Friendly Name is displayed prominently as an editable text input. The Original Name (if from a scan) is shown below it as non-editable text.
    - **First Middle Row**: A three-column layout for Quantity, Price, and Categories, all as editable inputs.
    - **Second Middle Row:** Three editable dropdown menus. One for each category (Category1, Category2, Category3). Category3 is only activated if Category2 was set. Category2 is only activated if Category1 was set. Each dropdown menu allows making a new entry or selecting one of the already created (and saved in the database) entries. If Category1 was set, then Category2 was set, then Category1 was emptied, then Category2 is also automatically emptied to avoid that Category2 is set without having a Category1.

        - **Bottom Row**: The Contributors multi-select dropdown, allowing for the assignment of users as contributors.
        - **Delete Action**: A "Delete" (trash can) icon is placed on the top-right corner of the card to remove that specific item.
    - **Visual Feedback**: When a user clicks and drags a card, it will lift with a subtle shadow and can be smoothly re-positioned within the list.
    - **User Interaction for Item card List**:
        1. **Re-ordering**: The user can click and hold the drag handle on any card and move it up or down in the list to change its order.
        2. **Editing**: The user can click directly on any field (e.g., Price, Friendly Name) within a card to edit its value.
        3. **Adding**: Clicking the "Add Item" button in the List Controls instantly adds a new card to the list, ready to be filled out.

**Section 3: Actions Footer**

- This final section provides an overview and contains the primary page actions.
- **Action Buttons**: A button group is aligned to the right.
    - Confirm Purchase: The primary action button (solid **Deep Blue**) to save all changes.
    - Delete Purchase: A destructive action button (solid **Alert Red**), visible to users with deletion rights. Clicking it will open a confirmation pop-up.
    - View Logs: A subtle, secondary button that opens a pop-up with the purchase's logs.

## 5.7 Purchase Review Page

The objective of this page is to provide a centralized, searchable, and easy-to-browse archive of all past purchases. It functions like a digital filing cabinet, allowing users to quickly find a specific purchase and navigate to its detailed view. The design prioritizes efficient searching and clear information presentation.

### 5.7.1 Visual Layout & Design

The page will use the Persistent Navigation Bar for consistency. The main content area is structured into a control section and a dynamic content grid.

- **Section 1: Filtering and Controls**
    - A clean control bar is situated at the top of the page, below the main header.
    - **Search Filter**: A prominent search bar will be the central element, as specified. It will have placeholder text like "Search by purchase title..." to guide the user. The search will filter the list to show only purchases with a title containing the search string.
    - **Sorting Controls**: To enhance usability, a "Sort by" dropdown will be included next to the search bar. While the default sort is always newest-to-oldest, this will give users additional options like "Date: Oldest to Newest" or "Total: High to Low".

- **Section 2: Purchase List (Card Grid)**
    - The main area of the page displays all filtered purchases in a responsive card grid.
    - **The Purchase Card**: Each card acts as a high-level summary and a gateway to the detailed view. The entire card is a single clickable element. Each card will contain:
        - **Purchase Name**: The name of the purchase, displayed as the card's title.
        - **Purchase Date**: Displayed clearly below the title.
        - **Total Cost**: A large, bold display of the purchase's final total amount.
        - **Payer username:** The username of the user that is assigned as payer.
        - **Contributor Avatars**: A small section showing the user avatars (or initials) of the contributors involved, giving a quick visual cue of who shared the cost.
        - **Receipt Icon**: A small icon (e.g., a paperclip) to indicate if the purchase has one or more receipt images attached.
    - **Empty State**: If the user has no purchases, or if a search yields no results, the grid area will display a clean, helpful message instead of a blank space. For example: "No purchases found. Ready to scan your first receipt?" with a link to the "Scan Receipt" page.

### 5.7.2 Responsiveness (Mobile View)

- The card grid will seamlessly collapse into a single, vertical column.
- Each purchase card will expand to the full width of the screen (with padding), making the list easy to scroll and tap on a mobile device.
- The filter and sort controls will remain at the top, optimized for a smaller viewport.

## 5.8 Dashboard & Analytics Page

The objective of this page is to transform raw purchase data into meaningful insights. It empowers users to explore their spending habits through a powerful, interactive filtering system and clear data visualizations, as detailed in the specification.

### 5.8.1 Visual Layout & Design

The page will use the Persistent Navigation Bar and a two-column layout on desktop to separate controls from content, providing a classic and effective dashboard experience.

- **Left Column: The Filter Menu**
    - A dedicated sidebar that houses all available filters, allowing users to refine the data they see.
    - **Time Filter (Mandatory)**: At the top, a segmented button group will allow the user to select the time frame: Day, Month, Year, or All Time. A date-picker will appear below, corresponding to the selected time frame. This filter is set to "Year" with the current year selected by default.
    - **Search String Filter**: A search input for filtering by Purchase Title.

    - **Category Filters**: A stack of three dependent dropdown menus for Category 1, Category 2, and Category 3. The categories are independently selectable. Which means that the user could select a Category 2 without selecting a Category 1.
    - **Purchase Item Filter**: A dropdown menu allowing the user to select a specific item they have purchased before.
- **Right Column: Data Visualization**
    - This main area displays the data that corresponds to the active filters.
    - **Summary View (KPI Cards)**: A row of "Key Performance Indicator" cards at the top provides a high-level summary. This will prominently show key figures, such as:
        - **Total Spending**: The primary metric showing how much the user paid within the filtered selection.
        - **Number of Purchases**: The total count of purchases matching the filters.
        - **Average Purchase Cost**: The total spending divided by the number of purchases.
    - **Graphical visualization:** dropdown menu allows selecting one of the two following visualizations of the filtered spendings:
        - **Scatter Plot**: Below the summary cards, a large, interactive scatter plot visualizes the spending over time.
            - **X-Axis**: Time
            - **Y-Axis**: Cost
            - Each point on the plot represents a single purchase. Hovering over a point will reveal a tooltip with the purchase name, date, and exact cost.
        - **Sankey diagram**: A full view of the flow of the filtered spendings where it starts on the left with the total amount and then going to the right a fine-graining of Category 1, Category 2, Category 3 and the items.

### 5.8.2 Responsiveness (Mobile View)

- The two-column layout will adapt to a single-column view.
- The **Filter Menu** will be collapsed by default and accessible via a "Show Filters" button at the top of the page. Tapping this button will slide the filter menu in as an overlay.
- The summary cards and the scatter plot/sankey diagram (whatever is selected to be visible) will stack vertically, with the chart allowing for horizontal scrolling or pinch-to-zoom to explore the data.

## 5.9 Settings Page

The objective of this page is to provide a centralized and clear location for users to manage their profile information, security settings. The design will use distinct sections to group related settings, making the page easy to navigate and understand.

### 5.9.1 Visual Layout & Design

The page will use the Persistent Navigation Bar. The main content area will be organized into separate cards for each category of settings, creating a clean and modular layout.

- **Section 1: Account Information**
    - A card at the top will display static, important user identifiers.
    - **User name**: User name: A non-editable field will clearly display the user's unique User name.
- **Section 3: Security**
    - A card focused on password management.
    - **Header**: <h2> heading: "**Change Password**".
    - **Form**: Three input fields will be provided:
        - Current Password
        - New Password
        - Confirm New Password
    - **Action Button**: A "**Change Password**" button will be at the bottom of the card. The button will be disabled until all fields are filled and the new passwords match and meet the length requirements (10-30 characters).

### 5.9.2 Responsiveness (Mobile View)

- The layout will collapse into a single column.
- Each settings section (Account Information, Profile, Security, etc.) will be presented as a full-width card, stacked vertically, making it easy to scroll through and manage on a mobile device.

## 5.10 User Management Page

This page is **exclusively for administrators**.The objective of this page is to provide administrators with a secure, clear, and efficient interface to manage all user accounts in the system. Given the powerful capabilities of this page, the design prioritizes clarity, deliberate actions, and confirmation steps to prevent accidental changes. This page is only visible to and accessible by users with the "Administrator" role.

### 5.10.1 Visual Layout & Design

The page will use the Persistent Navigation Bar. The layout will be structured to allow for both a high-level overview and detailed management of individual users.

- **Section 1: Page Header and Global Actions**
    - A prominent <h1> heading will state: "**User Management**".
    - **Add User Button**: A primary "**+ Add User**" button will be present. Clicking this will open a modal window for creating a new account.
- **Section 2: User Grid (Card View)**
    - The main area of the page will display all users in a responsive card grid.
    - The User Card: Each card is a self-contained unit representing a single user. It will clearly display:
        - User's Name: Displayed prominently as the card's title.
        - Role Tag: A distinct visual tag (e.g., a colored pill-shaped badge) indicating the user's role ("Administrator" or "User").

        - Manage Button: A clear "Manage" button at the bottom of the card, which opens the detailed management modal for that user.

### 5.10.2 Modal Windows for Specific Actions

To prevent clutter and ensure actions are deliberate, key functions will be handled in modal dialogs.

- **"Add User" Modal**:
    - This modal allows administrators to create a new account by providing a unique user name and a preliminary password.
- **"User Details & Management" Modal**:
    - Clicking the "Manage" button next to a user in the list will open a comprehensive modal dedicated to that specific user. This keeps the main list clean and focuses the admin's attention.
    - The modal header will display the user's name and email.
    - **Administrator Rights**: A toggle switch or a button to "Make Administrator" or "Revoke Administrator Rights". The page will update to show the new button state after the action is confirmed.
    - **Override Password**: A section with a single "New Preliminary Password" input field and a button to "Set New Password".
    - **Delete User**: A prominent, destructive-style (**Alert Red**) "**Delete User Account**" button.

# 6. Workflows

## 6.6 Administrator rights set up

- The authenticated administrator starts at any page.
- The Front-end provides a button to enter the user management page.
- The authenticated administrator clicks that button.
- The Front-end redirects the authenticated administrator to the user management page.
- The Front-end displays all registered users.
- The authenticated administrator selects one user from the card view.
    - If selected user is already administrator:
        - The Front-end displays the “Revoke administrator rights” button.
        - The authenticated administrator clicks the “Revoke administrator rights” button.
        - The Front-end sends the request to the Back-end.
        - The selected user is automatically logged out.
        - The Back-end sets the ‘Administrator’ flag to false for the selected user.
        - The Back-end sends the confirmation to the Front-end.
        - The Front-end updates the view by hiding the “Revoke administrator rights” button and displaying the “Make administrator” button.
    - If selected user is not yet administrator:

        - The Front-end displays the “Make administrator” button.
        - The authenticated administrator clicks the “Make administrator” button.
        - The Front-end sends the request to the Back-end.
        - The selected user is automatically logged out.
        - The Back-end sets the ‘Administrator’ flag to true for the selected user.
        - The Back-end sends the confirmation to the Front-end.
        - The Front-end updates the view by hiding the “Make administrator” button and displaying the “Revoke administrator rights” button.

## 6.8 Login

- The non-authenticated user starts at the login page.
- The Front-end provides a user name input field, a password field and a “Login” button.
- The non-authenticated user enters his user name, his password and clicks the “Login” button.
- The Front-end sends the request to the Back-end
- The Back-end checks the existence of the account
    - If account does not exist:
        - The Back-end informs the Front-end about the denial.
        - The Front-end displays an “Account not known, not validated, or wrong password” notification.
        - The non-authenticated user restarts by putting in again the login information.
    - If account exists:
        - The Back-end checks if the password is correct.
            - If the password is not correct:
                - The Back-end informs the Front-end about the denial.
                - The Front-end displays an “Account not known, not validated, or wrong password” notification.
                - The non-authenticated user restarts by putting in again the login information.
            - If the password is correct:
                - The Back-end informs the Front-end about the success
                - The user is now considered as an authenticated user.
                - The Front-end redirects the authenticated user to the main page.

## 6.9 Logout

- The authenticated user starts at any page.
- The Front-end provides a “Logout” button.
- The authenticated user clicks the “Logout” button.
- The Front-end sends the request to the Back-end.
- The Back-end sets the user to logged out.
- The Front-end sets the user to logged out.

- The Front-end redirects the user to the Login page.

## 6.10 Create a Purchase from a receipt

- The authenticated user starts from the main page.
- The Front-end provides a “Scan receipt” button.
- The authenticated user clicks the “Scan receipt” button.
- The Front-end opens a file picker that allows the authenticated user to select up to 5 (JPEG, PNG or HEIC) images.
- The Front-end provides the means to rotate and crop each image.
- The authenticated user rotates the image and crops them
- The Front-end provides an “Upload image(s)” button.
- The authenticated user clicks the “Upload image(s)” button.
- The Front-end sends the image(s) to the Back-end.
- The Front-end displays a “Processing ongoing” notification to the authenticated user.
- The Back-end calls an API for each image.
- The API returns the extracted items from the image.
- The Back-end collects and merges all items from all images.
- The Back-end applies the [Friendly Name Mapping] Logic for each item.
- The Back-end sends the results to the Front-end.
- The Front-end displays each item and its parameters in a list.
- The Front-end displays a name input field for the purchase.
- The Front-end displays a date input field for the purchase.
- The Front-end displays an “Add Tax” check box which is unticked by default.
- The Front-end displays a “Apply discounts” check box which is unticked by default.
- The Front-end displays a “Confirm purchase” button.
- The Front-end displays a “Delete” button.
- The authenticated user enters a name for the purchase which sets by default the current date.
- The authenticated user modifies the date for the purchase, if necessary.
- The authenticated user updates the parameters of the items, if necessary
- The authenticated user clicks the “Confirm purchase” button.
- The Front-end checks if all parameters are conform
    - If not all parameters are conform on Front-end:
        - The Front-end highlights parameters that are not conform.
        - The Front-end displays a notification about the non-conformity.
        - The authenticated user has to correct all parameters before proceeding.
    - If all parameters are conform on Front-end:
        - The Front-end sends the request to the Back-end.
        - The Back-end checks the conformity of all parameters.
            - If not all parameters conform on Back-end:
                - The Back-end sends a notification to the Front-end.
                - The Front-end highlights parameters that do not conform.
                - The Front-end displays a notification about the non-conformity.

        - The authenticated user has to correct all parameters before proceeding.
    - If all parameters conform on Back-end:
        - The Back-end applies the [Friendly Name Storing] Logic to store Friendly Names for each item.
        - The Back-end saves the images in the Cloud storage.
        - The Back-end stores the purchase in the database.
        - The Back-end informs the Front-end about the completion.
        - The Front-end displays a notification that the purchase was saved.
        - The Front-end displays a summary of the purchase on top of the screen.

## 6.11 Manually create a Purchase

- The authenticated user starts from the main page.
- The Front-end provides a “Create Purchase” button.
- The authenticated user clicks the “Create Purchase” button.
- The Front-end displays a name input field for the purchase.
- The Front-end displays a date input field for the purchase.
- The Front-end displays an “Add item” button.
- The Front-end displays an “Add Tax” check box which is unticked by default.
- The Front-end displays a “Apply discounts” check box which is unticked by default.
- The Front-end displays a “Confirm purchase” button.
- The Front-end displays a “Delete” button.
- The Front-end displays each created item and its parameters in a list.
- The authenticated user adds and modifies up to 100 items per purchase.
- The authenticated user provides a name for the purchase.
- The authenticated user modifies the date of the purchase, if necessary.
- The authenticated user clicks on “Confirm purchase” button.
- The Front-end checks if all parameters are conform
    - If not all parameters are conform on Front-end:
        - The Front-end highlights parameters that are not conform.
        - The Front-end displays a notification about the non-conformity.
        - The authenticated user has to correct all parameters before proceeding.
    - If all parameters are conform on Front-end:
        - The Front-end sends the request to the Back-end.
        - The Back-end checks the conformity of all parameters.
            - If not all parameters conform on Back-end:
                - The Back-end sends a notification to the Front-end.
                - The Front-end highlights parameters that do not conform.
                - The Front-end displays a notification about the non-conformity.

        - The authenticated user has to correct all parameters before proceeding.
    - If all parameters conform on Back-end:
        - The Back-end applies the [Friendly Name Storing] Logic to store Friendly Names for each item.
        - The Back-end saves the images in the Cloud storage.
        - The Back-end stores the purchase in the database.
        - The Back-end informs the Front-end about the completion.
        - The Front-end displays a notification that the purchase was saved.
        - The Front-end displays a summary of the purchase on top of the screen.

## 6.12 Delete a Purchase

- The authenticated user triggers this process from any step of either a manual purchase creation, a purchase creation by scanning a receipt or a purchase modification.
- The Front-end displays a “Delete” button.
- The authenticated user clicks the “Delete” button.
- The Front-end displays a pop-up if the authenticated user really wants to cancel which includes that unsaved data is lost.
    - If the authenticated user confirms the deletion:
        - The Front-end informs the Back-end.
        - The Back-end checks if any entries for this purchase exist in the database.
            - If entries exist in the database:
                - The Back-end checks if the requestor is the creator of the purchase or a contributor.
                - If the requestor is the creator or a contributor of the purchase:
                - The Back-end deletes any images linked to this purchase in the cloud storage.
                - The Back-end deletes the entries in the database linked to this purchase.
                - The Back-end informs the Front-end about the deletion completion.
                - The Front-end redirects the authenticated user to the main-page.
                - The Front-end removes any traces of the deleted purchase.
                - If the requestor is not the creator or a contributor of the purchase:
                - The Back-end rejects the deletion request.
                - The Back-end informs the Front-end about the rejection of the request.

- The Front-end displays a message to the end requestor.
- If no entries exist in the database:
  - The Back-end informs the Front-end about the deletion completion.
  - The Front-end redirects the authenticated user to the main-page.
  - The Front-end removes any traces of the deleted purchase.
- If the authenticated user rejects the deletion:
  - The pop-up closes.
  - The authenticated user remains on the page.

### 6.15 Display purchase statistic

- The authenticated user starts from any page.
- The Front-end displays a “Statistics” Button.
- The Authenticated User clicks the “Statistics” Button and the user is redirected to the Statistics page.
- The Front-End displays a filter menu.
- The authenticated user applies the filters he wants.
- The Front-end sends the request to the Back-end
- The Back-end retrieves the data from the database.
- The Back-end returns the data to the Front-End
- The Front-end displays a summary view.
- The Front-end uses the retrieved data to update the summary view.
- The authenticated user can do further filtering adaptations.

## 7. Permissions & Access Control

- A standard **User** can view,edit and delete a purchase if they are the original `creator_user_id` OR if their `user_id` appears in the `contributors` or `payers` table for at least one item within that purchase.
- An **Administrator** bypasses these checks and has universal access. Admin actions on other users' purchases must still be logged with the admin's identity.

## 8. Contributors & Sharing System

When assigning a contributor and  a payer, the UI will show a dropdown list of all registered users on the system.

## 9. Backend Processing & Logic

### 9.1 Friendly Name Mapping Logic (Pseudocode)

FUNCTION get_friendly_name(original_name, user_id):

```
  substrings = generate_all_substrings(original_name.lower())
 //Example: “Milk Frsh Alpine” will be decomposed to [milk, frsh, alpine, milk frsh, milk frsh aline, frsh alpine]

  // Step 1: Check user-specific mappings
  user_mappings = find_mappings_for(substrings, created_by=user_id)
  IF user_mappings is not empty:
    winner = find_most_common_friendly_name(user_mappings)
    IF winner is not a tie:
      RETURN winner

  // Step 2: Check global mappings
  global_mappings = find_mappings_for(substrings, created_by=ANY_USER)
  IF global_mappings is not empty:
    winner = find_most_common_friendly_name(global_mappings)
    IF winner is not a tie:
      RETURN winner

  // Step 3: Default to original
  RETURN original_name
```

## 9.2 Friendly Name Setting Logic (Pseudocode)

```
FUNCTION set_friendly_name(original_name, friendly_name user_id):
  substrings = generate_all_substrings(original_name.lower())
 //Example: “Milk Frsh Alpine” will be decomposed to [milk, frsh, alpine, milk frsh, milk frsh alpine, frsh alpine]

 // Step 1: Check user-specific mappings
 for each substring in substring:
   insert_or_update_database(user_id,substring,friendly_name)
```

Typos and wrong extractions will not be automatically corrected. Instead, they will just be incorporated in the friendly name mapping. Which means that, if a first purchase identified “Milk Frsh Alpine” this will be mapped to [milk, frsh, alpine, milk frsh, milk frsh alpine, frsh alpine]. But if in a second purchase the API extracts “Milk Frsh Aline” then the database will be appended with [aline, milk frsh aline, frsh aline]. But the entries from the first purchase remain in the database.

## 9.3 Cloud Storage

Cloud storage shall use an abstraction layer to assure a simple change between multiple cloud storage providers and also a local storage solution. The source code for the abstraction layer template looks like this:

```
class StorageInterface:
        """A generic contract for all storage operations."""
```

```
    def upload_file(self, source_path, destination_name):
        """Uploads & updates a file to the storage."""
        raise NotImplementedError

    def delete_file(self, file_name):
        """Deletes a file from the storage."""
        raise NotImplementedError

    def get_file_url(self, file_name):
        """Gets a publicly accessible URL for a file."""
        raise NotImplementedError
```

## 9.3 Receipt extract

Extracting the items from the receipt shall be exactly done as described here. The quality of the receipt images are often not good. This leads to some false extracts. this can be reduced by applying some pre-processing on the image. The exact code to be used for the preprocessing is this:

```
def preprocess_image(image: np.ndarray, threshold1: int, threshold2: int) -> np.ndarray:
    filtered = image.copy()
    gray_image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    gray_image = cv2.blur(gray_image,(3,3))
    contour_thresh = cv2.threshold(gray_image, 0, 255,
cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]

    # Invert
    contour_thresh = 255 - contour_thresh

    contours = cv2.findContours(contour_thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    contours = contours[0] if len(contours) == 2 else contours[1]

    for contour in contours:
        if cv2.contourArea(contour) > threshold2:
            cv2.drawContours(filtered, [contour], -1, color=(255, 255, 255),
thickness=cv2.FILLED)

     filtered_gray = cv2.cvtColor(filtered,cv2.COLOR_BGR2GRAY)
    filtered_image = cv2.threshold(filtered_gray, threshold1, 255,
cv2.THRESH_BINARY)[1]

    return filtered_image

def extract_information(image: np.ndarray) -> str:
    return pytesseract.image_to_string(image)
```

```
def analyze_information(extracted_text: str) -> Dict:
        api_key = os.environ["MISTRAL_API_KEY"]
        model = "mistral-small-latest"

        client = Mistral(api_key=api_key)

        system_message ="You are an expert extraction algorithm. The text is a shopping receipt. Only extract the purchased items including their price from the text. If you do not know the value of an attribute asked to extract, you may omit the attribute's value. ignore the total price. Only output the data with no further explanation or comment. use an empty string instead of null. split the items by a newline."

        messages = [{'role':"system",'content': system_message},{'role':"user",'content': extracted_text}]

        return client.chat.complete(model=model,messages=messages)

async def process_image(image_location : str) -> Dict:
        #Step 0: Load image
        TODO [Please write this code to load the image into the variable
        opencv_image = …

        threshold1 = 100
        threshold2 = 200

        # Step 1: Pre-process the image
        preprocessed_image = await run_in_threadpool(apply_threshold, opencv_image, threshold1, threshold2)

        # Step 2: Extract text from the processed image
        extracted_text = await run_in_threadpool(extract_text, preprocessed_image)

        # Step 3: Analyze the extracted text
        analysis_result = await run_in_threadpool(analyze_text, extracted_text)
```

## 9.4 Administrator notification

In some cases, the administrator has to validate certain adaptations to user accounts.
The administrator is not actively informed about that. Only when opening or refreshing the user management page, he will see the open requests.

## 9.4 Database calls

The database should be encapsulated by a database.py module. This module limits the access to the database by providing functions. The list of functions to be implemented is here:

### 9.4.1 User Management Functions

- create_user(user_name): Creates a new user with a given user name.
- get_user_by_name(user_name): Retrieves a user's complete data based on their user name.
- get_user_by_id(user_id): Retrieves a user's complete data based on their user_id.
- get_user_id_by_name(user_name): Retrieves a user's ID based on their user_name.
- update_user_password(user_name, new_password_hash): Updates the password_hash for a specified user.
- delete_user(user_id): Removes a user and all their associated, non-shared data from the database.
- set_administrator_rights(user_id, is_admin): Grants or revokes administrator privileges for a user.
- get_all_users(): Retrieves a list of all registered users.

### 9.4.2 Purchase Management Functions

- create_purchase(creator_user_id, payer_user_id, purchase_name, purchase_date, tax_is_added, discount_is_applied): Creates a new purchase record in the database.
- get_purchase_by_id(purchase_id): Retrieves all details for a single purchase.
- update_purchase(purchase_id, purchase_name, purchase_date, payer_user_id, tax_is_added, discount_is_applied): Modifies the metadata of an existing purchase.
- delete_purchase(purchase_id): Deletes a purchase and its associated items and logs.
- get_recent_purchases(user_id, limit=5): Retrieves the most recent purchases for a specific user.
- get_purchases_for_user(user_id, filters=None, sort_by=None): Fetches all purchases where the user is either the creator or a contributor, with optional filtering and sorting.
- check_purchase_existence(purchase_id): Checks if a purchase with the given ID exists in the database.

### 9.4.3 Item and Contributor Functions

- add_item_to_purchase(purchase_id, original_name, friendly_name, quantity, price, category_level_1, category_level_2, category_level_3): Adds a new item to an existing purchase.
- update_item(item_id, friendly_name, quantity, price, category_level_1, category_level_2, category_level_3): Modifies the details of an item.
- delete_item(item_id): Removes an item from a purchase.
- get_items_for_purchase(purchase_id): Retrieves all items associated with a particular purchase.
- add_contributor_to_item(item_id, user_id): Assigns a user as a contributor to a specific item.
- remove_contributor_from_item(item_id, user_id): Removes a user's contribution from an item.
- get_contributors_for_item(item_id): Retrieves all contributors for a given item.
- is_user_sole_contributor_and_payer(user_id): Checks which purchases have the specified user as the only payer and contributor, for deletion purposes.

### 9.4.4 Category and Friendly Name Functions

- create_category(user_id, category_name, level): Adds a new category to the database, associated with a user.
- get_all_categories(): Retrieves all unique category names for populating dropdown menus.
- find_friendly_name_mappings(substrings, user_id=None): Searches for friendly name mappings, either user-specific or global.
- create_friendly_name_mapping(user_id, substring, friendly_name): Creates or updates a mapping between an extracted substring and a friendly name.

### 9.4.6 Logging Functions

- create_purchase_log(purchase_id, user_id, log_message): Adds a log entry for an action performed on a purchase.
- get_logs_for_purchase(purchase_id): Retrieves all log messages associated with a given purchase.

# 10. Front-End Logic

## 10.1 Contribution splitting

When multiple contributors are selected, the distribution is always equal (for simplicity). This means, if an item cost 1.00€ then:
3 Contributors: each of them contributes with 0.34€
4 Contributors: each of them contributes with 0.25€
5 Contributors: each of them contributes with 0.20€

## 10.2: Category assignment

At the creation of an item, categories must be set in order (first Category1, then Category2, then Category3) freely chosen from the available dropdown menu or by writing a new category name. In that stage an underlying category must not exist when the category above was not set. This means that Category3 is automatically considered as empty when Category2 is empty. And Category2 is automatically considered as empty when Category1 is empty. A user can choose how many categories he sets (from zero to all three), but he must respect the order.

When filtering purchases in the Analytic page, a user can choose an underlying category without selecting the upper category. He could select a Category3 without selecting a Category2 or Category1. This flexibility is essential because the Category3 could exist for multiple Category2.

# 11. Database Structure (PostgreSQL/SQLite Detailed)

- **`users`**:
  - `user_id` (SERIAL PRIMARY KEY), `name`(VARCHAR(30), UNIQUE), `password_hash` (VARCHAR(255)), `administrator` (BOOLEAN).
- **`purchases`**:
  - `purchase_id` (SERIAL PRIMARY KEY), `creator_user_id` (INTEGER, FOREIGN KEY to `users.user_id`), `payer_user_id` (INTEGER, FOREIGN KEY to `users.user_id`), `purchase_name` (VARCHAR(255)), `purchase_date` (DATE)), `tax_is_added` (BOOLEAN) , `discount_is_applied` (BOOLEAN).
- **`items`**:
  - `item_id` (SERIAL PRIMARY KEY), `purchase_id` (INTEGER, FOREIGN KEY to `purchases.purchase_id`), `original_name` (TEXT), `category_level_1` (TEXT), `category_level_2` (TEXT), `category_level_3` (TEXT), `friendly_name` (TEXT), `quantity` (INTEGER), `price` (DECIMAL(10, 2)), `discount` (DECIMAL(10, 2)), `tax_rate` (DECIMAL(5, 2)).
- **`contributors`**:
  - `contributor_id` (SERIAL PRIMARY KEY), `item_id` (INTEGER, FOREIGN KEY to `items.item_id`), `user_id` (INTEGER, FOREIGN KEY to `users.user_id`).
- **`categories`**:
  - `category_id` (SERIAL PRIMARY KEY), `user_id` (INTEGER, FOREIGN KEY to `users.user_id`), `category_name` (VARCHAR(30)), `level` (INTEGER).
- **`purchase_logs`**:
  - `log_id` (SERIAL PRIMARY KEY), `purchase_id` (INTEGER), `user_id` (INTEGER), `timestamp` (TIMESTAMP), `log_message` (TEXT).
- **`friendly_names`**:
  - `friendly_name_id` (SERIAL PRIMARY KEY), `user_id` (INTEGER), `substring` (TEXT), `friendly_name`(TEXT).
- **`password_reset_tokens`:**
  - `token_id` (SERIAL PRIMARY KEY)
  - `user_id` (INTEGER, FOREIGN KEY to `users.user_id`, ON DELETE CASCADE)
  - `token_hash` (VARCHAR(255), UNIQUE, NOT NULL)
  - `expires_at` (TIMESTAMP, NOT NULL)
-

# 12. Technology Stack & Architecture

- **Architecture:** A containerized, API-driven application.
- **Containerization:** LXC containers, managed via Proxmox.

- **Backend:** Python / FastAPI / JWT, SQLAlchemy / PyYAML.
- **Frontend:** HTML, Bootstrap, Javascript, React / Vite / Tailwind CSS / Recharts / Zustand.
- **Database:** PostgreSQL (for scalability) or SQLite (for simplicity).

# 13. Configuration & Deployment

**Configuration File (`config.yaml`):** All system settings are managed in this file. A sample structure:

```
# --- System Configuration ---
database:
  # 'sqlite' or 'postgresql'
  type: 'sqlite'
  # Path for SQLite file (used if type is 'sqlite')
  path: '/app/data/database.db'
  # Connection string for PostgreSQL (used if type is 'postgresql')
  url: 'postgresql://user:password@db:5432/receiptdb'

storage:
# Maximum size for a single image upload in megabytes.
 max_upload_size_mb: 25

local: # Settings for when provider is 'local'
image_path: '/app/images'

# Credentials would be handled securely, e.g., via environment variables
#mistral_api_key: 'ENTER KEY HERE'



Receipt extract:
# An external service that is called with the following parameter. Any relevant parameters would be added here.
threshold1: #Service specific parameter
threshold2: #Service specific parameter
# Credentials would be handled securely, e.g., via environment variables
```

- **Deployment Process:** A `deploy.sh` script will automate the process: pull the latest code from Git, start and enable the service to run the application.

# 14. Platform & Future-Proofing

- **Web First:** The primary application is a responsive web app accessible on desktop and mobile browsers.
- **Android and IOS App:** The applications shall be designed in a way that it can be embedded in a web view to be installed as an Android or IOS App, without a re-development.

# 15. Design

The core philosophy is **"Clarity in Focus."** The application manages detailed financial data, so the design must prioritize legibility, intuitive navigation, and efficiency. The goal is to reduce cognitive load on the user, allowing them to process information and complete tasks quickly and without confusion. The aesthetic will be clean, modern, and data-centric, avoiding unnecessary clutter.

## 15.1 Color Palette

The color scheme is chosen to be professional, accessible, and easy on the eyes during prolonged use.

- **Primary Color (Deep Blue)**: Used for key interactive elements like primary buttons ("Confirm Purchase," "Login"), navigation headers, and active links. This color inspires trust and stability.
- **Secondary Color (Light Gray)**: Used for backgrounds of pages, cards, and input fields. It provides a neutral, non-distracting canvas for the content.
- **Accent Color (Vibrant Green)**: Used for success notifications (e.g., "Purchase saved" ), positive financial indicators, and confirmation actions.
- **Warning/Error Color (Alert Red)**: Used for delete confirmations , error messages (e.g., "non-conformity of the password" ), and highlighting non-conformant fields.
- **Text Color (Charcoal Gray)**: A dark gray is used for body text and labels instead of pure black to reduce eye strain.

## 15.2 Typography

The typography will be clean, modern, and highly legible.

- **Font Family**: A sans-serif font like **Inter** or **Lato** will be used for all text. These fonts are known for their excellent readability on screens.
- **Headings**: h1, h2, h3 will be semi-bold to establish a clear visual hierarchy.
- **Body Text**: Regular weight for all descriptive text and data within tables.
- **Labels & Helper Text**: A slightly smaller font size for input field labels and helper text to distinguish them from primary content.

## 15.3 Layout and Structure

The application will use a responsive, card-based layout that adapts seamlessly from desktop to mobile devices.

- **Main Container**: A fixed-width main container will be centered on large screens, with vertical padding to create breathing room. On mobile, it will expand to fill the screen width.

- **Navigation**: A persistent sidebar (on desktop) or a hamburger menu (on mobile) will provide access to the **Home**, **Purchases**, **Dashboard**, and **Settings**. The **User Management** link will appear here only for administrators.
- **Card-Based UI**: Purchases , and dashboard widgets will be presented in cards. This modular approach keeps information organized and visually separated. Each card will have consistent padding, a subtle border, and a light box-shadow to lift it off the background.
- **Responsive Grids**: A flexible grid system will be used to arrange content. For example, on the Dashboard, filter options might occupy a sidebar on desktop but collapse into an accordion or a modal on mobile.

## 15.4 Component Design

A consistent set of UI components will be used throughout the application.

- **Buttons**:
    - **Primary**: Solid background (Deep Blue) with white text for the main call-to-action on a page (e.g., "Confirm purchase" ).
    - **Secondary**: Outline-style (Deep Blue border, transparent background) for less critical actions (e.g., "Add manually" ).
    - **Destructive**: Solid background (Alert Red) for actions like "Delete account".
- **Forms and Inputs**:
    - Input fields will have a light gray background, clear labels positioned above the field, and a blue border on focus.
    - Validation errors will be displayed in red text directly below the respective field.
- **Modals (Pop-ups)**:
    - A semi-transparent overlay will cover the background page to focus the user's attention. Modals will be used for confirmations, such as deleting a purchase. They will always contain clear "Confirm" and "Cancel" actions.
- **Notifications**:
    - Toast notifications will slide in from a corner of the screen for non-blocking feedback, such as "We have sent instructions to your email" or "Purchase was saved". They will be color-coded (Green for success, Red for error).