Expense Management System

Use Case Document

Actors

1. **Primary Actor**
    - **Registered User** – normal end user managing their expenses.
2. **Supporting / Secondary Actors**
    - **Notification Service** – sends emails/push/SMS/in-app alerts.
    - **Exchange Rate Service** – provides currency conversion rates (internal or external API).
    - **Storage Service** – handles file uploads for attachments/receipts.

UC-01: Register User Account

Use Case ID: UC-01
Use Case Name: Register User Account
Primary Actor: Visitor (Unregistered User)
Level: User-goal
Scope: Expense Management System

1. Brief Description

A new user registers an account by providing basic details (name, email, password, optionally preferred currency). The system creates a user profile and sets up initial defaults (e.g., default categories, default currency).

2. Stakeholders & Interests

- **User:** Wants a quick, secure registration with minimal information, without duplicate accounts or email conflicts.
- **System Owner:** Wants to ensure unique emails, valid data, and secure password storage.

3. Preconditions

- The user is **not logged in**.
- User’s email is not yet registered in the system.

4. Postconditions

- **Success:**
  - A new `users` record is created.
  - Default categories (Food, Transport, Income, etc.) may be cloned for that user or linked as system defaults.
  - User's preferred currency is set (or defaulted to PKR if not provided).
- **Failure:**
  - No user record is created; error messages are shown.

5. Trigger

- The visitor clicks **"Sign Up / Register"** and submits the registration form.

6. Main Success Scenario (Basic Flow)

1. Visitor opens the registration page.
2. System displays registration form (full name, email, password, confirm password, preferred currency).
3. Visitor fills in all required fields.
4. Visitor submits the form.
5. System validates:
   - Required fields present.
   - Email format is valid.
   - Password meets strength rules.
6. System checks that email is **not already in use**.
7. System hashes the password and creates a new `users` record.
8. System creates initial setup:
   - Sets `status = ACTIVE`.
   - Sets `preferred_currency` (user-selected or default).
   - Optionally creates default categories for the user.
9. System confirms successful registration and may auto-login or redirect to login page.
10. Visitor becomes a **Registered User**.

7. Extensions (Alternative / Exception Flows)

- **4a. Mandatory fields missing**
  - 4a1. System highlights missing fields.
  - 4a2. System shows error message.
  - 4a3. Visitor corrects and resubmits form (return to Step 4).
- **5a. Password too weak**
  - 5a1. System displays password strength requirements.
  - 5a2. Visitor adjusts password and resubmits (back to Step 4).
- **6a. Email already exists**
  - 6a1. System shows: "Email already registered, please log in or reset password."
  - 6a2. Use case ends in failure; user may go to login page.

- **7a. System error saving data**
  - 7a1. System logs the error.
  - 7a2. System shows generic error message.
  - 7a3. Use case ends in failure.

8. Special Requirements

- All passwords must be stored hashed (e.g., BCrypt) and never in plain text.
- Registration form and API must use HTTPS.
- Optional email verification flow can be added as an extension.

UC-02: Log In and Manage User Session

Use Case ID: UC-02
Primary Actor: Registered User
Level: User-goal

1. Brief Description

User logs into the system with email and password to access their personal expense data.

2. Preconditions

- User has a registered account (UC-01).
- User account status is not `DELETED`.

3. Postconditions

- **Success:** Valid session/JWT is issued and user is redirected to dashboard.
- **Failure:** User remains unauthenticated; no sensitive data is shown.

4. Main Success Scenario

1. User opens login page.
2. System shows login form (email + password).
3. User enters credentials and submits.
4. System validates input.
5. System verifies:
   - Email exists.
   - Password hash matches stored hash.
6. System checks user `status` (e.g., ACTIVE).
7. System creates an authenticated session/JWT.
8. System redirects user to **Dashboard** (UC-08).

5. Extensions

- **5a. Invalid email or password**
  - 5a1. System shows error: "Invalid email/password."
  - 5a2. User may retry (back to Step 3).
- **6a. Account is LOCKED/INACTIVE**
  - 6a1. System shows appropriate message ("Account locked, contact support").
  - 6a2. Use case ends in failure.
- **7a. Session creation fails**
  - 7a1. System logs error.
  - 7a2. System shows generic error and aborts.

UC-03: Create and Manage Accounts (Wallets/Bank/Card)

Use Case ID: UC-03
Primary Actor: Registered User
Level: User-goal

1. Brief Description

User creates, edits, or archives financial accounts (cash, bank accounts, credit cards, mobile wallets). These accounts will be used when recording transactions.

2. Preconditions

- User is logged in (UC-02).
- At least one default account (e.g., Cash) may already exist (optional).

3. Postconditions

- **Success:**
  - New `accounts` record is created/updated/archived.
  - Account's `current_balance` is initialized (if new).
- **Failure:**
  - No change to account data.

4. Trigger

- User selects **"Accounts"** from navigation and chooses "Add / Edit / Archive Account".

5. Main Success Scenario: Create New Account

1. User opens Accounts screen.
2. System lists existing accounts with balances.
3. User clicks **“Add Account”**.
4. System displays account creation form:
   - Name
   - Type (Cash, Bank, Credit Card, Mobile Wallet, Other)
   - Currency
   - Initial Balance
5. User fills and submits the form.
6. System validates inputs (required fields, numeric balance).
7. System creates a new `accounts` record:
   - `initial_balance` set.
   - `current_balance` = `initial_balance`.
8. System shows updated list of accounts with new account added.

6. Extensions

- **6a. Validation error (missing or invalid data)**
  - 6a1. System highlights fields and shows errors.
  - 6a2. User corrects data and resubmits (back to Step 5).
- **Existing Account Update (Edit)**
  - E1. User selects an account and chooses “Edit”.
  - E2. System shows account edit form (name, type, currency – maybe restricted if has transactions).
  - E3. User updates and submits.
  - E4. System validates and updates the `accounts` record.
  - E5. System recalculates any derived values if necessary.
- **Archiving Account**
  - A1. User selects an account and chooses “Archive”.
  - A2. System checks if account has non-zero balance.
    - If non-zero, system warns user and may require confirmation or a transfer.
  - A3. On confirmation, system sets `is_archived = 1`.
  - A4. Account no longer appears in default selection lists but remains for historical data.

UC-04: Record Expense / Income Transaction

Use Case ID: UC-04
Primary Actor: Registered User
Level: User-goal

1. Brief Description

User records an expense or income transaction against a specific account and category, optionally linking merchant, tags, and attachments.

2. Preconditions

- User is logged in.
- At least one **Account** exists (UC-03).
- Relevant **Categories** exist (default or user-defined).

3. Postconditions

- **Success:**
  - A new `transactions` record is created (type EXPENSE or INCOME).
  - `accounts.current_balance` is updated accordingly.
- **Failure:**
  - No transaction saved; balances unchanged.

4. Trigger

- User clicks **"Add Transaction"** (or "Add Expense" / "Add Income") from dashboard or account view.

5. Main Success Scenario: Record Expense

1. User chooses **"Add Expense"**.
2. System displays transaction form:
   - Account
   - Category (type = EXPENSE)
   - Amount
   - Currency (default = account currency)
   - Date (default = today)
   - Time (optional)
   - Payment Method (optional)
   - Merchant (optional)
   - Tags (optional)
   - Description (optional)
3. User fills the form and submits.
4. System validates:
   - Required fields present.
   - Amount > 0.

    - Account and category belong to this user.
5. If currency ≠ account currency and base currency reporting is enabled:
    - System calculates or fetches `exchange_rate_to_base` and `base_amount`.
6. System creates a new row in `transactions`:
    - `type = 'EXPENSE'`.
    - Sets user/account/category references.
    - Sets `status = CLEARED` (or PENDING if user chose so).
7. System **decreases** `accounts.current_balance` by the expense amount.
8. System returns success message and shows updated account balance and recent transactions.

6. Main Success Scenario: Record Income

Same as above, except:

- Step 1: User chooses **"Add Income"**.
- Step 6: `type = 'INCOME'`.
- Step 7: System **increases** `accounts.current_balance`.

7. Extensions

- **5a. Currency conversion unavailable**
    - 5a1. System cannot fetch live exchange rate.
    - 5a2. System prompts user to **manually enter** rate.
    - 5a3. User enters rate; system calculates `base_amount` and continues.
- **4a. Invalid amount or missing fields**
    - 4a1. System shows validation errors.
    - 4a2. User corrects and resubmits.
- **7a. Insufficient Funds for Expense (optional rule)**
    - 7a1. If system enforces non-negative balances, it detects negative `current_balance`.
    - 7a2. System warns user and asks:
        - "Allow negative balance?" or "Change account?"
    - 7a3. User decides:
        - Allow negative → continue.
        - Cancel → use case ends with no transaction.
- **6b. Mark Transaction as Pending**
    - 6b1. User marks transaction as pending (e.g., card authorization).
    - 6b2. System sets `status = PENDING` and **does not** affect `current_balance` until cleared.

UC-05: Transfer Funds Between Accounts

Use Case ID: UC-05
Primary Actor: Registered User
Level: User-goal

1. Brief Description

User transfers money from one account to another (e.g., bank → cash). The system creates two linked transactions and adjusts both balances.

2. Preconditions

- User is logged in.
- At least two accounts exist for that user.

3. Postconditions

- **Success:**
  - One `transactions` record with `type = TRANSFER_OUT`.
  - One `transactions` record with `type = TRANSFER_IN`.
  - Both accounts' balances updated.
  - `linked_transaction_id` links the two records.
- **Failure:** No transaction created; balances unchanged.

4. Trigger

- User selects **"Transfer"** option from Accounts or Dashboard.

5. Main Success Scenario

1. User opens Transfer form.
2. System displays:
   - Source Account
   - Destination Account
   - Amount
   - Currency (source account currency)
   - Date, description, etc.
3. User selects source and destination accounts and enters amount.
4. System validates:
   - Source and destination accounts are **different**.
   - Amount > 0.
   - Accounts belong to the same user.
5. System checks available balance on source account (if non-negative policy).
6. System creates **TRANSFER_OUT** transaction:
   - `account_id = source account.`
   - `type = TRANSFER_OUT.`
   - Negative effect on balance.

7. System creates **TRANSFER_IN** transaction:
    - `account_id = destination account.`
    - `type = TRANSFER_IN.`
    - Positive effect on balance.
8. System sets each transaction's `linked_transaction_id` to the other.
9. System updates:
    - Decrease `source.current_balance` by amount.
    - Increase `destination.current_balance` by amount (converted if cross-currency).
10. System shows confirmation and updated balances.

6. Extensions

- **5a. Insufficient funds**
    - 5a1. System warns user that source account will go negative or has insufficient balance.
    - 5a2. User can adjust amount or cancel.
- **Cross-currency transfer**
    - C1. If source and destination currencies differ:
        - System prompts user for exchange rate or uses Exchange Rate Service.
    - C2. System calculates destination amount.
    - C3. Two transactions are created with different `amount` values and/or `exchange_rate_to_base`.
- **7a. Destination account not found or archived**
    - 7a1. System shows error and does not create transactions.

UC-06: Define & Track Budget

Use Case ID: UC-06
Primary Actor: Registered User
Level: User-goal

1. Brief Description

User defines budgets over a time period (monthly/weekly/yearly/custom) per category. System tracks actual expenditures vs. limits and provides alerts when thresholds are reached.

2. Preconditions

- User is logged in.
- Categories exist for expenses.

3. Postconditions

- **Success:**
  - New `budgets` and `budget_items` are created.
  - Budget tracking logic is activated for the defined period.
- **Failure:** No changes to budgets.

4. Trigger

- User selects **“Budgets”** and chooses “Create New Budget”.

5. Main Success Scenario: Create Budget

1. User opens **Budgets** page.
2. System lists existing budgets and their statuses (e.g., “In Progress”, “Expired”).
3. User clicks **“New Budget”**.
4. System displays form:
   - Name (e.g., “Jan 2026 Budget”).
   - Period type (Monthly, Weekly, Yearly, Custom).
   - Start date, end date.
   - Optional total limit.
5. User fills in details and submits.
6. System validates:
   - End date ≥ start date.
   - Period type consistent.
7. System creates `budgets` record.
8. System prompts user to **define category-wise limits** (Budget Items).
9. User selects one or more categories and assigns `limit_amount` and optional `warning_percent`.
10. System creates `budget_items` rows for each category.
11. System confirms creation and shows summary of the budget.

6. Main Success Scenario: Track Budget Usage

12. As transactions are added/edited (UC-04, UC-05), system:
    - Identifies if transaction is an EXPENSE within the budget’s date range and category.
    - Aggregates actual spending per `budget_item`.
13. System computes:
    - Consumed amount.
    - Percentage used (`actual / limit_amount * 100`).
14. On each dashboard or budget view, system shows progress (e.g., colored bars).

7. Extensions

- **10a. Duplicate budget for same period**
  - 10a1. System detects existing budget overlapping same period and categories.
  - 10a2. System warns user and allows either:

    - Continue (two budgets overlapping), or
    - Cancel or adjust dates.
- **14a. Threshold notification**
  - 14a1. When spending crosses `warning_percent` or exceeds `limit_amount`:
    - System generates an internal alert event.
  - 14a2. Notification Service (UC-11) sends alert via chosen channel (email, push, etc.).
- **Budget Modification**
  - B1. User edits an existing budget (change dates, amounts).
  - B2. System recalculates progress and may re-trigger threshold checks.

UC-07: Set Up and Manage Recurring Transactions

Use Case ID: UC-07
Primary Actor: Registered User
Level: User-goal

1. Brief Description

User schedules recurring transactions (e.g., rent, salary, subscriptions). System automatically generates future transactions according to frequency rules.

2. Preconditions

- User is logged in.
- Relevant account and category exist.

3. Postconditions

- **Success:**
  - A `recurring_rules` record is created.
  - On each due date, system creates new `transactions` entries and updates account balances.
- **Failure:** No recurring rules created.

4. Trigger

- User chooses **“Make this transaction recurring”** from a transaction form or goes to Recurring section and creates a new rule.

5. Main Success Scenario: Create Recurring Rule

1. User initiates "Add Recurring Transaction".
2. System displays form:
    - Account
    - Category
    - Type (Expense, Income, Transfer)
    - Default Amount & Currency
    - Description (e.g., "Monthly Rent")
    - Frequency (Daily, Weekly, Monthly, Yearly)
    - Interval (Every 1 month, every 2 weeks, etc.)
    - Day-of-month or day-of-week (if applicable)
    - Start Date, End Date (optional)
3. User fills details and submits.
4. System validates:
    - Dates are consistent.
    - Frequency + interval settings make sense.
5. System sets initial `next_run_date` based on start date and frequency.
6. System creates `recurring_rules` record.
7. System confirms to user and displays in Recurring list.

6. Main Success Scenario: Scheduler Generates Transactions

8. On or after `next_run_date`, a scheduler/background job runs.
9. For each active recurring rule:
    - Scheduler checks if today's date ≥ `next_run_date` and ≤ `end_date` (if set).
    - Scheduler creates a new `transactions` record:
        - uses rule's account, category, type, amount, currency, description.
        - sets `is_recurring_instance = 1`.
        - links `recurring_rule_id`.
    - Scheduler updates `account.current_balance`.
    - Scheduler calculates next `next_run_date` by applying frequency & interval.
10. Notification may be sent (optional) when transaction is generated.

7. Extensions

- **4a. Invalid schedule (e.g. Interval = 0)**
    - 4a1. System shows validation error and does not save rule.
- **9a. Insufficient funds**
    - 9a1. For recurring expenses, scheduler detects potential negative balance.
    - 9a2. System may:
        - Skip transaction and mark rule as "Failed execution".
        - Or create transaction but mark as PENDING.
    - 9a3. User is notified via Notification Service.
- **Deactivate or Pause Rule**
    - D1. User views recurring rules list.
    - D2. User toggles rule to inactive.
    - D3. System sets `is_active = 0`; scheduler ignores this rule.

UC-08: View Dashboard & Reports

Use Case ID: UC-08
Primary Actor: Registered User
Level: User-goal

1. Brief Description

User views an overview of their financial situation: account balances, recent transactions, category-wise spending, budget progress, and other analytics (e.g., trends, top categories).

2. Preconditions

- User is logged in.
- At least one account and transaction may exist.

3. Postconditions

- No data is modified; only read operations.
- User gains insight into financial health.

4. Trigger

- User logs in (UC-02) and is redirected to Dashboard, or selects **"Dashboard / Reports"**.

5. Main Success Scenario

1. User opens Dashboard.
2. System loads:
   - Current balances for each account.
   - Recent transactions (e.g., last 10–20).
   - Summaries (total expenses, total income in selected period).
   - Category-wise charts (e.g., pie chart of expenses by category).
   - Budget status (e.g., progress bars, over-limit warnings).
3. User may change filters:
   - Date range (this month, last month, custom).
   - Account selection.
   - Only expenses / only income / both.
4. System recomputes and refreshes:
   - Aggregations and charts.
   - Budget consumption metrics.
5. User may click into specific reports, such as:
   - Detailed category breakdown.

    - Month-over-month trend graph.
    - Expense vs. income comparison.
6. System shows selected report with drill-down capability (click a category → list transactions).

6. Extensions

- **3a. No data for selected filters**
    - 3a1. System shows empty state messages (“No transactions for this period”) and suggestions (“Add your first transaction”).
- **Cross-currency reporting**
    - C1. If multi-currency is enabled, system uses `base_amount` from transactions or recalculates using `exchange_rates`.
    - C2. All charts & totals are displayed in user’s base currency.
- **Export from Report**
    - E1. From any report, user clicks **“Export”**.
    - E2. System triggers UC-13 (Export Data).

UC-09: Search, Filter & Tag Transactions

Use Case ID: UC-09
Primary Actor: Registered User
Level: User-goal

1. Brief Description

User searches and filters transactions (by date, category, amount, tags, etc.) and manages tags (create, assign, unassign).

2. Preconditions

- User is logged in.
- Transactions exist (otherwise results will be empty).

3. Postconditions

- **Success:**
    - User views a filtered list of transactions.
    - Tag assignments (`transaction_tags`) may be updated.
- **Failure:** No data changed.

4. Trigger

- User opens **“Transactions / History”** page or uses search bar.

5. Main Success Scenario

1. User opens Transactions page.
2. System shows:
   - Filter controls (date range, account, category, type, amount range, tags, text search).
3. User selects filters and/or enters a keyword (e.g., “petrol”, “school fee”).
4. System validates filter ranges (e.g., start date ≤ end date).
5. System queries transactions with selected criteria.
6. System displays paginated results with key fields (date, account, category, amount, description, tags).
7. User may select one or more transactions and:
   - Add tags (choose existing or create new).
   - Remove tags.
8. System updates `tags` and `transaction_tags` accordingly.
9. User may click a transaction to view full details or edit (UC-04 flows for edit).

6. Extensions

- **3a. Complex search (e.g., “amount > 500 AND tag = ‘Office’”)**
  - 3a1. System supports advanced search syntax or advanced filter panel.
  - 3a2. Parsing and validation happen; invalid conditions show error.
- **7a. Create new tag on the fly**
  - 7a1. User types a new tag name.
  - 7a2. System checks if tag already exists for this user.
  - 7a3. If not, system creates a `tags` record and then links it via `transaction_tags`.

UC-10: Attach & Manage Receipts

Use Case ID: UC-10
Primary Actor: Registered User
Secondary Actor: Storage Service
Level: User-goal

1. Brief Description

User attaches receipt images or documents to transactions for record-keeping and verification.

2. Preconditions

- User is logged in.

- Target transaction exists and belongs to the user.

3. Postconditions

- **Success:**
  - Files are uploaded and `attachments` records created.
- **Failure:**
  - No attachment created; transaction remains unchanged.

4. Trigger

- From transaction detail view, user clicks **“Add Attachment / Upload Receipt”**.

5. Main Success Scenario

1. User opens a transaction detail view.
2. System shows existing attachments (if any) and **“Add Attachment”** button.
3. User clicks “Add Attachment” and selects one or more files (image/PDF).
4. System validates:
   - File size within allowed limit.
   - File type allowed (image, PDF, etc.).
5. System uploads files via Storage Service (local or cloud).
6. For each successfully stored file, system:
   - Creates an `attachments` entry with file name, path/URL, mime type, size.
7. System shows updated list of attachments with preview or download links.

6. Extensions

- **4a. File too large or unsupported type**
  - 4a1. System rejects file and shows error.
  - 4a2. User may choose another file.
- **Delete Attachment**
  - D1. User clicks “Delete” on an attachment.
  - D2. System asks for confirmation.
  - D3. System deletes attachment record and optionally deletes physical file.
  - D4. Attachment list is refreshed.

UC-11: Configure Notifications & Alerts

Use Case ID: UC-11
Primary Actor: Registered User
Secondary Actor: Notification Service
Level: User-goal

1. Brief Description

User configures which alerts they want to receive (budget threshold, large expenses, recurring failures) and through which channel (email, push, SMS, in-app).

2. Preconditions

- User is logged in.

3. Postconditions

- **Success:**
  - Corresponding `notification_preferences` entries are created/updated.
- **Failure:**
  - No changes to notification settings.

4. Trigger

- User navigates to **"Settings → Notifications"**.

5. Main Success Scenario

1. User opens Notification Settings page.
2. System retrieves current preferences for user and displays:
   - Budget threshold alerts.
   - Large expense alerts (with threshold amount).
   - Recurring failure alerts, etc.
   - For each: channels (Email, Push, SMS, In-app).
3. User toggles each notification type on/off and chooses channels.
4. For large expense alerts, user sets `threshold_amount`.
5. User clicks "Save".
6. System validates:
   - Threshold amount (if provided) is ≥ 0.
7. System updates or creates `notification_preferences` rows.
8. System confirms that preferences have been saved.

6. Extensions

- **3a. Channel not available (e.g. SMS disabled for region)**
  - 3a1. System greys out unavailable channels or shows a message.
  - 3a2. User can only select valid channels.
- **Runtime notification triggers (from other UCs)**
  - T1. Budget crosses warning threshold (from UC-06).
  - T2. Large single expense exceeds threshold (from UC-04).
  - T3. Recurring rule failed to post transaction (from UC-07).

- In each case:
  - System checks `notification_preferences`.
  - If enabled, Notification Service sends alerts using chosen channels.

UC-12: Export / Backup Data

Use Case ID: UC-12
Primary Actor: Registered User
Level: User-goal

1. Brief Description

User exports their data (transactions, accounts, budgets, etc.) into a file (CSV, Excel, PDF, or JSON) for backup or offline analysis.

2. Preconditions

- User is logged in.

3. Postconditions

- **Success:** Export file is generated and made available for download.
- **Failure:** No file generated.

4. Trigger

- User clicks **"Export Data"** from Dashboard or Reports (UC-08).

5. Main Success Scenario

1. User opens Export dialog.
2. System asks user to choose:
   - Data types (Transactions, Accounts, Budgets, Categories, etc.).
   - Date range (for transactions).
   - Format (CSV, Excel, PDF, JSON).
3. User selects desired options and clicks "Export".
4. System validates:
   - At least one data type chosen.
   - Date range valid.
5. System queries data according to criteria.
6. System generates export file in selected format.
7. System provides download link or automatically downloads file to user.

6. Extensions

- **5a. No data found for selected criteria**
    - 5a1. System displays “No data to export for selected period.”
    - 5a2. No file is generated.
- **6a. Large data volume**
    - 6a1. System may paginate and stream or queue background export.
    - 6a2. System notifies user when export is ready (via UC-11).