1. Product Requirement Document (PRD)

Introduction

This document outlines the requirements for an Expense Management application designed to streamline and automate expense reimbursement processes for companies[1]. The system will provide clear roles, flexible approval workflows, and multi-currency support to create a transparent and efficient user experience.

Target Audience

- **Admins:** Responsible for setting up the company account, managing users, and defining approval rules.
- **Employees:** Submit expense claims for reimbursement.
- **Managers:** Review and approve/reject expense claims submitted by their team members.

Core Features & User Flow 🗺️

The application flow follows a logical sequence from setup to reimbursement.

Phase 1: Company & User Setup (Admin)

1. **Admin Signup:**
    - A user signs up with their Name, Email, and Password.
    - They must select a country from a dropdown list.
    - **System Action:** Upon successful signup, a new **Company** is automatically created. The currency of the selected country becomes the company's

base currency[2]. The signup user is assigned the

Admin role for this company.

2. **User Management:**
    - The Admin can access a user management dashboard.
    - From here, they can create new users (Employees, Managers) by providing their name and email[3].
    - The Admin assigns a

Role (Employee or Manager) and sets a **Manager** for each Employee[4].

    - A "Send Password" button emails the new user a randomly generated unique password for their first login.

Phase 2: Approval Rule Configuration (Admin)

1. **Rule Creation:**
    - The Admin defines approval rules for different users or scenarios (e.g., "Miscellaneous Expenses for Marc").
    - Each rule can have multiple approvers (e.g., John, Mitchell).
2. **Rule Logic:**
    - **Manager First:** A checkbox determines if the expense must first be approved by the employee's direct manager before proceeding to other approvers[5].
    - **Approval Sequence:** A checkbox to enforce a specific order of approval. If checked, the request goes to Approver 1, then Approver 2, and so on. If unchecked, all approvers are notified at once[6666].
    - **Approval Percentage:** The Admin can specify a minimum percentage of approvers required for the expense to be approved (e.g., 60%)[7].

Phase 3: Expense Submission (Employee)

1. **Create Expense:**
    - An employee logs in and navigates to their expense dashboard.
    - They can create a new expense by uploading a receipt or filling out the form manually.
    - The form includes fields like Description, Category, Date, Amount, and Currency[8]. The employee can submit the expense in

any currency they used for the transaction.

2. **Submit for Approval:**
    - Initially, the expense is in a **Draft** state.
    - Upon submission, the expense status changes to **Waiting Approval**, and the record becomes read-only for the employee. The system then initiates the approval workflow based on the rules defined by the Admin.

Phase 4: Expense Approval (Manager)

1. **Review Dashboard:**
    - A manager logs in and sees a list of expense requests "waiting for approval"[9].

    - The expense amount is **automatically converted** and displayed in the company's base currency using real-time exchange rates.

2. **Approve/Reject:**
    - The manager can view the details of each expense and choose to

Approve or **Reject** it, optionally adding comments[10].

    - Once an action is taken, the status is updated, and the Approve/Reject buttons disappear for that manager. The expense then moves to the next approver in the sequence, or its final status is set if the approval conditions are met.

Non-Functional Requirements

- **External APIs:**
    - **Countries & Currencies:** Use https://restcountries.com/v3.1/all?fields=name,currencies to populate the country selection dropdown during signup[11].
    - **Currency Conversion:** Use https://api.exchangerate-api.com/v4/latest/{BASE_CURRENCY} for real-time currency conversion on the manager's dashboard[12].
- **OCR for Receipts (Stretch Goal):** For the hackathon, OCR is an additional feature. The core functionality should focus on manual expense entry. If time permits, implement OCR to auto-fill expense details from a receipt image[13].

---

2. System Architecture & Tech Stack

This architecture is designed for rapid development using a modern, scalable tech stack.

- **Frontend: Next.js** (React Framework)
- **Backend: Flask** (Python Web Framework)
- **Database: PostgreSQL** (via Supabase for auth, storage, and database hosting)

---

3. Database Schema (PostgreSQL)

Here are the essential tables and their structures for your Supabase project.

companies

Stores company information.

| Column | Type | Constraints | Description |

| :--- | :--- | :--- | :--- |

| id | uuid | Primary Key, default gen_random_uuid() | Unique identifier for the company. |

| name | text | Not Null | Company name provided during signup. |

| base_currency | varchar(3) | Not Null | 3-letter currency code (e.g., USD). |

| created_at | timestamptz | default now() | Timestamp of creation. |

profiles (extends Supabase auth.users)

Stores user-specific data and links users to companies.

| Column | Type | Constraints | Description |

| :--- | :--- | :--- | :--- |

| id | uuid | Primary Key, Foreign Key to auth.users.id | Corresponds to the user's auth ID. |

| full_name | text | | User's full name. |

| company_id | uuid | Foreign Key to companies.id | The company the user belongs to. |

| role | text | Check (role IN ('admin', 'manager', 'employee')) | User's role in the system. |

| manager_id | uuid | Foreign Key to profiles.id (self-referencing) | The user's direct manager. Can be NULL. |

expenses

Stores all expense records.

| Column | Type | Constraints | Description |

| :--- | :--- | :--- | :--- |

| id | uuid | Primary Key, default gen_random_uuid() | Unique identifier for the expense. |

| employee_id | uuid | Foreign Key to profiles.id | The employee who submitted the expense. |

| company_id | uuid | Foreign Key to companies.id | The company the expense belongs to. |

| description | text | Not Null | Description of the expense. |

| category | text | | Expense category (e.g., Food, Travel). |

| amount | numeric | Not Null | The amount in the original currency. |

| currency | varchar(3) | Not Null | 3-letter code of the original currency. |

| expense_date | date | Not Null | Date the expense was incurred. |

| status | text | Check (status IN ('draft', 'submitted', 'approved', 'rejected')) | Current status of the expense. |

| receipt_url | text | | URL to the uploaded receipt image (Supabase Storage). |

| created_at | timestamptz | default now() | Timestamp of submission. |

approval_rules

Defines the rules for expense approvals.

| Column | Type | Constraints | Description |

| :--- | :--- | :--- | :--- |

| id | uuid | Primary Key | Unique identifier for the rule. |

| company_id | uuid | Foreign Key to companies.id | The company this rule belongs to. |

| name | text | Not Null | A descriptive name for the rule. |

| is_manager_approver_first | boolean | default false | If the direct manager must approve first. |

| is_sequence | boolean | default false | If approvers must approve in a specific order. |

| min_approval_percentage | integer | | Minimum percentage of approvers needed. |

rule_approvers

A join table linking users (approvers) to approval rules.

| Column | Type | Constraints | Description |

| :--- | :--- | :--- | :--- |

| id | uuid | Primary Key | Unique identifier. |

| rule_id | uuid | Foreign Key to approval_rules.id | The rule this entry belongs to. |

| approver_id | uuid | Foreign Key to profiles.id | The user who is an approver. |

| sequence_order | integer | | The order of approval (1, 2, 3...) if is_sequence is true. |

expense_approvals

Tracks the status of each approver for a given expense.

| Column | Type | Constraints | Description |

| :--- | :--- | :--- | :--- |

| id | uuid | Primary Key | Unique identifier. |

| expense_id | uuid | Foreign Key to expenses.id | The expense being approved. |

| approver_id | uuid | Foreign Key to profiles.id | The user assigned to approve. |

| status | text | Check (status IN ('pending', 'approved', 'rejected')) | The approval status for this specific user. |

| comments | text | | Optional comments from the approver. |

| updated_at | timestamptz | | When the approver took action. |

---

4. Backend API Design (Flask)

Here are the core API endpoints you'll need to build.

Auth Endpoints

- **POST /api/auth/signup**
  - **Body:** { "name": "...", "email": "...", "password": "...", "country": "India" }
  - **Action:** Creates a Supabase auth user, a company record, and a profile record with the 'admin' role. Fetches the base currency from the country.
  - **Response:** 201 Created, { "access_token": "...", "user": {...} }
- **POST /api/auth/login**
  - **Body:** { "email": "...", "password": "..." }
  - **Action:** Authenticates with Supabase.
  - **Response:** 200 OK, { "access_token": "...", "user": {...} }

User Management Endpoints (Admin Only)

- **POST /api/users**
  - **Body:** { "full_name": "...", "email": "...", "role": "employee", "manager_id": "..." }
  - **Action:** Admin creates a new user. Sends a password reset/setup email.
  - **Response:** 201 Created, { "user": {...} }
- **GET /api/users**
  - **Action:** Get a list of all users in the admin's company.

    - **Response:** 200 OK, [{"id": "...", "full_name": "..."}, ...]

Expense Endpoints

- **POST /api/expenses** (Employee)
    - **Body:** { "description": "...", "category": "Food", "amount": 5000, "currency": "INR", "expense_date": "2025-10-04" }
    - **Action:** Creates an expense in 'draft' or 'submitted' status. If submitted, triggers the approval workflow logic.
    - **Response:** 201 Created, { "expense": {...} }
- **GET /api/expenses/my** (Employee)
    - **Action:** Returns a list of all expenses submitted by the logged-in employee.
    - **Response:** 200 OK, [{"id": "...", "description": "..."}, ...]
- **GET /api/approvals/pending** (Manager)
    - **Action:** Returns a list of expenses waiting for the logged-in manager's approval. Amounts are converted to the company's base currency.
    - **Response:** 200 OK, [{"id": "...", "request_owner": "Sarah", "amount_in_base_currency": 60.50}, ...]
- **POST /api/expenses/{expense_id}/approve** (Manager)
    - **Body:** { "comments": "Looks good." }
    - **Action:** Marks the expense as approved by the current manager. Checks rules to see if the expense is fully approved or moves to the next approver.
    - **Response:** 200 OK, { "status": "approved" }
- **POST /api/expenses/{expense_id}/reject** (Manager)
    - **Body:** { "comments": "Missing receipt." }
    - **Action:** Marks the expense as rejected. This immediately sets the expense status to 'rejected'.
    - **Response:** 200 OK, { "status": "rejected" }

---

5. Hackathon Development Plan (Under 5 Hours) ⏱️

1. **Hour 1: Setup & Core Backend**

- Initialize Supabase project, get API keys. Create the database tables using the schema above.
- Set up Flask project.
- Implement **Signup and Login** endpoints. Test them.

2. **Hour 2: Core Frontend & Expense Submission**
    - Set up Next.js project.
    - Build the Signup, Login, and Employee Dashboard pages.
    - Create the "New Expense" form.
    - Connect the frontend to the backend to allow an employee to log in and submit an expense. At this point, the approval logic isn't needed, just save it to the DB.
3. **Hour 3: Approval Logic (Backend)**
    - This is the most complex part. Implement the backend logic for:
        - The /api/approvals/pending endpoint.
        - The /api/expenses/{id}/approve and /api/expenses/{id}/reject endpoints.
        - Write the logic that checks approval_rules after an approval action is taken.
4. **Hour 4: Manager View & Admin Setup**
    - Build the Manager's "Approvals to review" dashboard in Next.js.
    - Connect the Approve/Reject buttons to the backend APIs.
    - Build the basic Admin screens for creating users and defining a simple approval rule.
5. **Hour 5: Integration, Testing & Polish**
    - Run through the entire user flow: Admin signs up -> creates users -> creates a rule -> Employee submits expense -> Manager approves.
    - Fix bugs, clean up the UI, and prepare for presentation.

Good luck with the hackathon! 👍