**CHAPTER 1**

**INTRODUCTION**

**1.1 INTRODUCTION**

A Personal Expense Tracker is an essential tool for individuals aiming to manage their finances effectively. It allows users to record, categorize, and analyze their income and expenses, helping them monitor spending habits, set budgets, and make informed financial decisions. Utilizing a Database Management System (DBMS) for such a tracker enhances its efficiency and reliability.

A DBMS offers several advantages for an expense tracking system. It ensures data integrity, maintaining accurate and consistent financial records, which is crucial for generating trustworthy reports. Additionally, DBMSs provide robust security features, protecting sensitive financial information from unauthorized access.

Moreover, DBMSs offer backup and recovery capabilities, ensuring that financial data is not lost due to system failures or other unforeseen events. In summary, integrating a Personal Expense Tracker with a DBMS significantly enhances its functionality, security, and reliability, making it an invaluable tool for personal financial management.

**1.2 PROBLEM STATEMENT**

The aim of this mini project is to develop a Personal Expense Tracker using a relational database management system (DBMS). The system will allow users to record and manage their daily expenses efficiently. Users should be able to add, edit, and delete expense entries, categorize expenses, and generate insightful reports based on various criteria such as date range, categories, and total expenditure. The system should ensure data integrity, provide a user-friendly interface for interaction, and offer secure access to authorized users only. The primary objective is to provide individuals with a tool to monitor their spending habits effectively and make informed financial decisions.

**1.3 Objective**

* Help users manage and track their income and expenses effectively.
* Provide tools for budgeting, financial planning, and spending analysis.

**1.4 Features**

* Transaction Management: Add, edit, delete, and categorize income and expense entries, including recurring transactions.
* Budgeting and Analysis: Set budgets, track spending, and visualize financial data with graphs, charts, and detailed reports.
* Accessibility and Security: Mobile-friendly design, multi-currency support, secure login, and encrypted data storage with backup options.

**CHAPTER 2**

**SOFTWARE REQUIRMENT SPECIFICATION**

**2.1 HARWARE REQUIREMENTS**

The basic hardware required for the development of this project is:

* Processor: Intel Core i5
* Processor speed :1.19 GHz
* RAM :8 GB RAM
* System Type :64-Bit Operating System

**2.2 SOFTWARE REQUIREMENTS**

The software required for the development of this project is:

* Software: XAMPP v3.3.0 & PHP.
* OS: Windows 10 and higher.
* Front End: HTML, CSS .
* Backend : Javascript, PHP.
* Programming Language: SQL
* Database: MySQL

**Purpose**

The purpose of this document is to outline the functional and non-functional requirements for a Personal Expense Tracker. The application is designed to help users track and manage their expenses, while allowing an admin to oversee categories, users, and reports.

**Scope**

The system will include two interfaces:

**Admin Interface**: Manage expense categories, registered users, reports, and profile settings.

**User Interface**: Track personal expenses via dashboards, add/manage expenses, generate reports, and manage user profiles.

The software will support web and mobile platforms and provide a user-friendly experience for both roles.

**1.3 Definitions, Acronyms, and Abbreviations**

Admin: A system administrator who manages categories, users, and reports.

User: A registered user who tracks their personal expenses.

**1.5 Overview**

This document describes the functional and non-functional requirements for the Personal Expense Tracker**.**

**2. Overall Description**

**2.1 Product Perspective**

The Personal Expense Tracker is a standalone system allowing users to track their expenses and admins to manage the system. It provides expense insights and reporting tools.

**2.2 Product Functions**

**The system will provide the following functionalities:**

**Admin Interface:**

Login/logout functionality.

Manage expense categories.

Manage registered users.

Generate system-wide reports.

Update admin profile.

**User Interface:**

View expense dashboard.

Add new expenses with details like date, category, item, and cost.

Manage expenses (edit/delete).

Generate reports (day-wise, month-wise, year-wise, and category-wise).

Update user profile.

**2.3 User Characteristics**

Admin: Familiar with category and user management.

User: General users with basic knowledge of tracking expenses.

**2.4 Constraints**

The system must be accessible on modern web browsers and mobile devices.

Data storage should comply with security and privacy standards.

**2.5 Assumptions and Dependencies**

Internet connection is required to access the system.

Users and Admins will have unique login credentials.

**3. Functional Requirements**

**3.1 Admin Interface**

Login: Admin must log in using a username and password.

Category Management: Add, edit, and delete expense categories.

User Management: View, deactivate, or activate registered users.

Reports: Generate expense reports across all users by day, month, year, or category.

Profile: Update profile information (name, email, password).

Logout: Securely log out of the application.

**3.2 User Interface**

Dashboard: Display expense summaries for:

Today’s expenses

Yesterday’s expenses

Last 7 days

Last 30 days

Current year

Total expenses

Add Expense:

Fields: Date of expense, category, item (expense description), cost.

Validation for required fields.

Manage Expense:

View a list of added expenses.

Option to delete expenses.

Expense Report:

Generate reports filtered by:

Day-wise

Month-wise

Year-wise

Category-wise

Profile:

Fields: Full Name, Email, Mobile Number, Registration Date (read-only).

Option to update editable fields.

Logout: Securely log out of the application.

**4. Non-Functional Requirements**

**4.1 Performance Requirements**

The system should handle up to 500 concurrent users with minimal performance degradation.

Generate reports within 2 seconds.

**4.2 Security Requirements**

Passwords must be hashed before storing in the database.

Data access should be role-based (Admin/User).

User data should be protected using HTTPS encryption.

**4.3 Usability Requirements**

The interface should be intuitive and responsive on both desktop and mobile devices.

Provide feedback for actions like saving or deleting data.

**4.4 Reliability and Availability**

The system should have 99.9% uptime.

Automatic daily backups of user and expense data.

**4.5 Compliance Requirements**

GDPR compliance for storing and managing user data.

**Diagram References:**

**Use Case Diagrams for Admin and User Interfaces.**

**A diagram of a software system

Description automatically generated**

**Use Case: Manage Users**

**Primary Actor**: Admin  
**Goal in Context**: The admin manages user accounts, including adding, updating, or deleting user details.  
**Preconditions**:

* The admin must be logged into the system.
* User records must exist in the database for updates or deletions.  
  Trigger: The admin selects the "Manage Users" option in the system.  
  Main Success Scenario (Basic Flow):

1. The admin logs into the system.
2. The admin navigates to the "Manage Users" section.
3. The admin selects an action (add, update, or delete a user).
4. The system validates the input and performs the requested operation.
5. The system confirms the action and updates the user database accordingly.

**Use Case: Add Expense**

**Primary Actor**: User  
**Goal in Context**: The user adds a new expense entry to track spending.  
**Preconditions:**

* The user must be logged into the system.
* Expense categories must exist or be selectable.  
  Trigger: The user clicks on "Add Expense" in the dashboard.  
  Main Success Scenario (Basic Flow):

1. The user logs into the system.
2. The user navigates to "Add Expense".
3. The user inputs expense details (amount, category, date, description).
4. The system validates the details and saves the expense entry.
5. The system confirms the expense has been added successfully.

**Use Case: Manage Expense**

**Primary Actor**: User  
**Goal in Context**: The user edits or deletes existing expense entries.  
**Preconditions**:

* The user must be logged into the system.
* Expense records must exist in the database.  
  Trigger: The user navigates to the "Manage Expense" section.  
  Main Success Scenario (Basic Flow):

1. The user logs into the system.
2. The user selects "Manage Expense".
3. The system displays a list of recorded expenses.
4. The user chooses an expense to edit or delete.
5. The system validates and performs the requested operation.
6. The system updates the expense database and confirms the change.

**Use Case: Generate Expense Report**

**Primary Actor:** User  
**Goal in Context**: The user generates a report summarizing expenses for a specific period or category.  
**Preconditions**:

* The user must be logged into the system.
* Expense records must exist in the database.  
  Trigger: The user selects "Generate Expense Report" from the system.  
  Main Success Scenario (Basic Flow):

1. The user logs into the system.
2. The user navigates to "Generate Expense Report".
3. The user specifies criteria (e.g., date range, categories).
4. The system retrieves matching records and generates the report.
5. The system displays the report in tabular or graphical format.

**Use Case: Update Profile**

**Primary Actor**: User  
**Goal in Context**: The user updates their personal profile, such as name, email, or password.  
**Preconditions:**

* The user must be logged into the system.
* The user profile must exist in the database.  
  Trigger: The user selects "Update Profile" from the dashboard.  
  Main Success Scenario (Basic Flow):

1. The user logs into the system.
2. The user navigates to "Update Profile".
3. The user edits the desired fields (e.g., name, email, password).
4. The system validates the new information.
5. The system saves the changes and confirms the update.

**Use Case: Logout**

**Primary Actor**: User  
**Goal in Context**: The user logs out of the system to secure their account.  
**Preconditions:**

* The user must be logged into the system.  
  Trigger: The user clicks on "Logout" in the dashboard.  
  Main Success Scenario (Basic Flow):

1. The user selects the "Logout" option.
2. The system terminates the user session.
3. The system redirects the user to the login page.

**ER Diagram for database structure.**

**A diagram of a company

Description automatically generated**

**CHAPTER 3**

**PLANNING PHASE**

**Project Planning for Personal Expense Tracker**

**1. Project Overview**

The Personal Expense Tracker project consists of two interfaces:

Admin Interface: Features include login, category management, user management, reports, profile management, and logout.

User Interface: Features include a dashboard, expense management (add/delete), expense reports (day-wise, month-wise, year-wise, and category-wise), profile management, and logout.

|  |  |  |
| --- | --- | --- |
| Phase | Tasks | Timeline |
| Requirements Gathering | Define requirements and specifications. | Week 1 |
| System Design | Create wireframes, database schema, and diagrams. | Week 2 |
| Development Phase 1 | Develop core functionality (authentication, admin, user). | Weeks 3–4 |
| Development Phase 2 | Implement advanced features (reports, dashboard). | Weeks 5–6 |
| Testing | Perform functional and system testing | Week 7 |
| Deployment | Deploy the system on a server | Week 8 |

**2. Project Schedule**

**Project Plan for Personal Expense Tracker**

Total project duration: 8 weeks

**Project Overview**

The project is a personal expense tracker with separate interfaces for the admin and user. The admin interface manages users, categories, and reports, while the user interface focuses on tracking and managing expenses.

**Key Modules and Features**

**Admin Interface**

Login: Secure login using a username and password.

Category Management: Add, update, and delete expense categories.

User Management: Manage registered users (view, delete, or update user details).

Reports: View expense reports, including day-wise, month-wise, year-wise, and category-wise breakdowns.

Profile Management: Update admin details such as full name, email, and mobile number.

Logout: Secure logout functionality.

User Interface

Dashboard: Display insights such as:

Today's expenses

Yesterday's expenses

Expenses from the last 7 days

Last 30 days

Current year

Total expenses

Expense Management:

Add Expense: Fields include:

Date of expense

Category

Item description

Cost

Manage Expense: View, update, or delete expenses.

Expense Reports:

Day-wise expenses

Month-wise expenses

Year-wise expenses

Category-wise expenses

Profile: Displays and updates personal details such as full name, email, mobile number, and registration date.

Team and Responsibilities

Person A:

Design and develop the Admin Interface.

Implement login and category management features.

Handle user management and admin profile functionalities.

Person B:

Build the User Interface Dashboard.

Implement the logic for displaying expense summaries (e.g., today's expense, last 7 days, etc.).

Ensure the dashboard is visually appealing and functional.

Person C:

Develop the Expense Management Module.

Implement "Add Expense" and "Manage Expense" functionalities, including the delete option.

Person D:

Work on the Expense Reports Module.

Create logic and UI for generating day-wise, month-wise, year-wise, and category-wise reports.

Person E:

Handle the User Profile Module.

Ensure seamless user experience for updating profile details.

Implement validation for profile data.

Project Timeline

Week 1: Requirements finalization and interface design.

Week 2: Set up the backend (database and APIs) and start development on the admin interface.

Week 3: Develop the user dashboard and expense management module.

Week 4: Build the reports module and profile functionalities.

Week 5: Testing and debugging.

Week 6: Deployment and final review.

Additional Notes

Ensure responsive design for seamless use across devices.

Prioritize security for login credentials and sensitive user data.

Use version control (e.g., Git) to manage collaborative development effectively.

**1.2 DBMS (DATABASE MANAGEMENT SYSTEM)**

Database is a collection of related data and data is a collection of facts and figures that can be processed to produce information. Mostly data represents recordable facts. Data aids in producing information, which is based on facts. For example, if we have data about marks obtained by all students, we can then conclude about toppers and average marks.

A database management system (DBMS) is a software package designed to define, manipulate, retrieve and manage data in a database. A DBMS generally manipulates the data itself, the data format, field names, record structure and file structure. It also defines rules to validate and manipulate this data.

A DBMS relieves users of framing programs for data maintenance. Fourth- generation query languages, such as SQL, are used along with the DBMS package to interact with a database. Some other DBMS examples include:

* + MySQL
  + SQL Server
  + Oracle
  + dBASE
  + FoxPro

**1.3 XAMPP ,PHP & MySQL Database**

XAMPP is a popular open-source software package that provides a web server environment for Apache, MySQL, PHP, and Perl. However, XAMPP versions typically follow a structure like 7.4.3 or similar, indicating the versions of Apache, PHP, and other components bundled with it.

PHP is a widely-used open-source scripting language that is especially suited for web development and can be embedded into HTML.

MySQL is an open-source relational database management system (RDBMS) developed by MySQL AB, which was later acquired by Oracle Corporation.

Together, PHP and MySQL form a powerful duo for building dynamic and database-driven web applications, providing the tools needed for creating interactive and responsive websites.

**1.5 OBJECTIVES**

The primary objective of a Personal Expense Tracker in a DBMS (Database Management System) is to provide individuals with a tool to effectively manage and track their expenses. This objective encompasses several key goals:

**Expense Recording and Tracking:**

The system should allow users to record their daily expenses accurately and efficiently. This includes capturing details such as amount spent, date of expenditure, description, and categorization.

**Expense Categorization:**

Users should be able to categorize expenses into predefined categories (e.g., groceries, utilities, entertainment) or create custom categories. This helps users to organize and analyze their spending habits.

**Financial Insight and Analysis:**

The system should generate reports and insights based on recorded expenses. Users should be able to view summaries, trends, and comparisons over different time periods (e.g., daily, weekly, monthly). Visual representations like charts and graphs can aid in understanding expenditure patterns.

**Budget Management:**

Optionally, the system may include features for setting and monitoring budgets across various expense categories. This helps users to manage their finances more effectively and make informed decisions about their spending.

**User Accessibility and Security:**

Ensuring secure access to the system through user authentication and authorization mechanisms is crucial. Users should feel confident that their financial data is protected and only accessible to authorized individuals.

**1.6 FEATURES**

**User Registration:**

Users can sign up by providing a username, email, and password.

The system should hash the password before storing it in the database.

**User Authentication:**

Users can log in using their email and password.

Implement session management to track logged-in users.

**User Profile Management:**

Users can view and update their profile information (username and email).

Users can change their password.

**Expense Features:**

**Add New Expense:**

Users can add a new expense by providing the amount, category, description, and date.

The expense is linked to the logged-in user.

**View Expenses:**

Users can view a list of their expenses with filtering options (by date, category, amount).

Display expenses in a tabular format.

**Expense Analysis:**

Display total expenses for a user over a specified period.

Show expenses categorized by month and by category.

**Expense Category Features:**

**Add New Category:**

Users can create new expense categories.

**View Categories:**

Users can view a list of all available categories.

**Edit Category:**

Users can update the name of an existing category.

**Delete Category:**

Users can delete a category.

**Additional Features for a Mini Project:**

**Dashboard:**

A summary dashboard displaying key metrics (total expenses, expenses by category, recent expenses).

**Reports:**

Generate simple reports summarizing expenses for a selected period (monthly, quarterly).

**CHAPTER 2**

**REQUIREMENT SPECIFICATION**

**2.1 HARWARE REQUIREMENTS**

The basic hardware required for the development of this project is:

* Processor: Intel Core i5
* Processor speed :1.19 GHz
* RAM :8 GB RAM
* System Type :64-Bit Operating System

**2.2 SOFTWARE REQUIREMENTS**

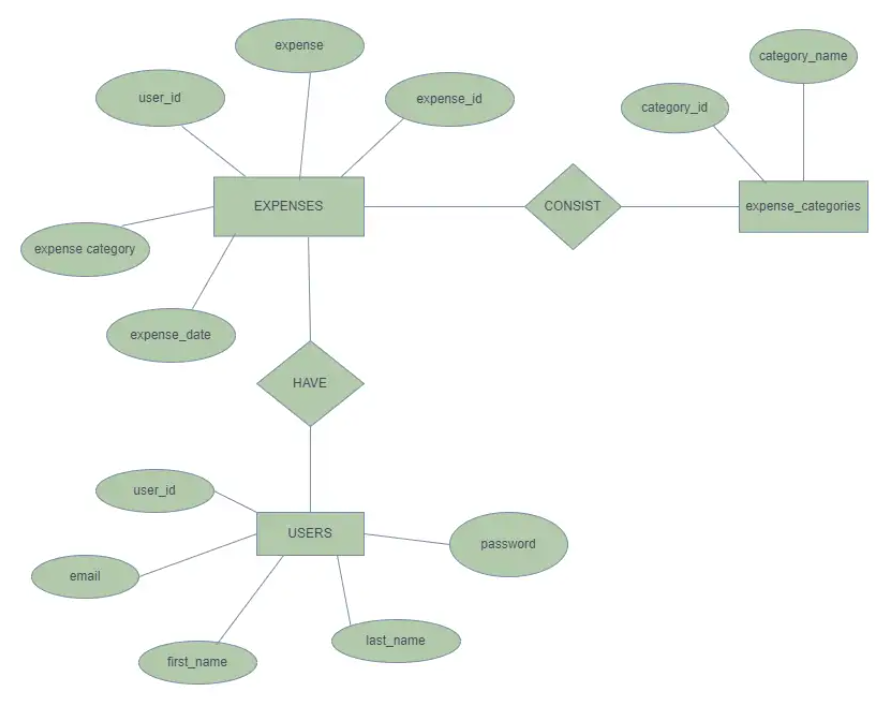
The software required for the development of this project is:

* Software: XAMPP v3.3.0 & PHP.
* OS: Windows 10 and higher.
* Front End: HTML, CSS .
* Backend : Javascript, PHP.
* Programming Language: SQL
* Database: MySQL

**CHAPTER 3**

**DESIGN**

**3.1 ER DIAGRAM AND DESCRIPTION**



**3.1.1 DESCRIPTION**

The ER Model figure shows conceptual view of the database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases. So, let’s see each entity

EXPENSES TABLE shows structure and details of the expense’s table.

EXPENSES CATEGORIES TABLE shows structure and details of the expense category’s table.

USER TABLE shows structure and details of the user’s table.

**3.2 SEVEN STEPS FOR ER TO SCHEMA CONVERSION**

**Step 1: Mapping of Regular Entity Types.**

For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E. Include only the simple component attributes of a composite attribute. Choose one of the key attributes of E as the primary key for R. If the chosen key of E is a composite, then the set of simple attributes that form it will together form the primary key of R. If multiple keys were identified for E during the conceptual design, the information describing the attributes that form each additional key is kept in order to specify secondary (unique) keys of relation R. Knowledge about keys is also kept for indexing purposes and other types of analyses.

**Step 2: Mapping of Weak Entity Types.**

For each weak entity type W in the ER schema with owner entity type E, create a relation R and include all simple attributes (or simple components of composite attributes) of was attributes of R. In addition, include as foreign key attributes of R, the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s); this takes care of mapping the identifying relationship type of W. The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any. If there is a weak entity type E2 whose owner is also a weak entity type E1, then E1 should be mapped before E2 to determine its primary key first.

**Step 3: Mapping of Binary 1:1 Relationship Types.**

For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. There are three possible approaches: 1. The foreign key approach. 2. The merged relationship approach, and 3. The cross-reference or relationship approach. The first approach is the most useful and should be followed unless special conditions exist, as we discuss below.

**1.Foreign key approach:**

Choose one of the relations—S, say—and include as a foreign key in S the primary key of T. It is better to choose an entity type with total participation in R in the role of S. Include all the simple attributes (or simple components of composite attributes) of the 1:1 relationship type R as attributes of S.

**2.Merged relation approach:**

An alternative mapping of a 1:1 relationship type is to merge the two entity types and the relationship into a single relation. This is possible when both participations are total, as this would indicate that the two tables will have the exact same number of tuples at all times.

**3.Cross-reference or relationship relation approach:**

The third option is to set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types. As we will see, this approach is required for binary M: N relationships. The relation R is called a relationship relation (or sometimes a lookup table), because each tuple in R represents a relationship instance that relates one tuple from S with one tuple from T. The relation R will include the primary key attributes of S and T as foreign keys to S and T. The primary key of R will be one of the two foreign keys, and the other foreign key will be a unique key of R. The drawback is having an extra relation, and requiring an extra join operation when combining related tuples from the tables.

**Step 4: Mapping of Binary 1: N Relationship Types.**

For each regular binary 1: N relationship type R, identify the relation S that represents the participating entity type at the N-side of the relationship type. Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R; we do this because each entity instance on the N-side is related to at most one entity instance on the 1-side of the relationship type. Include any simple attributes (or simple components of composite attributes) of the 1: N relationship type as attributes of S.

**Step 5: Mapping of Binary M: N Relationship Types.**

For each binary M: N relationship type R, create a new relation S to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S. Also include any simple attributes of the M: N relationship type (or simple components of composite attributes) as attributes of S. Notice that we cannot represent an M: N relationship type by a single foreign key attribute in one of the participating relations (as we did for 1:1 or 1: N relationship types) because of the M: N cardinality ratio; we must create a separate relationship relation S.

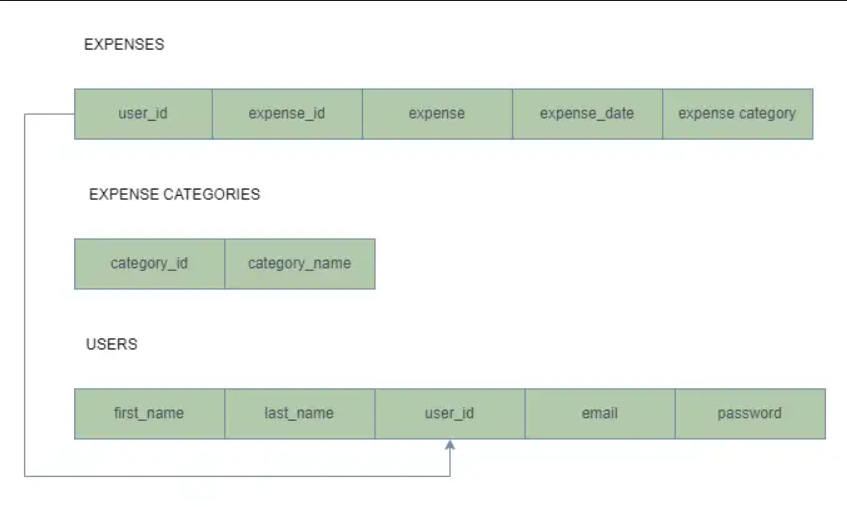
**Step 6: Mapping of Multivalued Attributes.**

For each multivalued attribute A, create a new relation R. This relation R will include an attribute corresponding to A, plus the primary key attribute K—as a foreign key in R—of the relation that represents the entity type or relationship type that has A as a multivalued attribute. The primary key of R is the combination of A and K. If the multivalued attribute is composite, we include its simple components.

**Step 7: Mapping of N-array Relationship Types.**

For each n-array relationship type R, where n > 2, create a new relation S to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types. Also include any simple attributes of the n-array relationship type (or simple components of composite attributes) as attributes of S. The primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity types. However, if the cardinality constraints on any of the entity types E participating in R is 1, then the primary key of S should not include the foreign key attribute that references the relation E ‘corresponding to E.

**3.3 SCHEMA DIAGRAM**



**3.4 DATABASE DESCRIPTION**

**• expense**

**Table 3.4.1: Description table for expense**

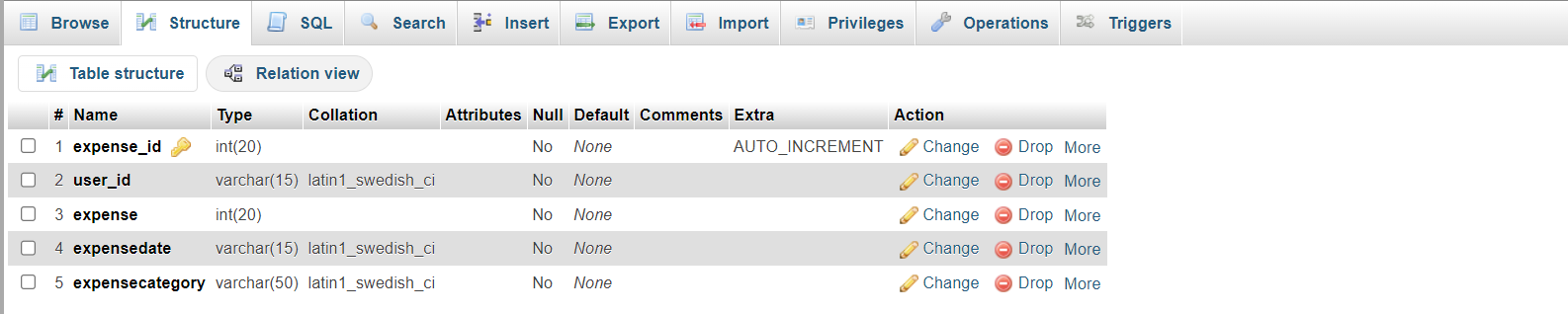


Table 3.4.1 Table shows structure and details expense table.

**• expense\_categories**

**Table 3.4.2: expense\_categories table**

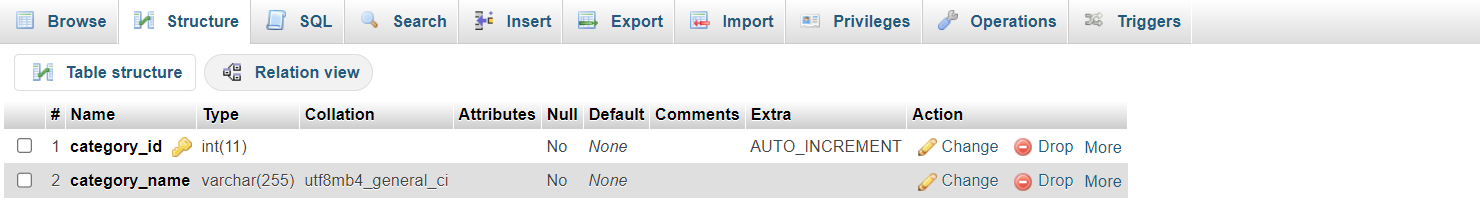


Table 3.4.2 Table shows structure and details of the expense categoriestable

**• users**

**Table 3.4.3: users table**

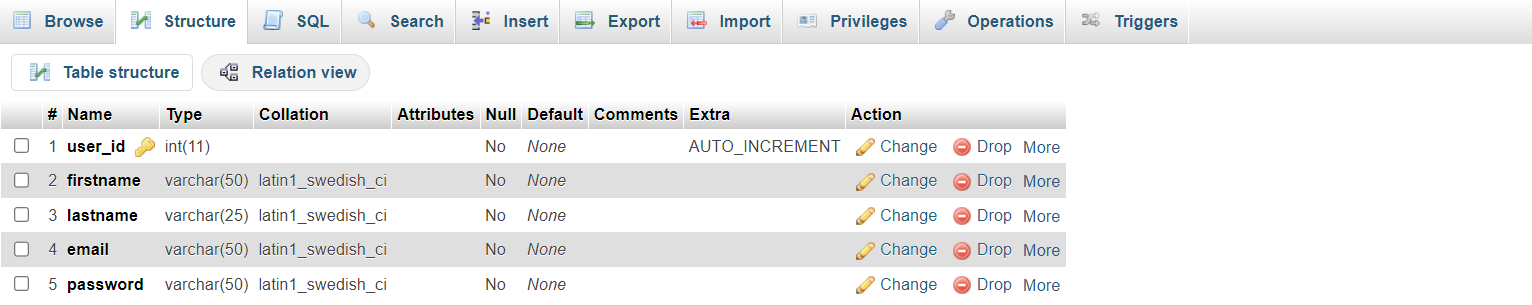


Table 3.4.3 shows structure and details of the users table.

**CHAPTER 4**

**IMPLEMENTATION CODE**

**4.1 CONNECION CODE FOR FRONT END TO BACK END**

<?php

$con = mysqli\_connect("localhost","root","","dailyexpense");

**if** (mysqli\_connect\_errno())

{

**echo** "Failed to connect to MySQL: " . mysqli\_connect\_error() ." | Seems like you haven't created the DATABASE with an exact name";

}

?>

**4.1.1 Login page code**

<?php

require('config.php');

session\_start();

$errormsg = "";

if (isset($\_POST['email'])) {

$email = stripslashes($\_REQUEST['email']);

$email = mysqli\_real\_escape\_string($con, $email);

$password = stripslashes($\_REQUEST['password']);

$password = mysqli\_real\_escape\_string($con, $password);

$query = "SELECT \* FROM `users` WHERE email='$email'and password='" . md5($password) . "'";

$result = mysqli\_query($con, $query) or die(mysqli\_error($con));

$rows = mysqli\_num\_rows($result);

if ($rows == 1) {

$\_SESSION['email'] = $email;

header("Location: index.php");

} else {

$errormsg = "Wrong";

}

} else {

}

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<meta name="description" content="">

<meta name="author" content="">

<title>Login</title>

<!-- Bootstrap core CSS -->

<link href="css/bootstrap.css" rel="stylesheet">

<style>

.login-form {

width: 340px;

margin: 50px auto;

font-size: 15px;

}

.login-form form {

margin-bottom: 15px;

background: #fff;

box-shadow: 0px 2px 2px rgba(0, 0, 0, 0.3);

padding: 30px;

border: 1px solid #ddd;

}

.login-form h2 {

color: #636363;

margin: 0 0 15px;

position: relative;

text-align: center;

}

.login-form h2:before {

left: 0;

}

.login-form h2:after {

right: 0;

}

.login-form .hint-text {

color: #999;

margin-bottom: 30px;

text-align: center;

}

.login-form a:hover {

text-decoration: none;

}

.form-control,

.btn {

min-height: 38px;

border-radius: 2px;

}

.btn {

font-size: 15px;

font-weight: bold;

}

</style>

</head>

<body>

<div class="login-form">

<form action="" method="POST" autocomplete="off">

<h2 class="text-center">Personal Expense Tracker</h2>

<p class="hint-text">Login Panel</p>

<div class="form-group">

<input type="text" name="email" class="form-control" placeholder="Email" required="required">

</div>

<div class="form-group">

<input type="password" name="password" class="form-control" placeholder="Password" required="required">

</div>

<div class="form-group">

<button type="submit" class="btn btn-success btn-block" style="border-radius:0%;">Login</button>

</div>

</form>

<p class="text-center">Don't have an account?<a href="register.php" class="text-danger"> Register Here</a></p>

</div>

</body>

<!-- Bootstrap core JavaScript -->

<script src="js/bootstrap.min.js"></script>

<script src="js/feather.min.js"></script>

<!-- Menu Toggle Script -->

<script>

$("#menu-toggle").click(function(e) {

e.preventDefault();

$("#wrapper").toggleClass("toggled");

});

</script>

<script>

feather.replace()

</script>

</html>

**4.1.2 Logout page code**

<?php

session\_start();

if(session\_destroy())

{

header("Location: login.php");

}

?>

**4.1.3 Register page code**

<?php

require('config.php');

if (isset($\_REQUEST['firstname'])) {

if ($\_REQUEST['password'] == $\_REQUEST['confirm\_password']) {

$firstname = stripslashes($\_REQUEST['firstname']);

$firstname = mysqli\_real\_escape\_string($con, $firstname);

$lastname = stripslashes($\_REQUEST['lastname']);

$lastname = mysqli\_real\_escape\_string($con, $lastname);

$email = stripslashes($\_REQUEST['email']);

$email = mysqli\_real\_escape\_string($con, $email);

$password = stripslashes($\_REQUEST['password']);

$password = mysqli\_real\_escape\_string($con, $password);

$query = "INSERT into `users` (firstname, lastname, password, email) VALUES ('$firstname','$lastname', '" . md5($password) . "', '$email' )";

$result = mysqli\_query($con, $query);

if ($result) {

header("Location: login.php");

}

} else {

echo ("ERROR: Please Check Your Password & Confirmation password");

}

}

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<meta name="description" content="">

<meta name="author" content="">

<title>Register</title>

<!-- Bootstrap core CSS -->

<link href="css/bootstrap.css" rel="stylesheet">

<style>

body {

color: #000;

background: #fff;

font-family: 'Roboto', sans-serif;

}

.form-control {

height: 40px;

box-shadow: none;

color: #969fa4;

}

.form-control:focus {

border-color: #5cb85c;

}

.form-control,

.btn {

border-radius: 3px;

}

.signup-form {

width: 450px;

margin: 0 auto;

padding: 30px 0;

font-size: 15px;

}

.signup-form h2 {

color: #636363;

margin: 0 0 15px;

position: relative;

text-align: center;

}

.signup-form h2:before,

.signup-form h2:after {

content: "";

height: 2px;

width: 30%;

background: #d4d4d4;

position: absolute;

top: 50%;

z-index: 2;

}

.signup-form h2:before {

left: 0;

}

.signup-form h2:after {

right: 0;

}

.signup-form .hint-text {

color: #999;

margin-bottom: 30px;

text-align: center;

}

.signup-form form {

color: #999;

border-radius: 3px;

margin-bottom: 15px;

background: #fff;

box-shadow: 0px 2px 2px rgba(0, 0, 0, 0.3);

padding: 30px;

border: 1px solid #ddd;

}

.signup-form .form-group {

margin-bottom: 20px;

}

.signup-form input[type="checkbox"] {

margin-top: 3px;

}

.signup-form .btn {

font-size: 16px;

font-weight: bold;

min-width: 140px;

outline: none !important;

}

.signup-form .row div:first-child {

padding-right: 10px;

}

.signup-form .row div:last-child {

padding-left: 10px;

}

.signup-form a:hover {

text-decoration: none;

}

.signup-form form a {

color: #5cb85c;

text-decoration: none;

}

.signup-form form a:hover {

text-decoration: underline;

}

</style>

</head>

<body>

<div class="signup-form">

<form action="" method="POST" autocomplete="off">

<h2>Register</h2>

<div class="form-group">

<div class="row">

<div class="col"><input type="text" class="form-control" name="firstname" placeholder="First Name" required="required"></div>

<div class="col"><input type="text" class="form-control" name="lastname" placeholder="Last Name" required="required"></div>

</div>

</div>

<div class="form-group">

<input type="email" class="form-control" name="email" placeholder="Email" required="required">

</div>

<div class="form-group">

<input type="password" class="form-control" name="password" placeholder="Password" required="required">

</div>

<div class="form-group">

<input type="password" class="form-control" name="confirm\_password" placeholder="Confirm Password" required="required">

</div>

<div class="form-group">

<button type="submit" class="btn btn-danger btn-lg btn-block" style="border-radius:0%;">Register</button>

</div>

</form>

<div class="text-center">Already have an account? <a class="text-success" href="login.php">Login Here</a></div>

</div>

</body>

<!-- Bootstrap core JavaScript -->

<script src="js/jquery.slim.min.js"></script>

<script src="js/bootstrap.min.js"></script>

<!-- Croppie -->

<script src="js/profile-picture.js"></script>

<!-- Menu Toggle Script -->

<script>

$("#menu-toggle").click(function(e) {

e.preventDefault();

$("#wrapper").toggleClass("toggled");

});

</script>

<script>

feather.replace()

</script>

</html>

**4.2 SQL STATEMENTS**

CREATE TABLE `expenses` (

`expense\_id` int(20) NOT NULL,

`user\_id` varchar(15) NOT NULL,

`expense` int(20) NOT NULL,

`expensedate` varchar(15) NOT NULL,

`expensecategory` varchar(50) NOT NULL

);

INSERT INTO `expenses` (`expense\_id`, `user\_id`, `expense`, `expensedate`, `expensecategory`) VALUES

(101, '9', 789, '2023-08-31', 'Medicine'),

(102, '9', 3, '2023-08-31', 'Entertainment'),

(103, '9', 469, '2023-08-29', 'Clothings'),

(104, '9', 985, '2023-08-25', 'Entertainment'),

(105, '12', 3, '2023-08-31', 'Clothings'),

(106, '12', 89, '2023-08-16', 'Bills & Recharges'),

(107, '9', 3, '2023-09-06', 'Clothings'),

(108, '9', 300, '2023-07-04', 'Food'),

(109, '9', 456, '2023-09-01', 'Clothings'),

(110, '9', 3, '2023-08-28', 'Entertainment'),

(111, '9', 300, '2023-09-03', 'Clothings'),

(112, '9', 789, '2021-06-03', 'Medicine'),

(113, '9', 756, '2021-02-23', 'Entertainment'),

(114, '9', 123, '2022-09-03', 'Medicine'),

(115, '9', 256, '2021-09-07', 'Medicine'),

(116, '9', 798, '2023-09-04', 'Medicine'),

(117, '9', 45, '2023-08-28', 'Entertainment'),

(118, '9', 50, '2023-10-20', 'Medicine'),

(119, '9', 786, '2023-10-20', 'Food'),

(120, '9', 1000, '2023-10-04', 'Entertainment'),

(121, '9', 500, '2023-10-19', 'Clothings'),

(122, '9', 426, '2023-10-16', 'Household Items');

CREATE TABLE `expense\_categories` (

`category\_id` int(11) NOT NULL,

`category\_name` varchar(255) NOT NULL

);

INSERT INTO `expense\_categories` (`category\_id`, `category\_name`) VALUES

(1, 'Medicine'),

(2, 'Food'),

(3, 'Bills & Recharges'),

(4, 'Entertainment'),

(5, 'Clothings'),

(6, 'Rent'),

(7, 'Household Items'),

(8, 'Others');

CREATE TABLE `users` (

`user\_id` int(11) NOT NULL,

`firstname` varchar(50) NOT NULL,

`lastname` varchar(25) NOT NULL,

`email` varchar(50) NOT NULL,

`password` varchar(50) NOT NULL

);

INSERT INTO `users` (`user\_id`, `firstname`, `lastname`, `email`, `password`) VALUES

(9, 'Anjalita', 'Fernandes', 'anjalita@sjec.in', 'b7161ae9080c2604adb157463312ed47'),

(12, 'Ebey', 'Joe Regi', 'ejr@sjec.in', '25d55ad283aa400af464c76d713c07ad');

ALTER TABLE `expenses`

ADD PRIMARY KEY (`expense\_id`);

ALTER TABLE `expense\_categories`

ADD PRIMARY KEY (`category\_id`);

ALTER TABLE `users`

ADD PRIMARY KEY (`user\_id`);

ALTER TABLE `expenses`

MODIFY `expense\_id` int(20) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=123;

ALTER TABLE `expense\_categories`

MODIFY `category\_id` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=9;

ALTER TABLE `users`

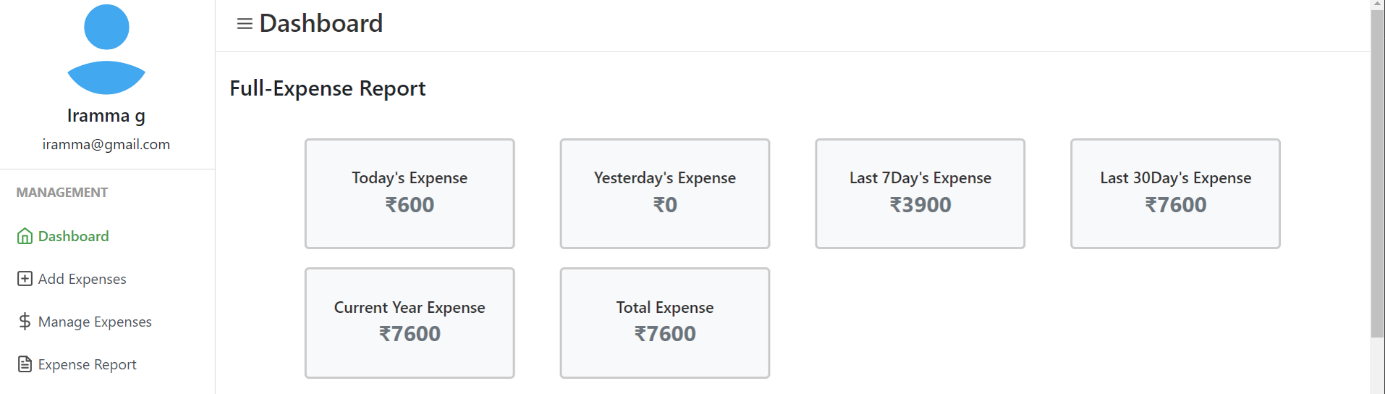
MODIFY `user\_id` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=13;

COMMIT;

**CHAPTER 5**

**SNAPSHOTS**

**Fig 5.1: Dashboard Page**



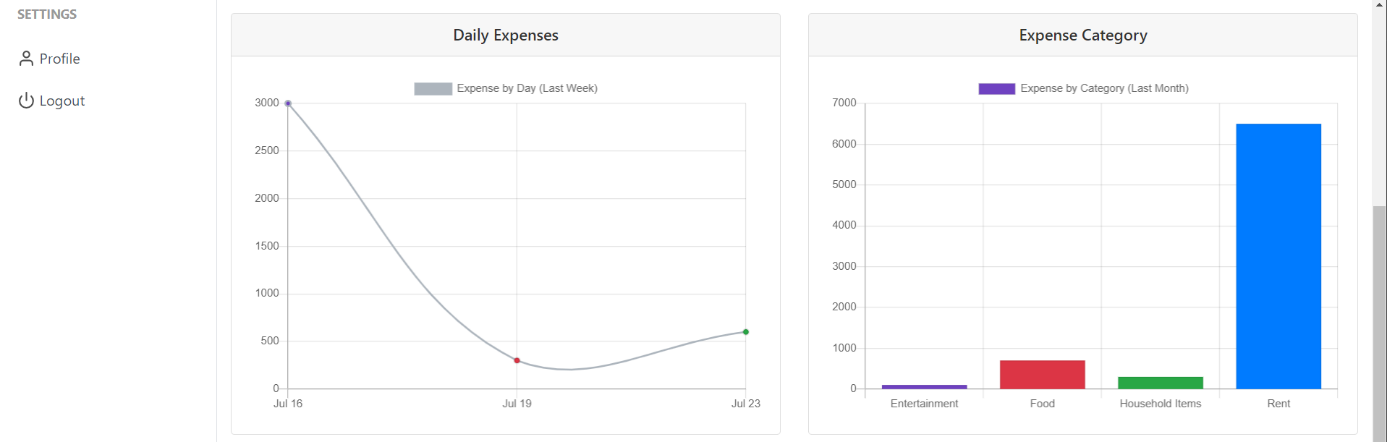
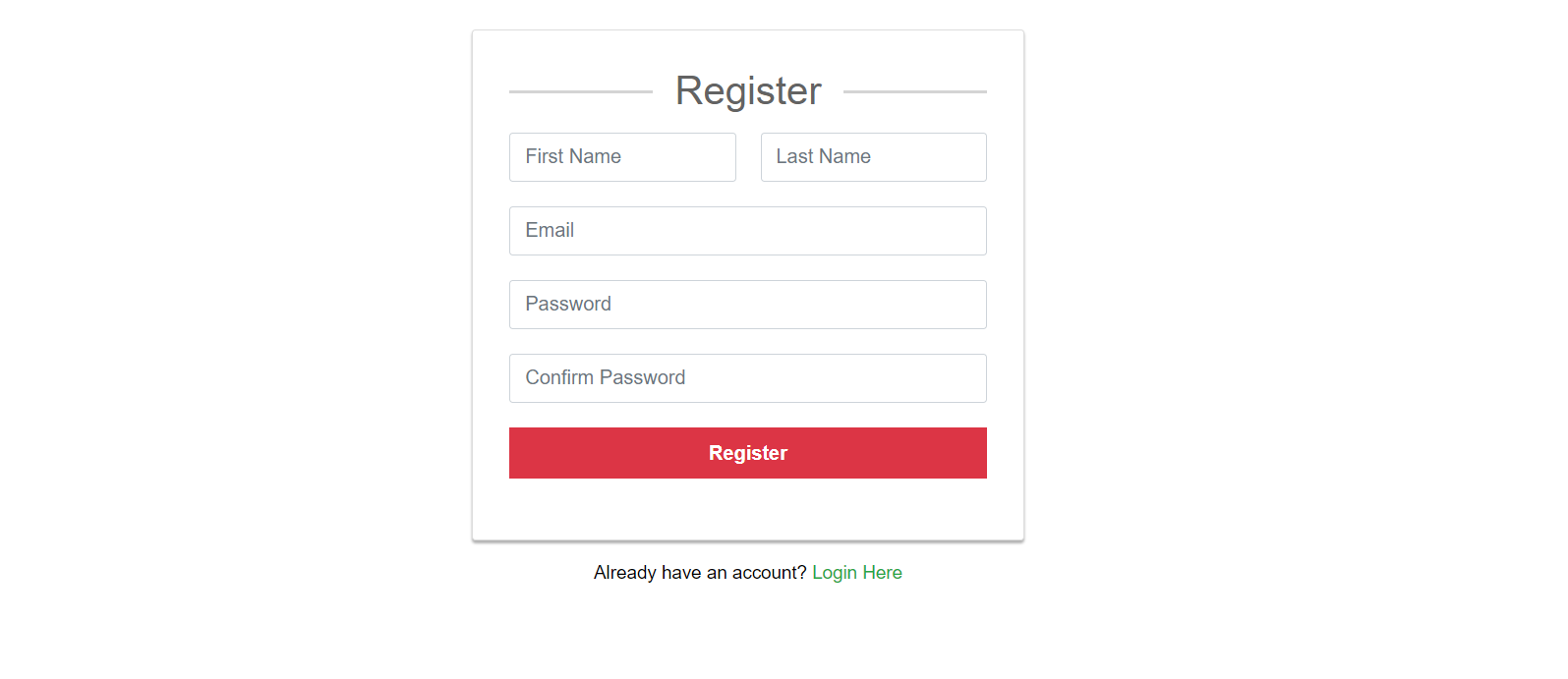




Figure 5.1 shows the dashboard page of the Personal Expense Tracker system.

**Fig 5.2: Register Page**

Figure 5.1 shows the register page of the Personal Expense Tracker system.

**Fig 5.3: Login Page**

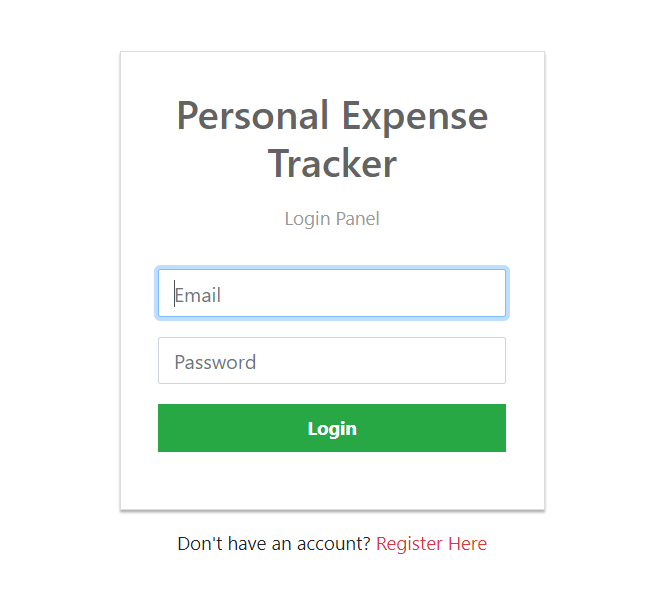


Figure 5.3 shows the login page of the Personal Expense Tracker system.

**Fig 5.4: Adding Daily expense Page**

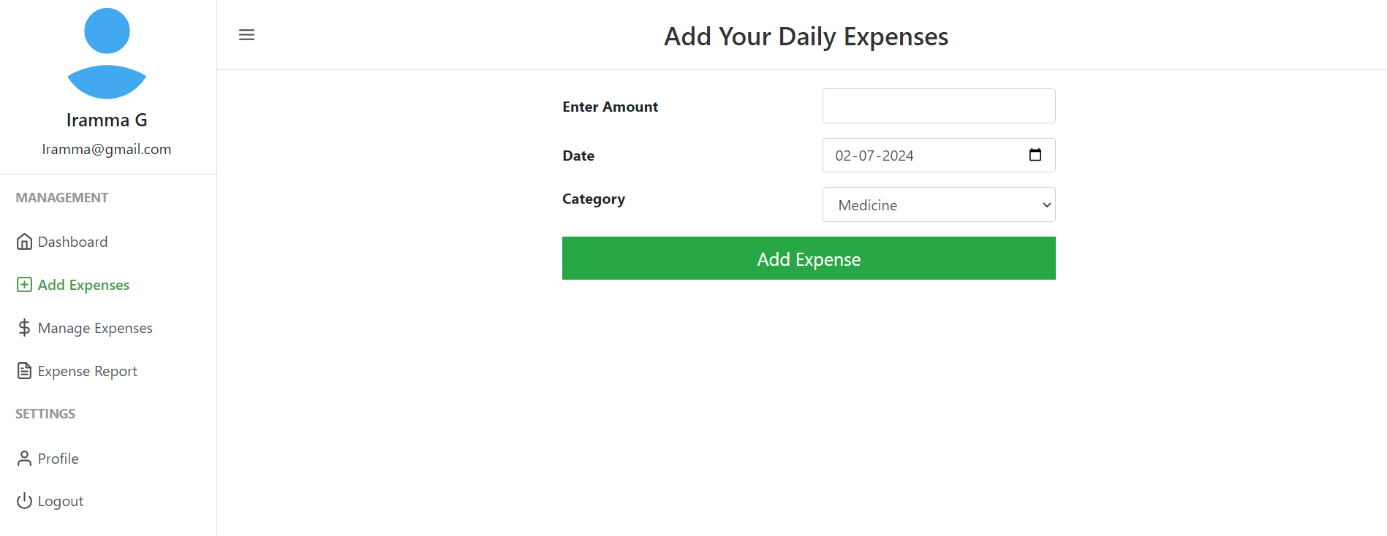


Figure 5.4 shows the adding daily expense page of the Personal Expense Tracker system.

**Fig 5.5 :Daily expense category Page**

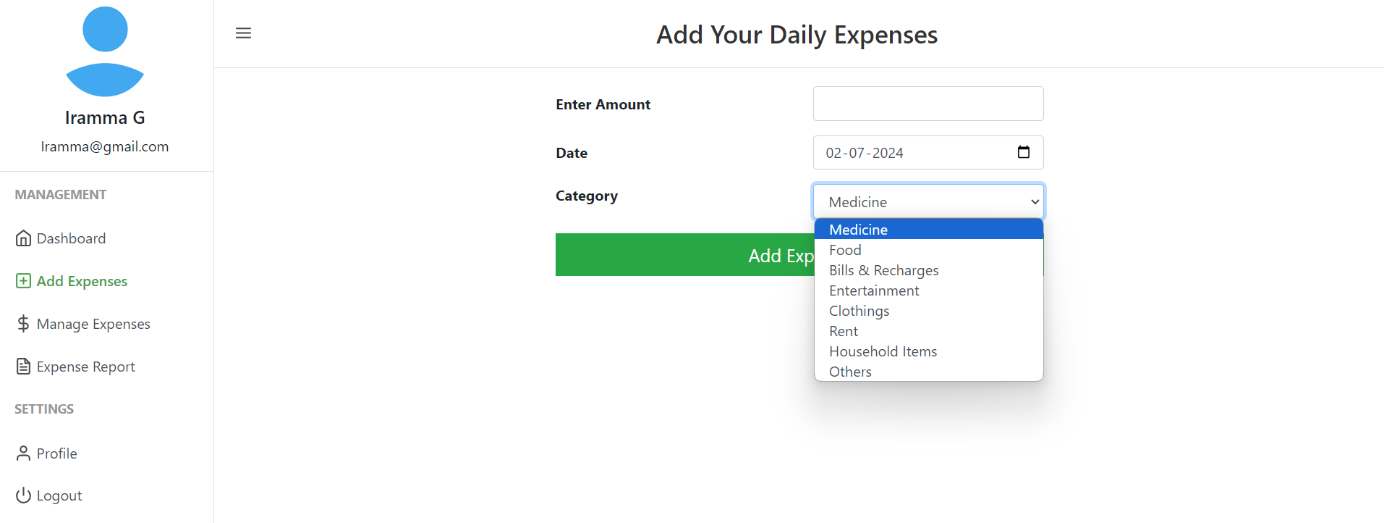


Figure 5.5 shows the daily expense category page of the Personal Expense Tracker system.

**Fig 5.6:Generate expense report Page**

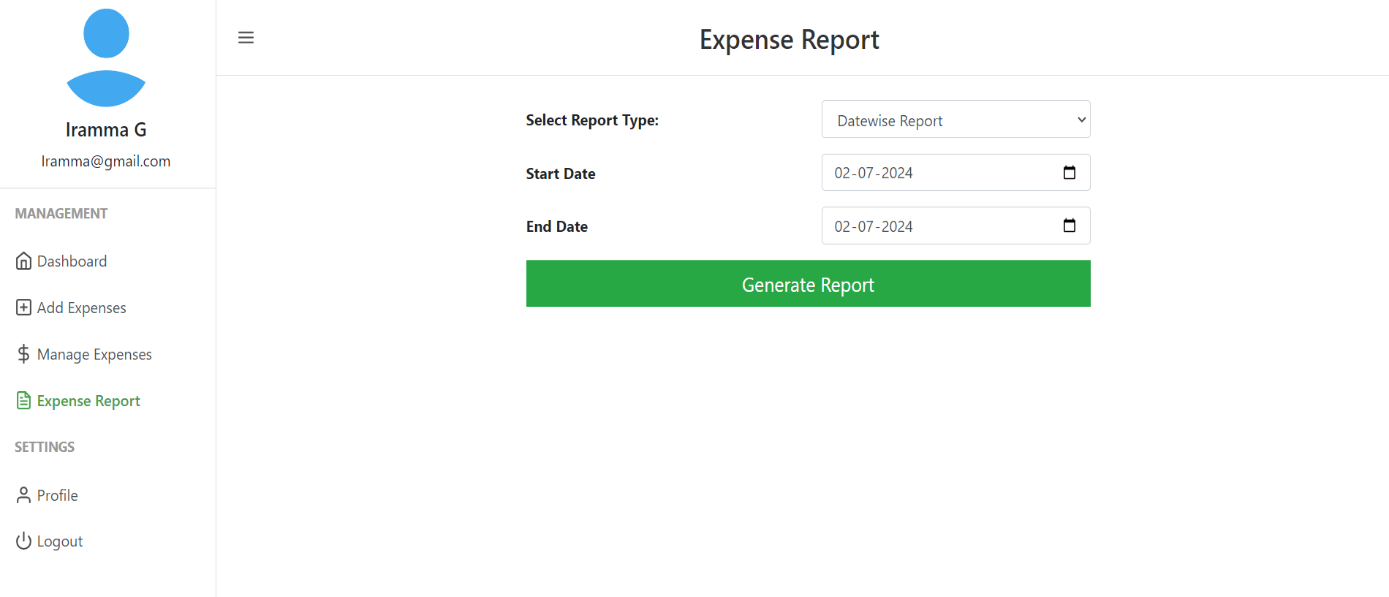


Figure 5.4 shows the generate expense report page of the Personal Expense Tracker system.

**Fig 5.7:Manage Expense report Page**

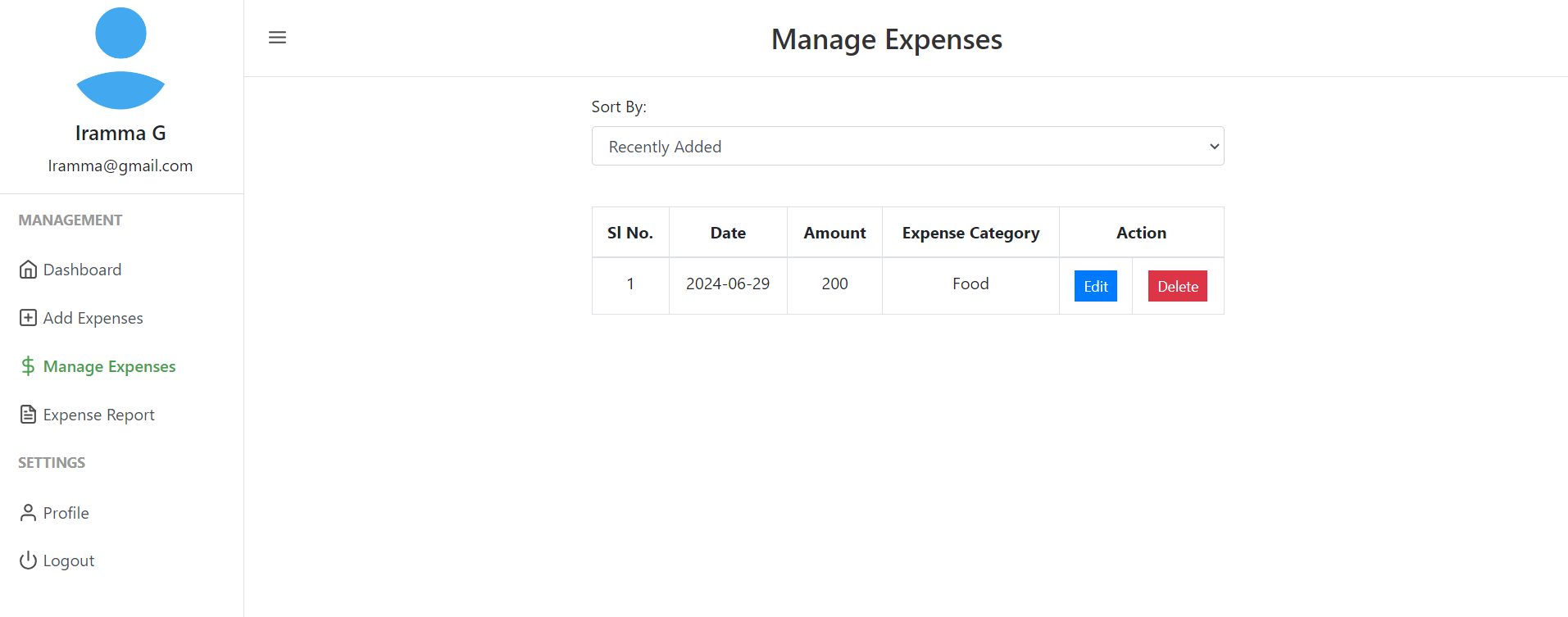


Figure 5.7 shows the manage expense report page of the Personal Expense Tracker system.

**Fig 5.8:Profile Page**

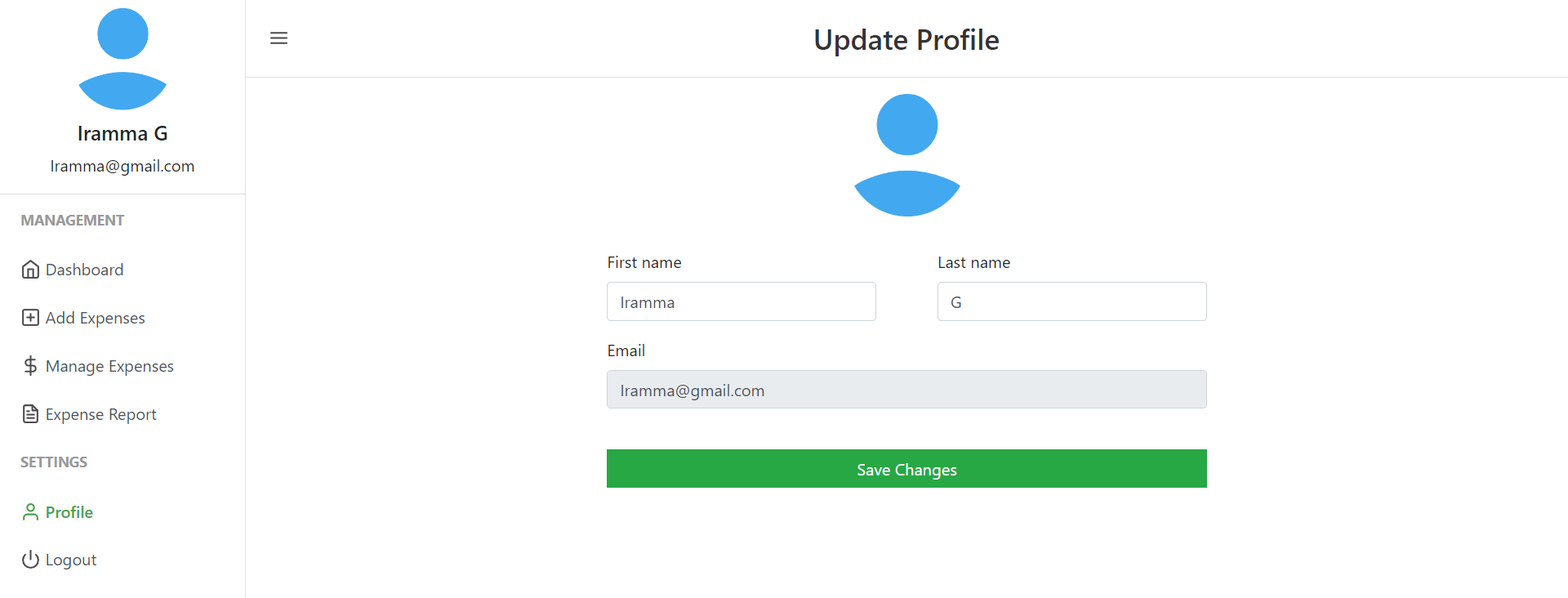


Figure 5.8 shows the profile page of the Personal Expense Tracker system.

**CONCLUSION**

The personal expense tracker DBMS project demonstrates a robust and efficient system for managing individual financial records. By leveraging database management techniques, the project enables users to systematically record, categorize, and analyze their expenses, promoting better financial literacy and budgeting habits. The system's user-friendly interface ensures ease of use, while the integration of data analytics provides insightful visualizations and reports on spending patterns. This project not only highlights the importance of organized financial tracking but also showcases the practical application of DBMS principles in real-world scenarios. Overall, the personal expense tracker DBMS project is a valuable tool for anyone seeking to gain control over their finances and make informed economic decisions. Its successful implementation underscores the significance of technology in simplifying and enhancing everyday financial management.

**REFERENCES**

1. Fundamentals of database systems, Ramez Elmasri and S B Navathe, 7th Edition, 2017, Pearson
2. Database management systems, Ramakrishnan, and Gehrke, 3rd Edition, 2014, McGraw Hill.
3. <https://phpgurukul.com/daily-expense-tracker-using-php-and-mysql/>
4. <https://www.studocu.com/in/document/dr-apj-abdul-kalam-technical-university/btech/mini-project-report/29935835>
5. <https://phpgurukul.com/daily-expense-tracker-using-php-and-mysql/>
6. https://www.geeksforgeeks.org/build-an-expense-tracker-with-html-css-and-javascript/