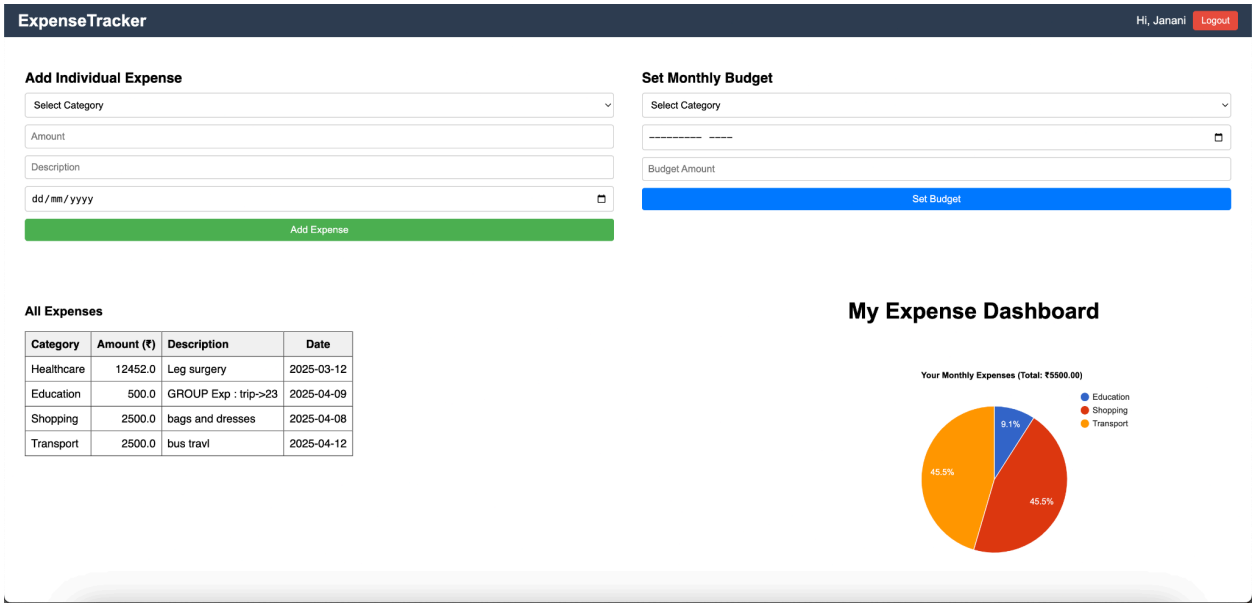
# L7 Expense Tracker Web Application - FLASK

Github link: https://github.com/itzjanani/L7-Informatics-Assignment
Name: Janani M
Register Number: 21MIA1070
Mail ID: janani7m@gmail.com




## Objective

To develop a Python-based full-stack **Expense Tracker Web Application** that:

- Allows users to track personal and group expenses
- Supports monthly budgets
- Includes login/logout functionality using local storage
- Alerts users when spending approaches budget limits
- Provides a fully styled and user-friendly frontend

---

## Phase 1: Database Design

### Finalized Database Models

The application uses the following SQLAlchemy ORM models:

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False, unique=True)
    email = db.Column(db.String(100), unique=True, nullable=False)
    password = db.Column(db.String(100), nullable=False)

class Category(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), nullable=False, unique=True)

class Expense(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    amount = db.Column(db.Float, nullable=False)
    description = db.Column(db.String(200))
    date = db.Column(db.DateTime, default=datetime.utcnow)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    category_id = db.Column(db.Integer, db.ForeignKey('category.id'), nullable=False)

class Budget(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    category_id = db.Column(db.Integer, db.ForeignKey('category.id'), nullable=False)
    month = db.Column(db.String(10), nullable=False)  # e.g., '2025-04'
    amount = db.Column(db.Float, nullable=False)

class Group(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False, unique=True)

class GroupMember(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    group_id = db.Column(db.Integer, db.ForeignKey('group.id'), nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

class GroupExpense(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    group_id = db.Column(db.Integer, db.ForeignKey('group.id'), nullable=False)
    amount = db.Column(db.Float, nullable=False)
    description = db.Column(db.String(200))
    category_id = db.Column(db.Integer, db.ForeignKey('category.id'), nullable=False)
    date = db.Column(db.DateTime, default=datetime.utcnow)
```

---

# Phase 2: API and Route Development

The application uses Flask APIs to manage each table:

### Key Routes:

```
@app.route('/')
def index():
    return render_template('index.html', ...)

@app.route('/user', methods=['POST'])
def create_user(): ...

@app.route('/category', methods=['POST'])
def create_category(): ...

@app.route('/expense', methods=['POST'])
def add_expense(): ...

@app.route('/budget', methods=['POST'])
def set_budget(): ...

@app.route('/group', methods=['POST'])
def create_group(): ...

@app.route('/group/member', methods=['POST'])
def add_group_member(): ...

@app.route('/group/expense', methods=['POST'])
def add_group_expense():
    # Splits group expense among members and adds entries in Expense table
```

This group expense route is a core functionality, ensuring correct expense distribution.

---

# Phase 3: Frontend Testing & Rendering

- **Tested APIs** via a basic HTML frontend
- Rendered all tables live on the UI for verification
- Successfully created and validated test data
- Ensured live update of tables when entries were added

## Create Group

Group Name | Create Group

## Add User

User Name | Email | Password | Add User

## Add Category

Category Name | Add Category

## Add User to Group

User Name | Group Name | Add to Group

## Add Group Expense

Group Name | Category Name | Amount | Description | dd/mm/yyyy | Split Expense

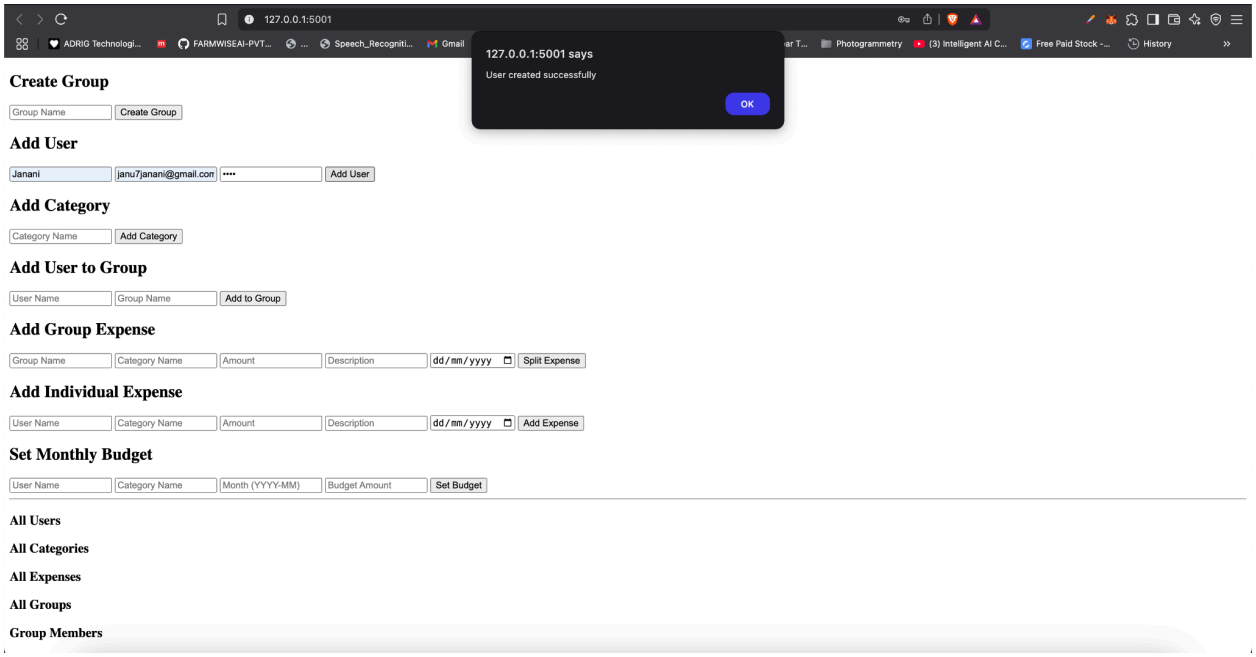
## Add Individual Expense

User Name | Category Name | Amount | Description | dd/mm/yyyy | Add Expense

## Set Monthly Budget

User Name | Category Name | Month (YYYY-MM) | Budget Amount | Set Budget

---



**All Users**

- ID: 1 | Janani | janu7janani@gmail.com
- ID: 2 | Vijay | vijay@sfs.com
- ID: 3 | Varshini | varsss@gma.com

**All Categories**

- ID: 1 | food

**All Expenses**

**All Groups**

- ID: 1 | friends

**Group Members**

- Group ID: 1 | User ID: 1
- Group ID: 1 | User ID: 2
- Group ID: 1 | User ID: 3

**Group Expenses**

**Budgets**

---

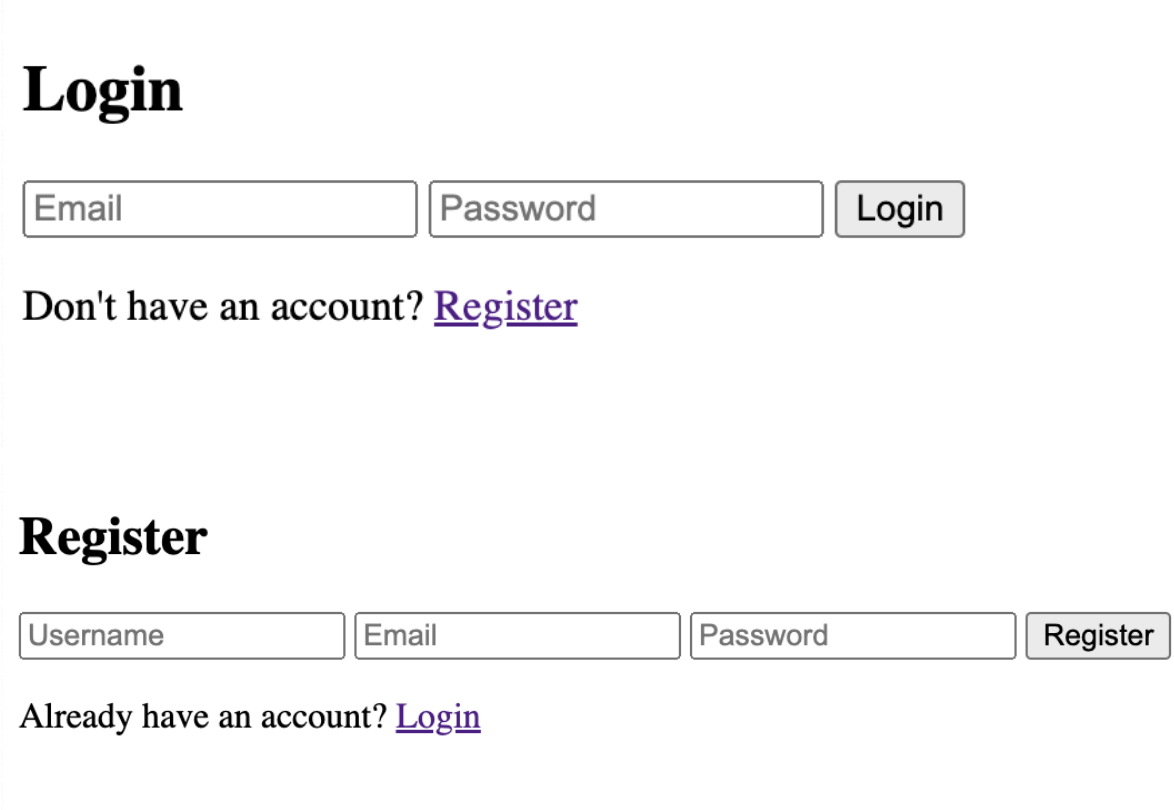
# Phase 4: Login, Local Storage & Conditional Rendering

## Enhancements:

- Created **register** and **login** routes
- Used **localStorage** to persist login state
- Automatically redirects users:
    - If logged in: to dashboard
    - If not logged in: to login page
- Added a **logout button** to clear session from local storage

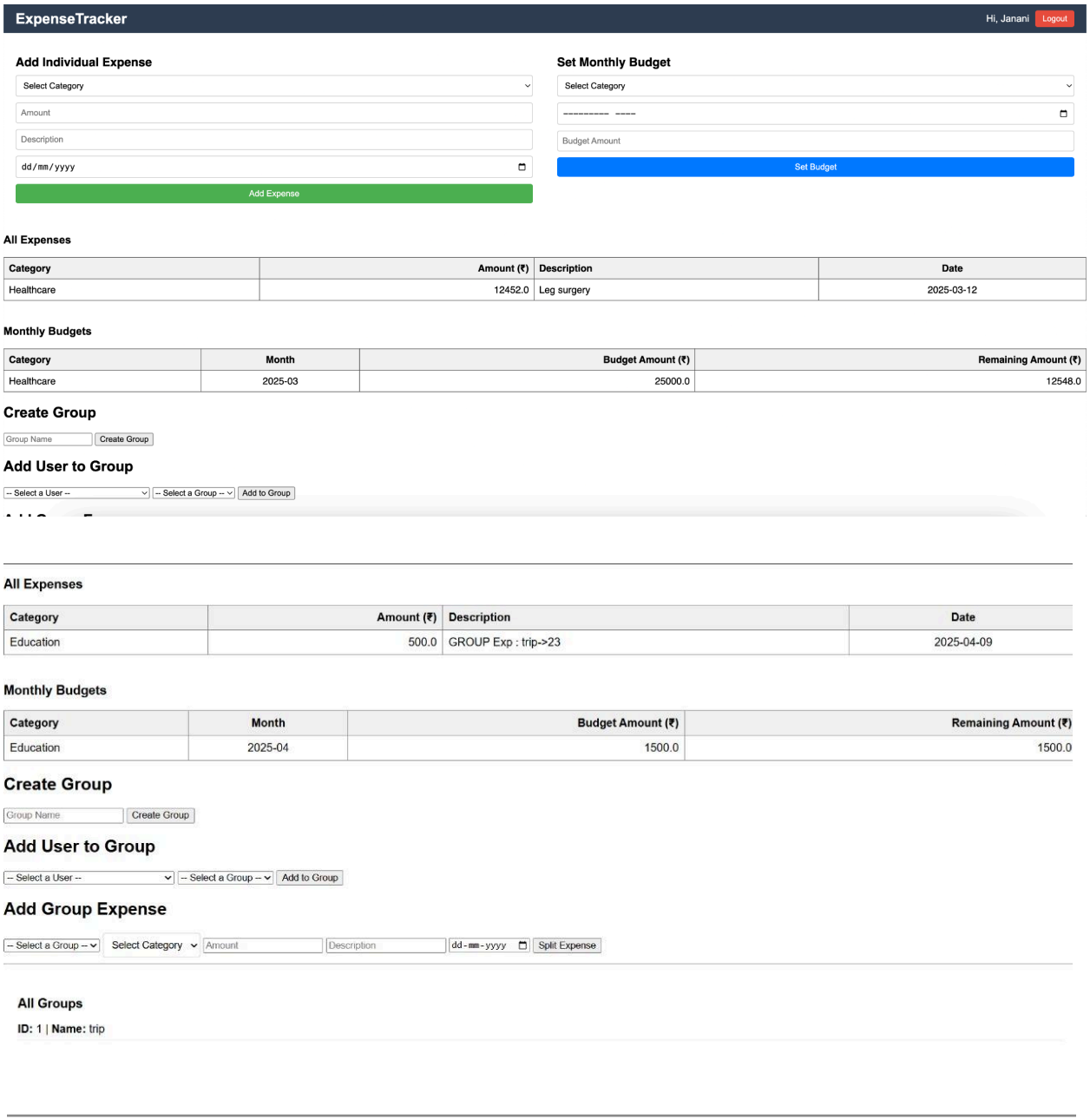
## Conditional Rendering:

- Forms and tables are rendered based on the logged-in user
- Removed need to input user ID; it is inferred from local storage session




## Phase 5: Group Visibility & Ownership, UI minor improvement

- Groups are rendered **only to their creators**
- Dropdown for user selection includes name + unique ID
- Group logic is tied directly to logged-in user identity

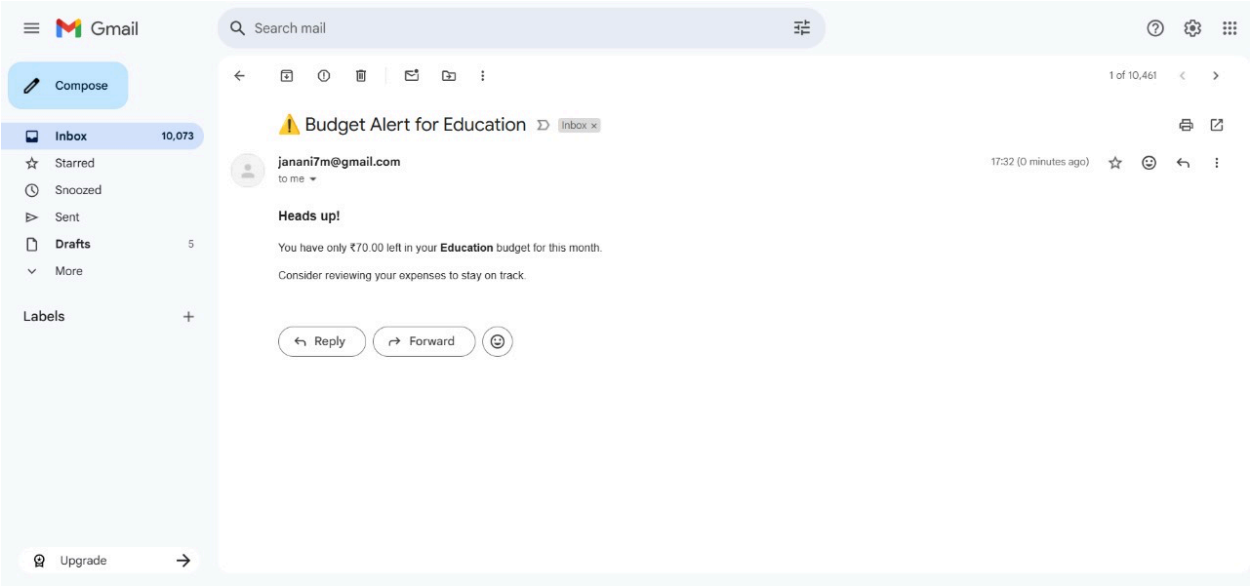



# Phase 6: Budget Management Logic

- Prevent duplicate budget entries for the same month + user + category
- If existing entry exists, **increment existing budget amount**
- All expenses (group/personal) are recorded in user's expense table
- Descriptions of group expenses clearly mention origin group name

# Phase 7: Budget Alert System

- Validates remaining budget
- If remaining budget < 10%, sends an **email alert** via **SMTP**
- Implemented email notifications for near-exceeded budgets

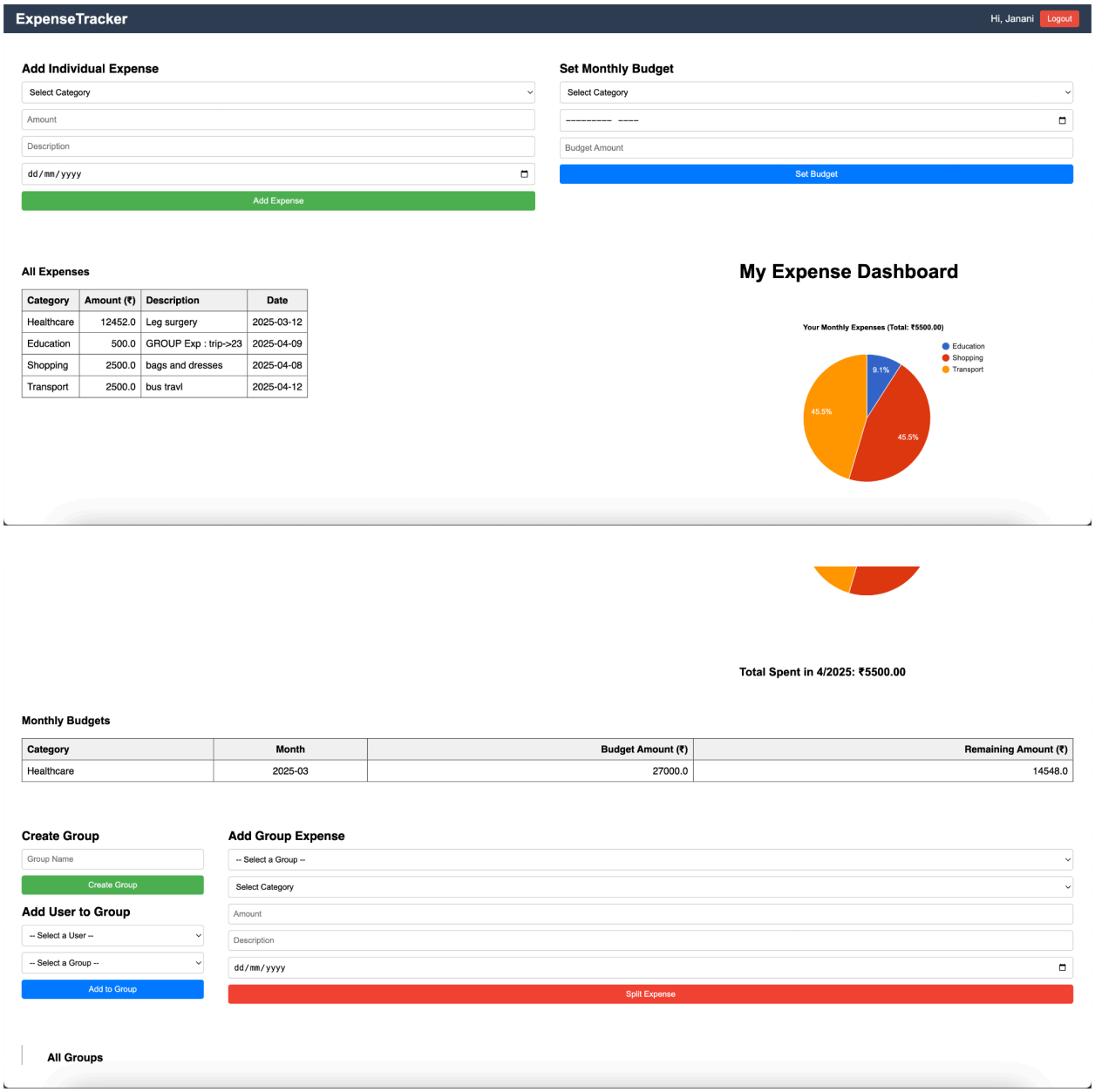



We have set up our app passwords and email in the .env file

---

# Phase 8: Final Styling

- GPT-assisted styling of:
  - Forms
  - Login/Register pages
  - Dashboard
- Final UI is fully responsive and polished




# Outcome

The application meets all expected requirements:

- Robust backend with models and routes
- Feature-rich frontend with conditional rendering
- Budget management with alerts
- Simple User-friendly UI with authentication

- Group expense management with fair splits
- Styled and deployed using modern best practices

Now we use local db, but after everything is completed we have changed to neon db cloud.