Divido – Expense Sharing Web Application Documentation

This document provides documentation for Divido, an expense sharing web application built using Python Django, SQLite, and Django Templates for the user interface.

## **1. Introduction**

This web application serves as a platform for managing shared expenses, offering functionalities typical of a modern expense-splitting system. It allows users to create groups, add and track shared expenses, settle balances, and view individual or group transaction histories. Administrators can manage users, groups, and system-wide settings.

**Key Technologies Used:**

* **Backend Framework:** Python Django
* **Database:** SQLite
* **Frontend Templating:** Django Templates
* **Programming Language:** Python

## **2. Core Features**

The application includes the following primary features:

* **User Management:**
  + User registration and login with OTP-based authentication
  + Each user has a personal dashboard to view their groups and expenses
  + Profile details are stored for group association and tracking
* **Group Management:**
  + Users can create groups and are assigned as the group admin by default
  + Group admin can add or remove members
  + Groups can be deleted by the group admin
* **Expense Management:**
  + Add expenses with equal or custom split
  + Select who paid and how much each person owes
  + View total and individual shares for each expense
  + Partial settlement support between users
* **Transaction History:**
  + Track all transactions within a group
  + See "To be paid" and "To pay" summaries
  + View consolidated balances across all groups
* **Filtering and Sorting:**
  + Sort or filter expenses by amount, group, or date
  + Easier navigation through historical transactions
* **Analytics Dashboard:**
  + "Month at a Glance" summary showing top contributors and category-wise spending
  + Pie chart visualization of expenses by category using [Chart.js](http://chart.js)
* **Light/Dark Theme Toggle:**
  + Remembers user preference across sessions for consistent UI.
  + Enhances readability in different lighting conditions

**3. Architecture**

Divido follows the Model-View-Template (MVT) architectural pattern, inherent to the Django framework.

* **Model:**
  + Represents the application's data structure, business logic, and rules.
  + Interacts directly with the SQLite database via Django's ORM.
  + Examples: User, Group, GroupMembers, Split, Payer, Settlement (defined in models.py files within Django apps).
* **View:**
  + Handles the request-response cycle, processes user input, interacts with models, and passes data to templates.
  + Often implemented as Python functions or class-based views.
  + Examples: functions/classes in views.py files.
* **Template:**
  + Responsible for presenting data to the user.
  + Uses Django's built-in templating language to dynamically generate HTML.
  + Examples: .html files in an app's templates directory.

**SQLite Database Interaction:**Django's Object-Relational Mapper (ORM) facilitates seamless interaction with the SQLite database. It allows developers to define database tables as Python classes (Models) and interact with database records using Python code rather than raw SQL.

**Django Templates for UI:**Django template files (.html files) contain a mix of HTML and Django template tags/variables. The template language is processed on the server to dynamically generate the final HTML that is sent to the client's browser. This allows for data pulled from the models via views to be presented in the templates.dels via views to be presented in the templates.

## **4. Setup and Installation**

To set up and run Divido locally, follow these steps:

**Prerequisites:**

* Python (version 3.8+ recommended)
* Django (version X.X.X, check requirements.txt)
* SQlite(version 3.31.1+ recommended (bundled with Python 3.8+))
* pip (Python package installer)

**Installation Steps:**

**1.Clone the Repository:** git clone <repository\_url>

cd <project\_directory>

**2.Create a Virtual Environment (Recommended):** python3 -m venv venv

source venv/bin/activate # On Windows: venv\Scripts\activate

**3.Install Python Dependencies:** pip install -r requirements.txt

# This file should contain:

# Django==5.2.3

4.**Configure Database:**

Open divido/settings.py.

# your\_project\_name/settings.py (example)

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': BASE\_DIR / "db.sqlite3",

}

}

**5.Apply Migrations (Create Database Schema):** python manage.py migrate

**6.Create a Superuser (for Admin Panel access):** python manage.py createsuperuser

Follow the prompts to set up username, email, and password.

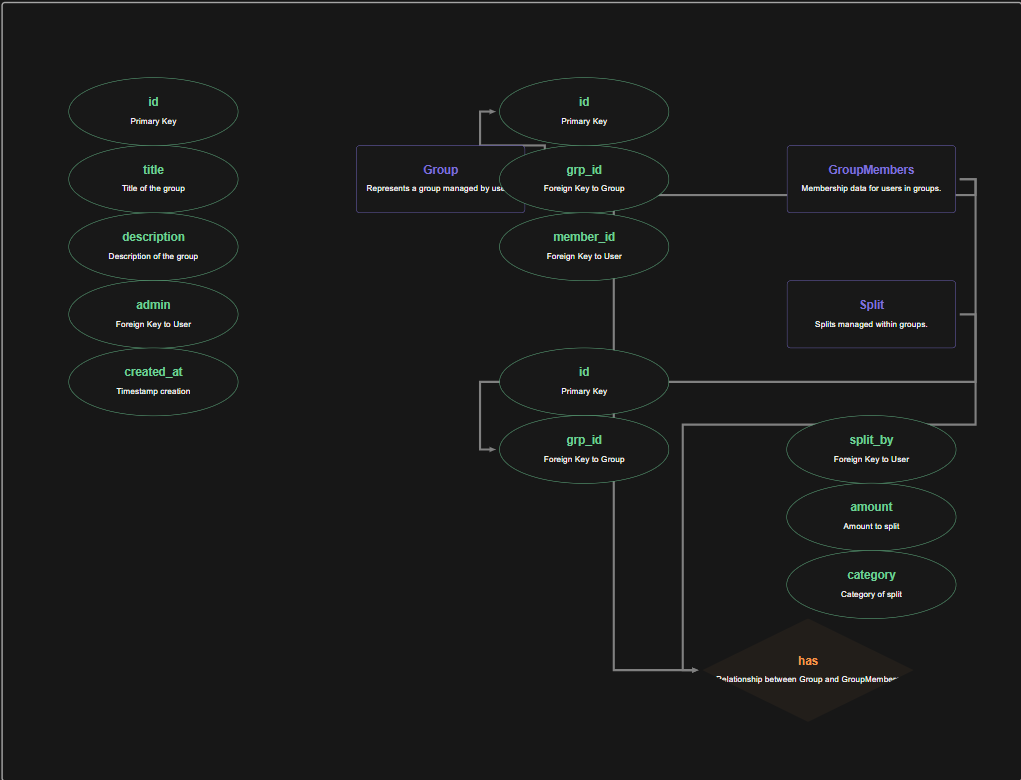
**7.Start the Django Development Server:** python manage.py runserver

8.**Access the Application:**

Open your web browser and navigate to<http://localhost:8000>.

Access the Admin Panel at<http://localhost:8000/admin/>.

## **5. Database Schema**



The core database tables and their relationships are defined using Django models in the models.py file of the app.These models represent entities like Group, GroupMembers, Split, Payer, and Settlement, which form the foundation of the application's data structure.The schema is designed to be compatible with SQLite (or any other supported database), ensuring portability and ease of setup.

* **user**: handled implicitly using Django’s built-in auth.User model for authentication and user-related data.
* **groups**: Stores group details including title, description, admin, and creation timestamp.
  + id (PK)
  + title
  + description
  + admin (FK to users)
  + created\_at
* **group\_members**: Maps users to groups they belong to.
  + id (PK)
  + grp\_id (FK to groups)
  + member\_id (FK to users)
  + (Unique together: grp\_id + member\_id)
* **splits**: Records expenses made in a group and who paid them.
  + id (PK)
  + grp\_id (FK to groups)
  + split\_by (FK to users)
  + amount
  + description
  + category (Choice field: Food, Travel, Rent, etc.)
  + fully\_paid (Boolean)
  + created\_at
* **payers**: Tracks how much each user owes for a split and if it’s paid.
  + id (PK)
  + split\_id (FK to splits)
  + payer\_id (FK to users)
  + amount
  + paid (Boolean)
  + (Unique together: split\_id + payer\_id)
* **settlements**: Logs payment transactions made against dues.
  + id (PK)
  + payer (FK to payers)
  + amount\_paid
  + paid\_on

## **6. Key Models, Views, and URLs**

### **Models ((core/models.py)**

* **User**: Uses Django’s built-in auth.User model for handling authentication, group membership, splits, and settlements.
* **Group**: Represents a group created by a user. Contains fields for title, description, admin (ForeignKey to **User**), and creation timestamp.
* **GroupMembers**: Manages the many-to-many relationship between **User** and **Group**. Ensures unique membership and supports member-specific actions.
* **Split**: Represents an expense split in a group. Includes category, total amount, who split it (**User**), and whether it's fully paid.
* **Payer**: Tracks individual user contributions/dues for a specific **Split**. Stores amount, payment status, and links to both the **Split** and **User**.
* **Settlement**: Logs payments made toward splits. Tied to a **Payer**, records amount paid and timestamp.

### **Views (core/**[**views.py**](http://views.py)**)**

* **Authentication Views**: Handle user login, signup with OTP verification, and password reset via OTP (login\_view, signup\_view, verify\_otp\_view, logout\_view, forgot\_password\_request\_view, verify\_reset\_otp\_view, reset\_password\_view).
* **Dashboard View**: Displays a financial overview for the user (dashboard\_view).
* **Group Views**:
  + groups\_view: Shows all groups the user is part of, with filtering and sorting.
  + create\_group\_view: Allows users to create a new group.
  + group\_detail\_view: Shows details and all splits in a specific group.
  + group\_info\_view: Displays group members, admin actions like add/remove members.
  + leave\_group\_view: Lets a user leave a group (after settling dues).
  + delete\_group\_view: Admin can delete a group (after dues are cleared).
* **Split Views**:
  + create\_split\_view: Creates a new expense split within a group.
  + split\_detail\_view: Shows who owes what for a split and payment tracking.
* **Settlement View**:
  + record\_settlement\_view: Records payment towards a user’s share in a split.
* **Transaction View**:
  + transactions\_view: Displays all payments involving the user.
* **Dues View**:
  + dues\_view: Displays net balances (to pay / to be paid) for each user, group-wise or overall.

### **URL Configuration (core/urls.py and divido/urls.py)**

* **divido/urls.py**: The main URL dispatcher that includes URLs from individual apps.  
  # divido/urls.py

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls), # Admin panel

path('', include('core.urls')), # Core app URLs

]

* **core/urls.py**: Defines URL patterns for the core app, including routes for authentication, group management, splits, dues, transactions, and settlements

# core/urls.py

# core/urls.py

from django.urls import path

from . import views

from django.contrib.auth.views import LogoutView

from django.contrib.auth.decorators import login\_required

urlpatterns = [

path('', views.home\_view, name='home'),

path('signup/', views.signup\_view, name='signup'),

path('verify-otp/', views.verify\_otp\_view, name='verify\_otp'),

path('login/', views.login\_view, name='login'),

path('logout/', login\_required(LogoutView.as\_view(next\_page='login')), name='logout'),

path('dashboard/', views.dashboard\_view, name='dashboard'),

path('transactions/', views.transactions\_view, name='transactions'),

path('groups/', views.groups\_view, name='groups'),

path('groups/create/', views.create\_group\_view, name='create\_group'),

path('groups/<int:group\_id>/', views.group\_detail\_view, name='group\_detail'),

path('groups/<int:group\_id>/create-split/', views.create\_split\_view, name='create\_split'),

path('groups/<int:group\_id>/info/', views.group\_info\_view, name='group\_info'),

path('groups/<int:group\_id>/add/', views.add\_member\_view, name='add\_member'),

path('groups/<int:group\_id>/remove/', views.remove\_member\_view, name='remove\_member'),

path('groups/<int:group\_id>/leave/', views.leave\_group\_view, name='leave\_group'),

path('groups/<int:group\_id>/delete/', views.delete\_group\_view, name='delete\_group'),

path('splits/<int:split\_id>/', views.split\_detail\_view, name='split\_detail'),

path('splits/<int:split\_id>/record-settlement/', views.record\_settlement\_view, name='record\_settlement'),

path('dues/', views.dues\_view, name='dashboard\_dues'),

path('groups/<int:group\_id>/dues/', views.dues\_view, name='group\_dues'),

]

## **7. Templates (Django Templates)**

All templates are located in the core/templates/ directory.

* **home.html**: Landing page where the name of the project is displayed.
* **login.html**, **signup.html**, **verify\_otp.html**: Handle user authentication and OTP verification.
* **groups.html**: Main page shown after login; displays the user’s groups.
* **create\_group.html**: Allows users to create a new group.
* **group\_detail.html**: Shows all expenses within a group and their payment status.
* **group\_info.html**: Displays group information including list of members and admin controls (add/remove members, delete group).
* **split\_detail.html**: Displays individual split details and settlements.
* **dashboard.html**: Shows overall financial summary (to pay / to be paid).
* **dues\_list.html**: Displays detailed dues by user or group.
* **transactions.html**: Lists all settlements made or received.

**Django Template Syntax Examples:**

* **Displaying Dynamic Content**

<h1>Welcome, {{ user.username }}!</h1>

<p>Amount Paid: ₹{{ t.amount\_paid|floatformat:2 }}</p>

* **Conditional Statements**

{% if is\_admin %}

<div class="admin-actions">

<!-- Admin-specific forms and controls -->

</div>

{% endif %}

* **Loops for Listing Data**

{% for t in transactions %}

<tr>

<td>{{ t.payer.split\_id.grp\_id.title }}</td>

<td>{{ t.paid\_on|date:"d M Y" }}</td>

</tr>

{% empty %}

<tr><td colspan="6">No transactions found.</td></tr>

{% endfor %}

* **Forms with CSRF Protection**

<form method="POST" action="{% url 'add\_member' group.id %}">

{% csrf\_token %}

<select name="user\_to\_add">

<option value="">Add a new member</option>

{% for user in users\_not\_in\_group %}

<option value="{{ user.id }}">{{ user.username }}</option>

{% endfor %}

</select>

<button type="submit">Add</button>

</form>

* **Template URL Tag**

<a href="{% url 'dashboard' %}">← back to dashboard</a>

## **8 .Application Pages**

### Home

### Signup

**Inputs**

* Username required
* Email required
* Password required
* Confirm password required

### Login

**Inputs**

* Email or Username required
* Password required

### Verify OTP

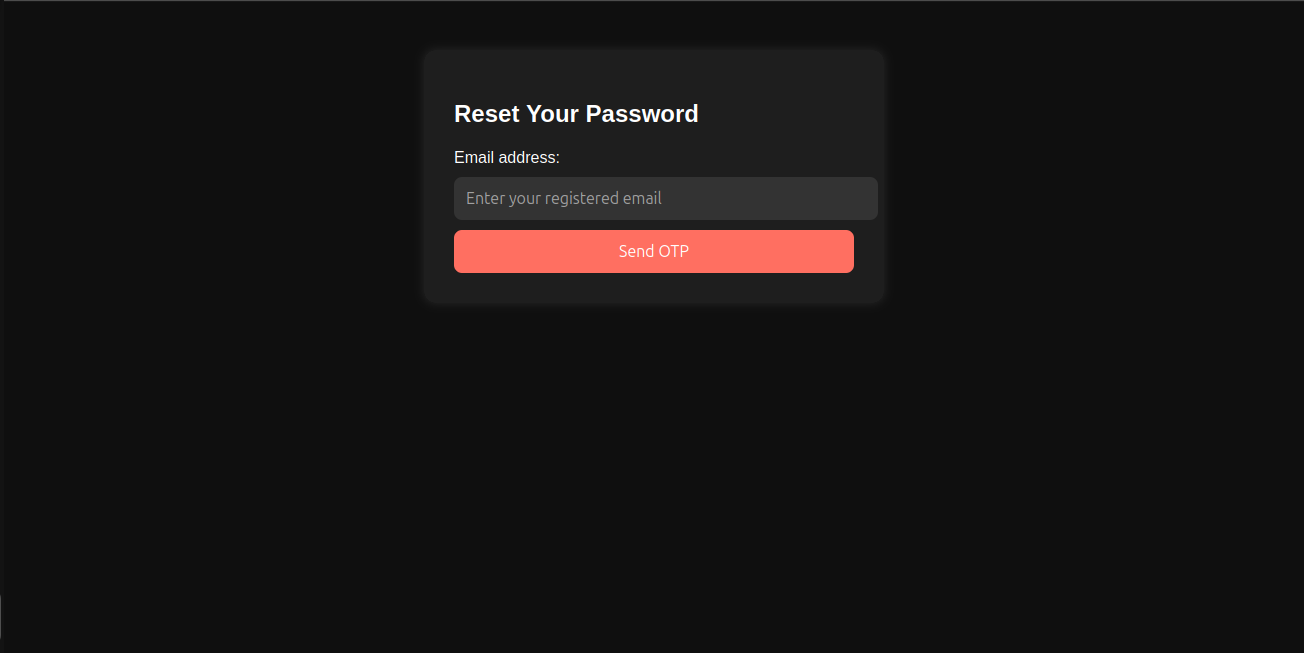
### 

**Inputs**

### OTP required

### 

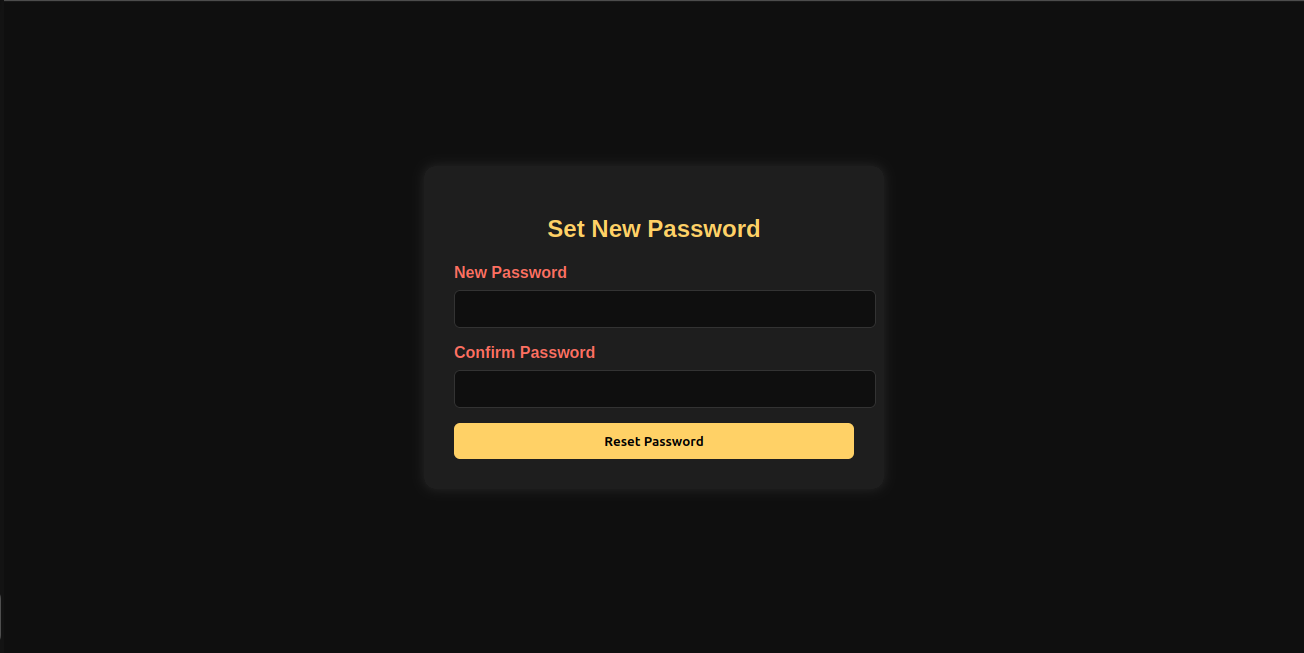
### Forgot password



**Inputs**

* Email required

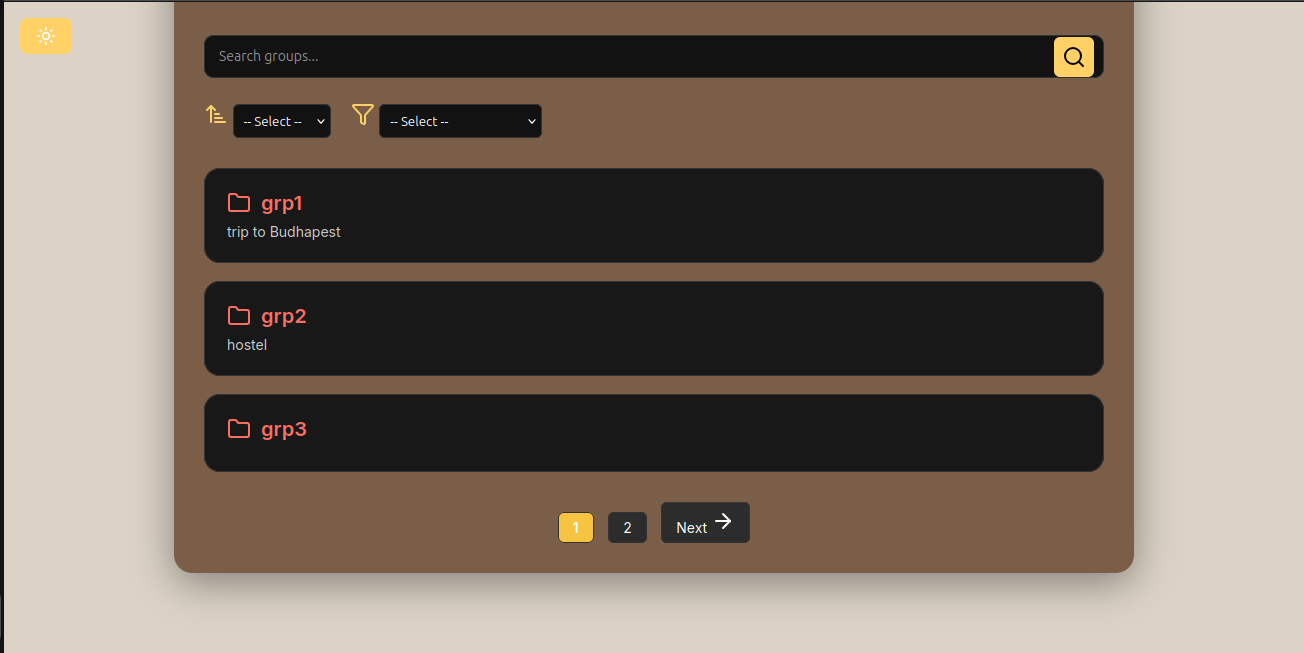
### Reset password



**Inputs**

* New password required
* Confirm password required

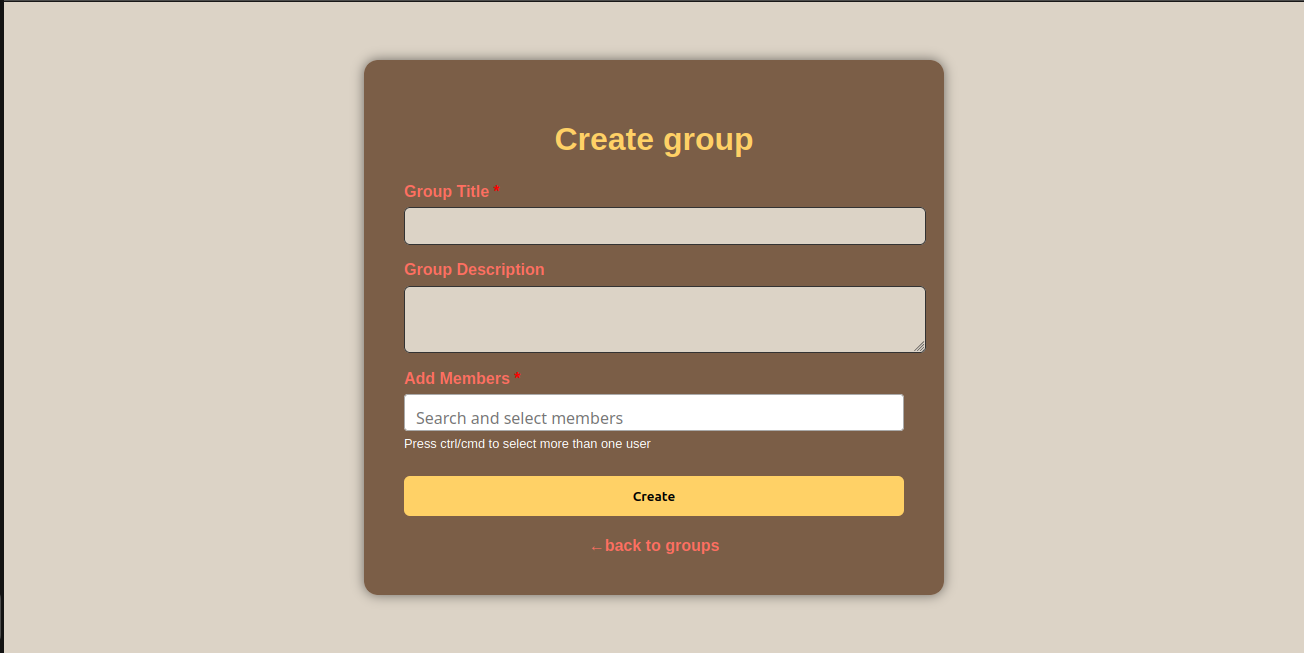
### Groups



**Inputs**

* Search groups optional
* Sort optional
* Filter optional

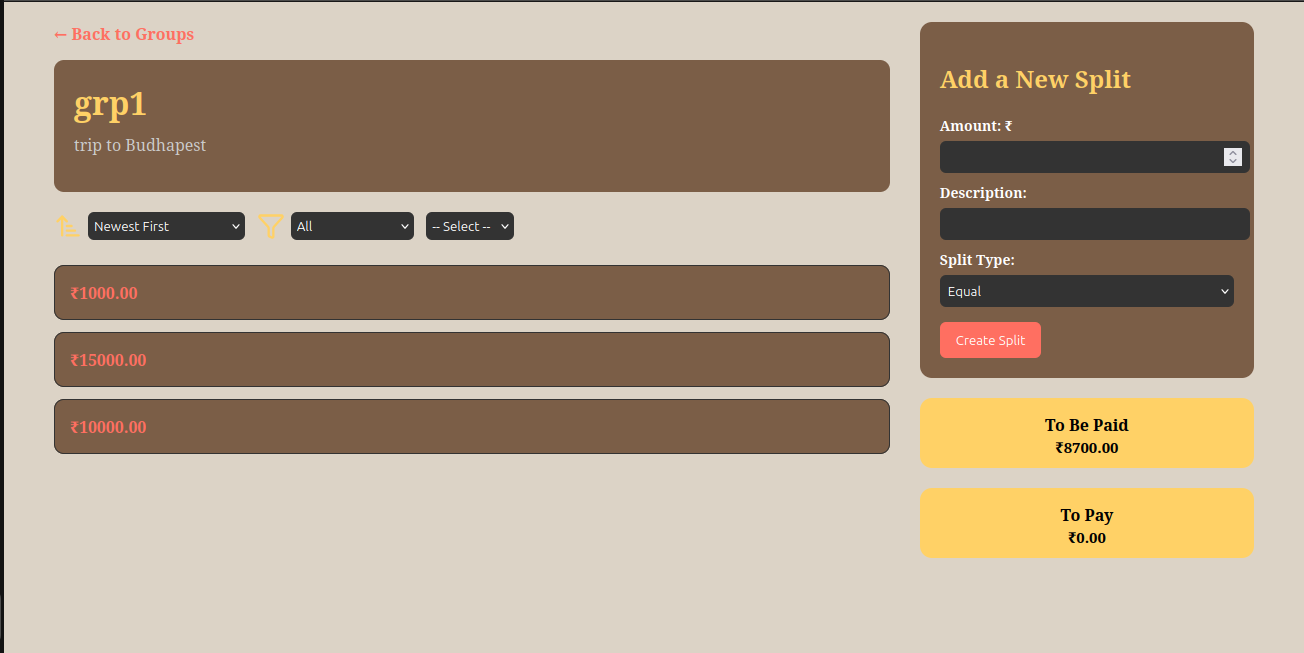
### Create Group



**Inputs**

* Group title required
* Group description optional
* Add members required

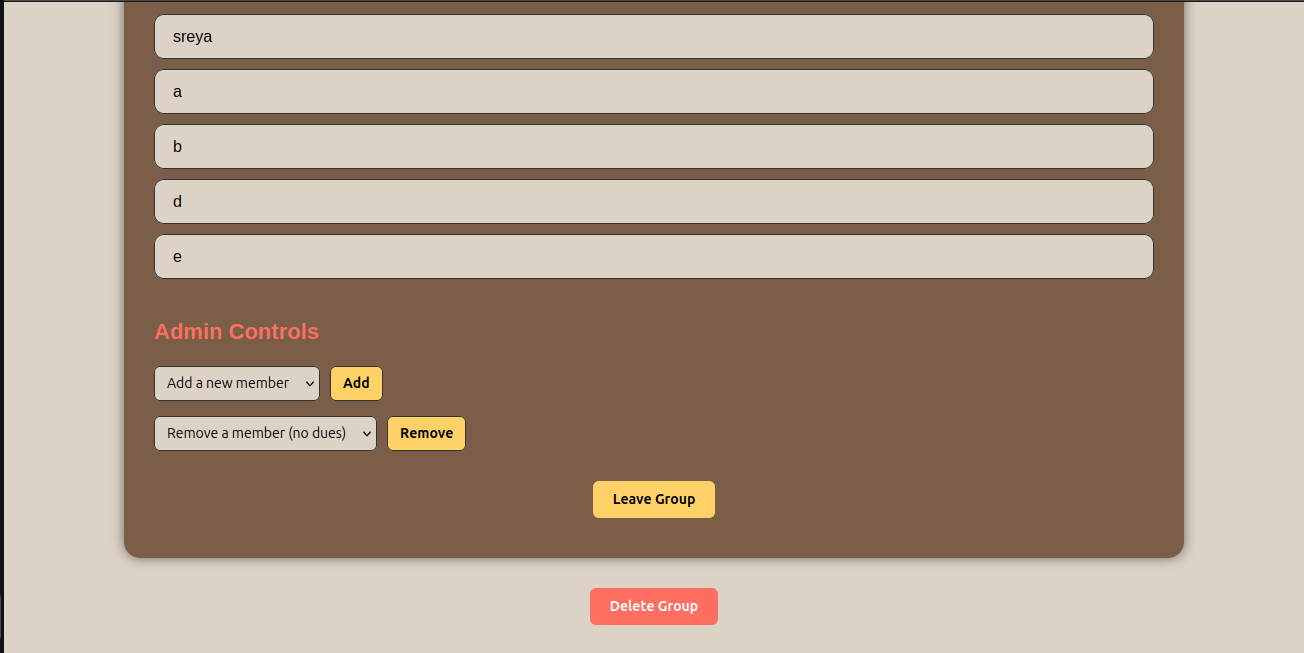
### Group Detail



**Inputs**

* Amount required
* Description optional
* Split type required

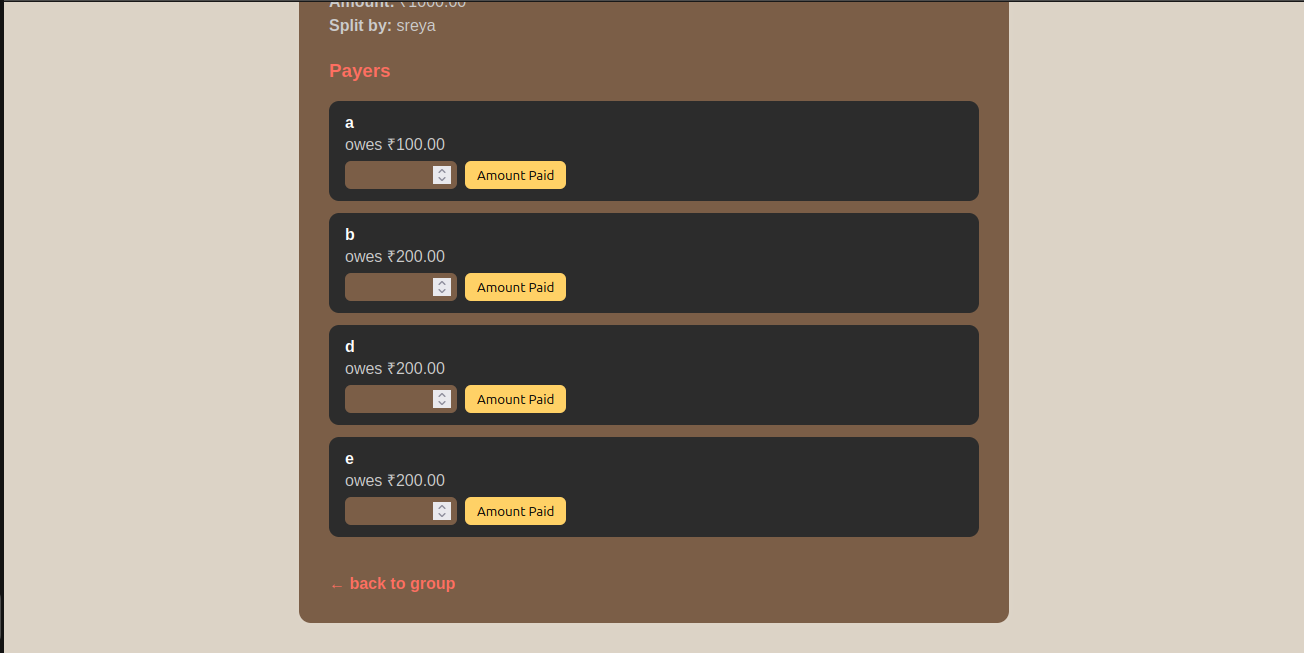
### Group Info



**Inputs**

* Search members optional
* Add a new member optional
* Remove a member optional

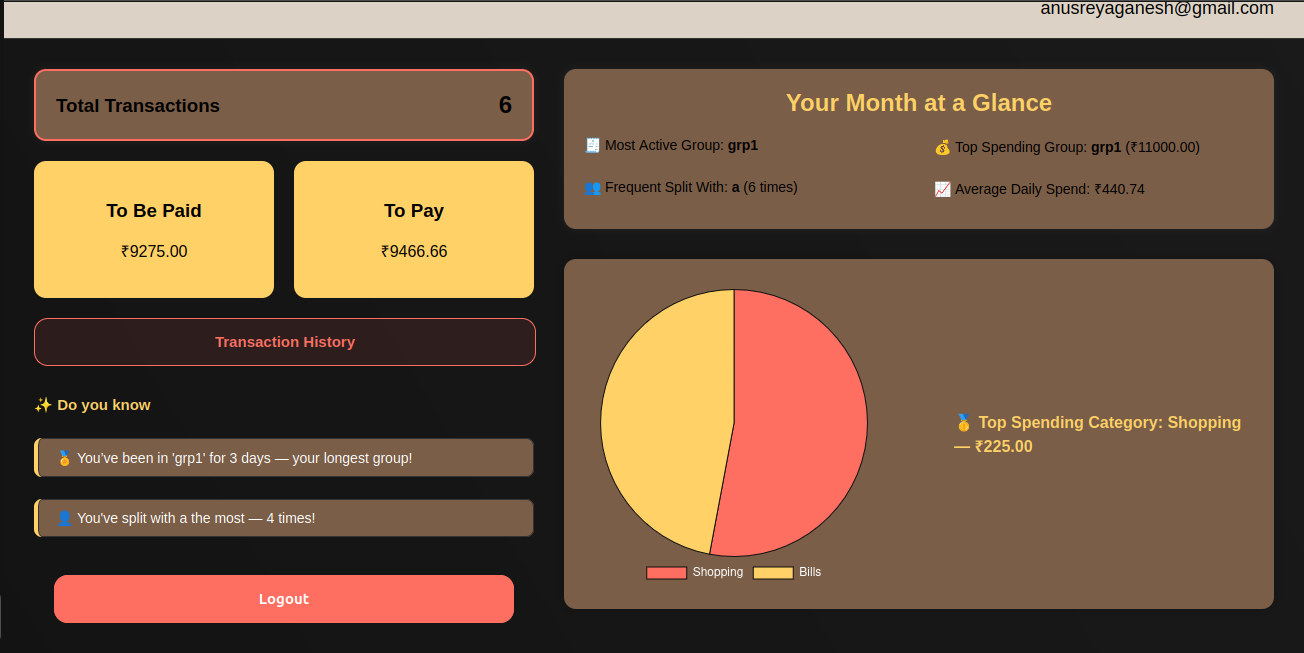
### Split Detail



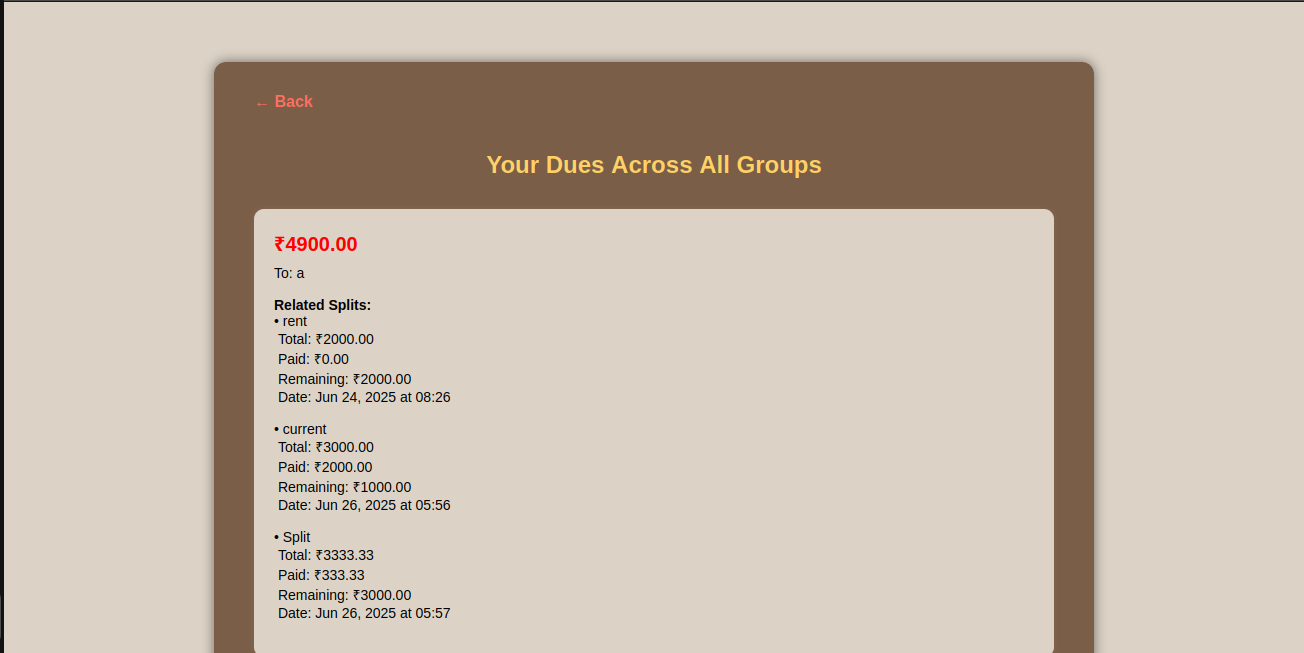
**Inputs**

* Amount paid optional

### Dashboard



### Dues List

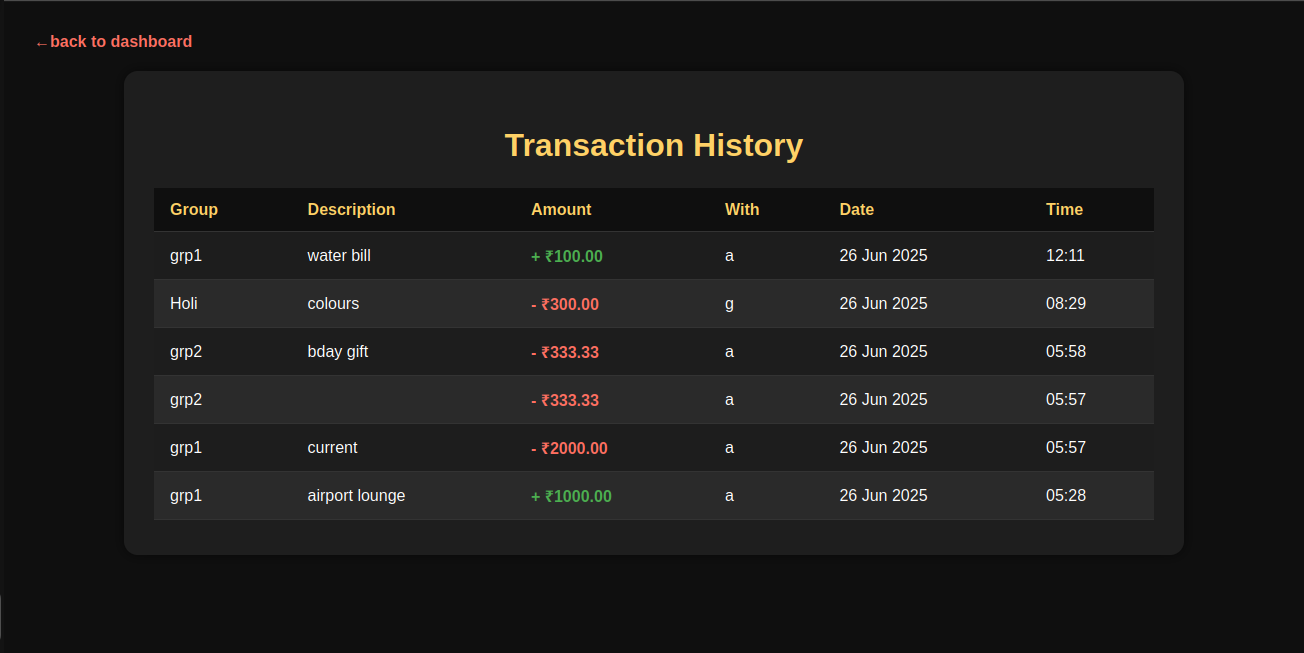


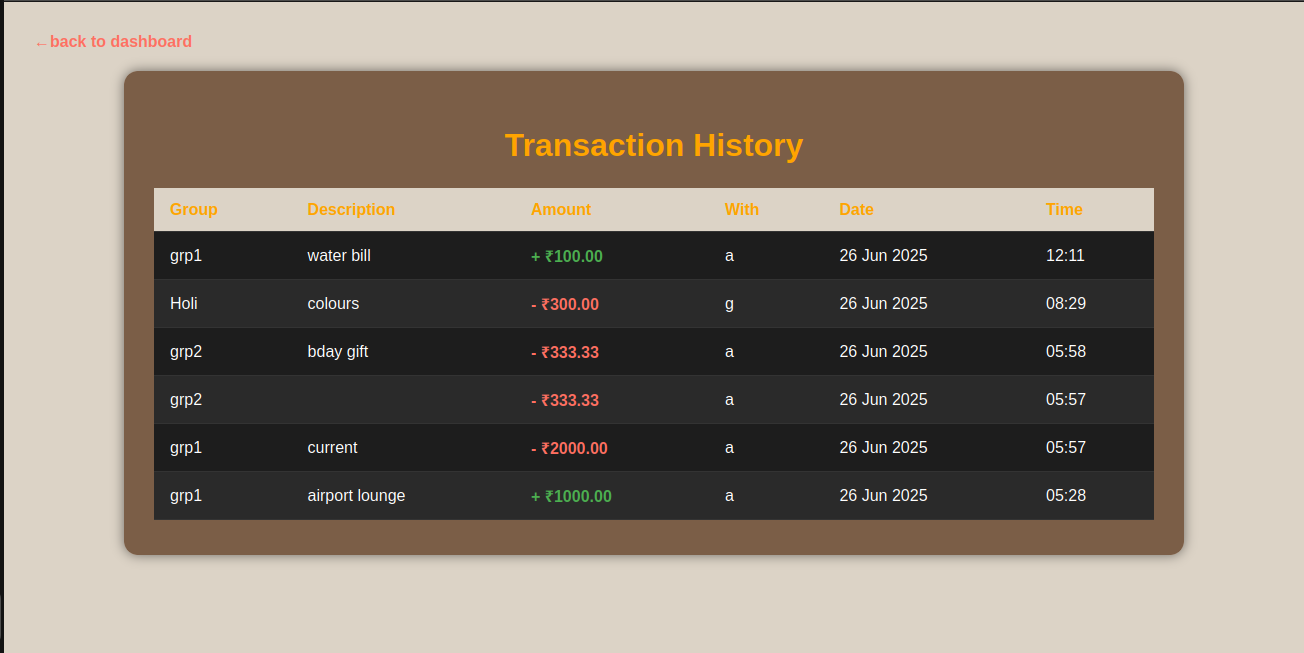
### 

### 

### 

### Transactions





## 

## 

## 

## **9. Future Enhancements**

* **Payment Gateway Integration:** Integrate UPI or payment gateway services (e.g., Razorpay, Stripe) to allow users to settle dues directly within the app.
* **Image Uploads:** Enable profile picture uploads or receipt attachments for splits using Django's ImageField and a proper media storage setup.
* **User Reviews & Ratings System:** Allow members to give feedback on group reliability or trustworthiness, especially useful in larger shared expense communities.
* **Notifications System**:

Implement in-app or email notifications new expenses added, dues assigned, settlements received.

* **Recommendation Engine:** Suggest optimal split methods or recurring expense templates based on previous group activity and patterns.
* **Performance Optimization:** Optimize database queries, apply caching techniques (e.g., Redis), and improve load times for larger datasets.
* **Security Enhancements:** Add features like two-factor authentication (2FA), stricter session handling, and better password policies for improved account protection.

This documentation provides a foundational understanding of the Group Expense Splitter Web App built with Django. For more detailed information, please refer to the codebase and comments within individual files.