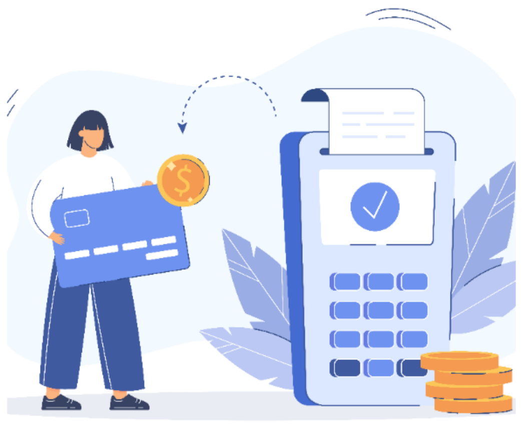
Utility Pay

Utility Payment Management System

Abstract:



UtilityPay is a full-stack web application developed using Flask, SQLAlchemy, and SQLite that allows users to manage and pay their utility bills securely and efficiently. The system provides authentication via OTP-based email verification using the Brevo API, integrates Razorpay for real-time online payments, and offers role-based access for users and administrators. The admin can generate bills, manage users, and track all transactions, while users can make payments, download receipts, and view transaction history. The platform features a responsive and user-friendly dashboard, secure data handling, and real-time updates, ensuring a seamless payment management experience.

Table of Contents

1. Abstract
2. Introduction
3. Objectives
4. System Overview
5. Features
6. System Architecture
7. Technology Stack
8. Modules Description
    i. Authentication Module
    ii. User Module
    iii. Admin Module
    iv. Payment Module
    v. Email OTP Verification Module
    vi. PDF Generation Module
9. Database Design
10. Implementation Details
11. User Interface Overview
12. Testing and Validation
13. Security Features
14. Future Enhancements
15. Conclusion
16. References

3. Introduction

UtilityPay was designed to simplify and automate the process of paying and managing utility bills such as electricity, water, and broadband. Many users face issues with late payments, lack of transaction tracking, and non-centralized billing. UtilityPay resolves these challenges by providing an integrated web-based platform where users can register, verify via email, view their bills, pay securely using Razorpay, and download payment receipts instantly.

The system also allows administrators to manage users, generate bills, and monitor all payments in one centralized dashboard. It ensures secure communication, efficient data storage, and a mobile-responsive interface that works seamlessly across devices.

4. Objectives

- To develop a secure, user-friendly platform for managing utility bill payments.
- To implement a reliable OTP-based authentication system using Brevo API.
- To integrate Razorpay for smooth and real-time payment transactions.
- To create a centralized admin dashboard for bill management and monitoring.
- To enable users to download and maintain digital payment receipts.
- To ensure responsive design and scalable backend structure for future enhancements.

5. System Overview

UtilityPay consists of two major roles: **User** and **Admin**.

- **User:** Can register, verify email, view bills, make payments, and view/download transaction history.

- **Admin:** Can generate monthly or custom bills, manage users, view reports, and handle all transactions.

The backend is powered by **Flask**, using **SQLAlchemy ORM** for database operations and **SQLite** as the storage layer. Email verification and OTP are handled through the **Brevo API**, while **Razorpay API** manages real-time payments. The frontend is designed using **HTML, CSS, and JavaScript** for an interactive experience, including AJAX for asynchronous requests.

6. Features

- Secure user authentication (login/register with OTP verification)
- Real-time email OTP via Brevo API
- Razorpay payment gateway integration
- Transaction history and recent payment tracking
- Admin dashboard for managing users and bills
- Dynamic total payment calculation
- PDF generation for payment receipts
- Responsive and mobile-friendly design
- CSV and PDF export for reports
- Role-based access control for users and admins

7. System Architecture

The system follows a **client-server architecture**.

- **Client Side:** Built with HTML, CSS, and JavaScript. Handles form inputs, AJAX requests, and updates the dashboard dynamically.
- **Server Side:** Flask handles routes, authentication, and database queries. Razorpay is used for payment processing, and Brevo is used for sending OTPs.

- **Database Layer:** SQLite manages structured data like users, transactions, and bills via SQLAlchemy ORM.

The architecture ensures modular design and scalability for future API-based integration or cloud deployment.

8. Technology Stack

Frontend: HTML, CSS, JavaScript

Backend: Python (Flask Framework)

Database: SQLite with SQLAlchemy ORM

Authentication: Email + OTP via Brevo API

Payment Integration: Razorpay API

Email Service: Flask-Mail

PDF Generation: ReportLab

Tools & Libraries: Flask, SQLAlchemy, Razorpay SDK, smtplib, jsonify, WTForms, and Jinja2

9. Modules Description

a. Authentication Module

Handles user registration, login, and email verification. Users register with an email and receive an OTP from Brevo for verification before accessing their dashboard. Tokens and sessions are securely managed using Flask sessions.

b. User Module

Enables users to log in, view their billing history, make payments through Razorpay, and download receipts. Displays the last five transactions with dynamic total amount updates.

c. Admin Module

Admins can log in with admin credentials, generate and assign bills, monitor all payments, and view reports. They can also download reports in PDF/CSV format.

d. Payment Module

Integrates Razorpay for secure online payments. Generates unique order IDs, handles callbacks, and confirms payment success before updating the transaction table.

e. Email OTP Verification Module

Uses Brevo API to send OTPs to users' registered emails for secure verification during signup and password reset. The OTP is validated server-side using Flask routes.

f. PDF Generation Module

After successful payments, users can download receipts generated using the ReportLab library. Each PDF includes user details, transaction ID, amount, and date.

10. Database Design

The database includes the following key tables:

1. **Users Table:**

- id (Primary Key)
- name
- email
- password (hashed)
- is_verified (boolean)
- role (user/admin)

2. **Transactions Table:**
    - id (Primary Key)
    - user_id (Foreign Key)
    - amount
    - transaction_id
    - date_time
    - payment_status
3. **Bills Table:**
    - id (Primary Key)
    - user_id
    - month
    - amount
    - status (paid/unpaid)

11. Implementation Details

- **main.py** connects all modules and initializes Flask app, routes, and database.
- **auth_routes.py** handles login, register, and OTP verification.
- **models.py** defines database schemas.
- **dbb.py** initializes database connection using SQLAlchemy.
- **dashboard.html** displays user interface elements and renders data from the backend.
- **script.js** manages client-side interactivity, AJAX calls, and payment triggers.

The Flask server runs locally or in production via WSGI and handles all CRUD operations securely.

12. User Interface Overview

- **Login/Register Page:** Simple form with email and password, OTP section for verification.
- **Dashboard:** Displays total payments, last five transactions, and “Download Receipt” option.
- **Admin Panel:** Includes options to create bills, view users, and track overall payments.
- **Responsive Design:** Works on both desktop and mobile screens seamlessly.

13. Testing and Validation

The application was tested for:

- Email verification flow (Brevo OTP correctness).
- Razorpay sandbox and live transactions.
- Database integrity and transaction consistency.
- UI responsiveness on various screen sizes.
- Edge cases (duplicate entries, invalid OTP, payment failures).

All tests passed with successful results in both user and admin workflows.

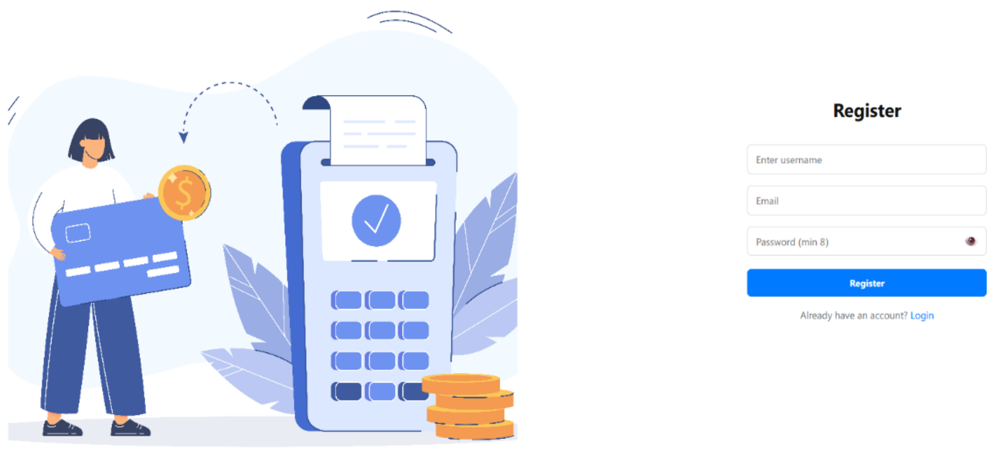
14. Security Features

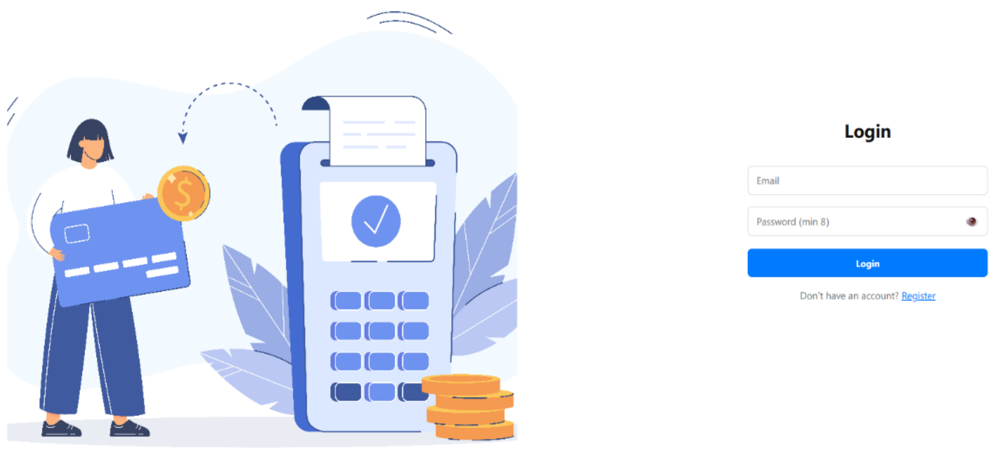
- OTP-based authentication and email verification.
- Encrypted password storage.
- Razorpay secure transaction gateway.
- Session-based login management.
- Input validation on both client and server sides.

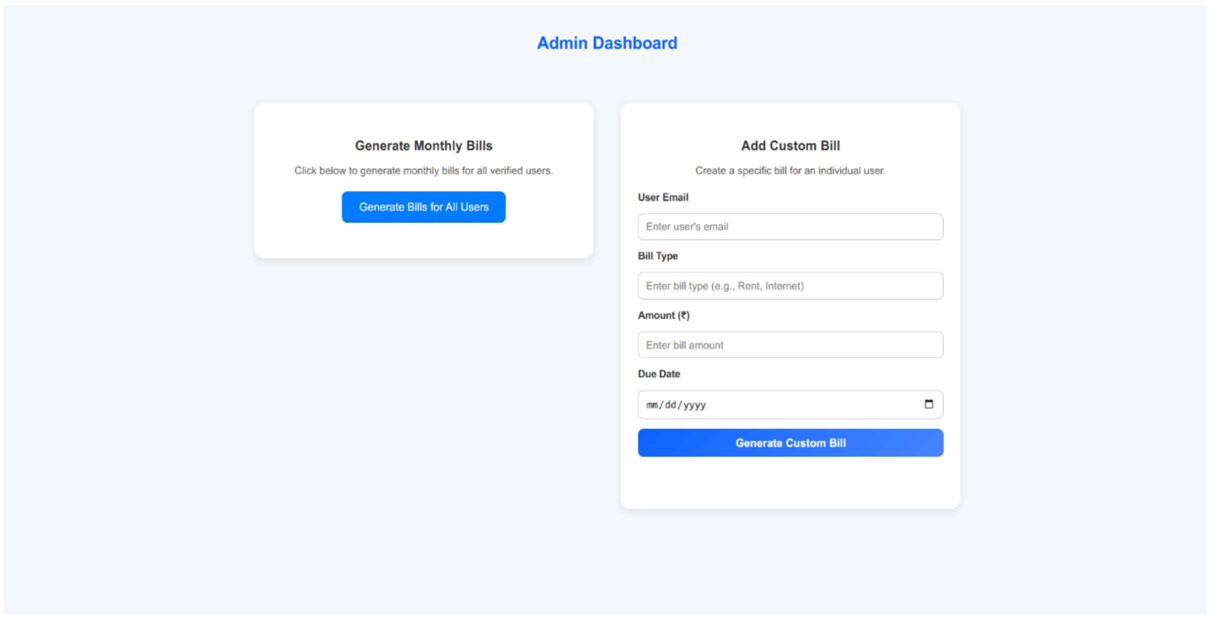
15. Future Enhancements

- Multi-payment gateway integration.
- SMS-based OTP in addition to email.
- AI-based bill prediction and analytics.
- Cloud deployment on AWS or GCP with CI/CD pipeline.
- Advanced reporting dashboards using React frontend and REST API backend.

Screenshots:







Admin Dashboard

Generate Monthly Bills

Click below to generate monthly bills for all verified users.

Add Custom Bill

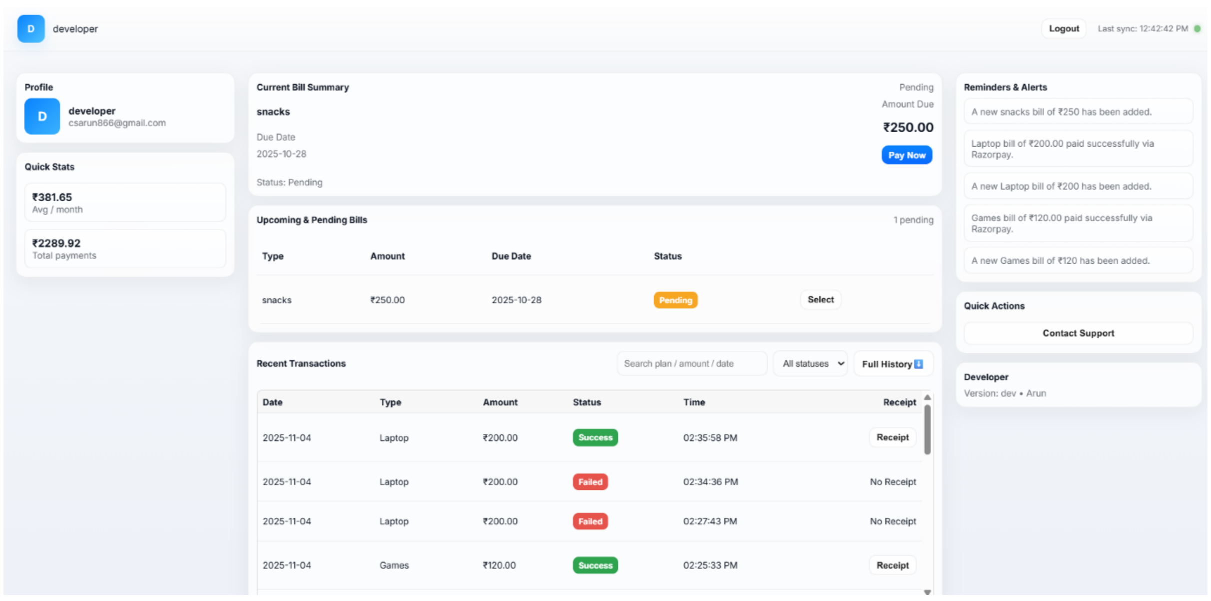
Create a specific bill for an individual user.

User Email: Enter user's email

Bill Type: Enter bill type (e.g., Rent, Internet)

Amount (₹): Enter bill amount

Due Date: mm/dd/yyyy



Profile

developer
csarun866@gmail.com

Quick Stats

₹381.65
Avg / month

₹2289.92
Total payments

Current Bill Summary

snacks

Due Date
2025-10-28

Status: Pending

Pending
Amount Due
₹250.00

Upcoming & Pending Bills — 1 pending

Type	Amount	Due Date	Status	
snacks	₹250.00	2025-10-28	Pending	Select

Recent Transactions

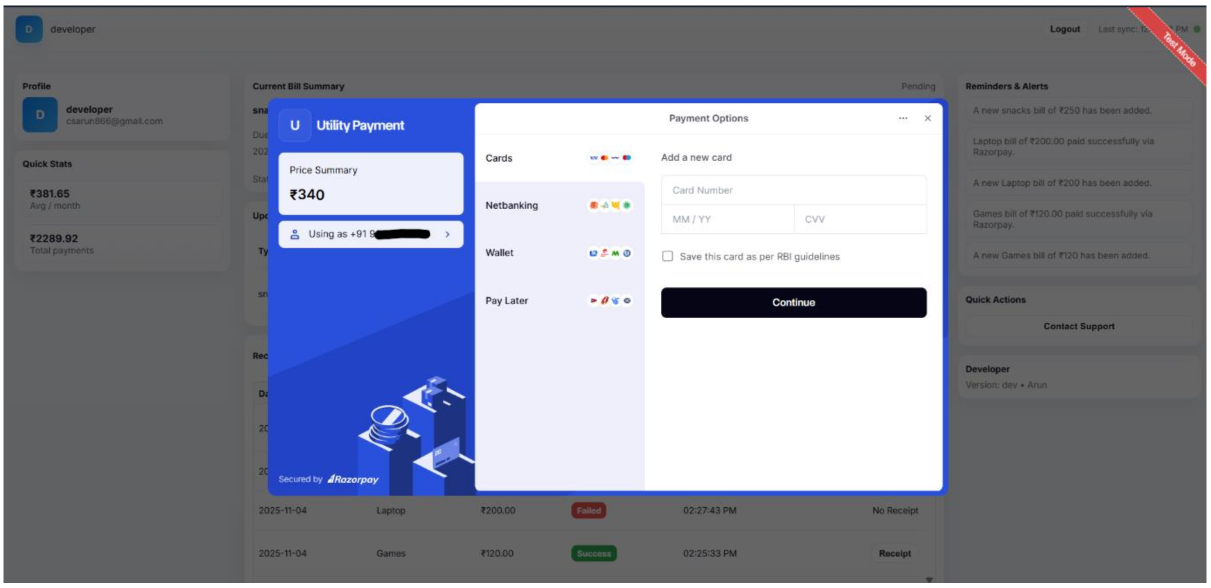
Date	Type	Amount	Status	Time	Receipt
2025-11-04	Laptop	₹200.00	Success	02:35:58 PM	Receipt
2025-11-04	Laptop	₹200.00	Failed	02:34:36 PM	No Receipt
2025-11-04	Laptop	₹200.00	Failed	02:27:43 PM	No Receipt
2025-11-04	Games	₹120.00	Success	02:25:33 PM	Receipt

Reminders & Alerts

- A new snacks bill of ₹250 has been added.
- Laptop bill of ₹200.00 paid successfully via Razorpay.
- A new Laptop bill of ₹200 has been added.
- Games bill of ₹120.00 paid successfully via Razorpay.
- A new Games bill of ₹120 has been added.

Quick Actions

Developer
Version: dev • Arun



U Utility Payment

Price Summary
₹340

Using as +91 9[redacted]

Secured by Razorpay

Payment Options

Add a new card

Card Number
MM / YY
CVV

Save this card as per RBI guidelines

16. Conclusion

UtilityPay successfully integrates authentication, billing, payment, and reporting functionalities into a single, secure web platform. It automates the payment workflow, ensures transparency for both users and admins, and provides an extendable base for future enhancements like mobile app integration and cloud scalability.

17. References

- Flask Documentation – https://flask.palletsprojects.com
- Razorpay Developer Docs – https://razorpay.com/docs
- Brevo (Sendinblue) API – https://developers.brevo.com
- SQLAlchemy ORM – https://docs.sqlalchemy.org