URS Web Application Documentation

1 Overview

The Unified Rewards System (URS) web application is a robust platform designed to streamline customer loyalty programs for businesses. It enables vendors to **register seamlessly, manage customer transactions efficiently, and gain valuable insights through data analytics**. The backend is developed using Flask, a lightweight and scalable framework, while Firebase Firestore serves as the database, offering real-time data synchronization and cloud storage.

2 Project Structure

The URS project is structured into multiple directories, ensuring a modular and scalable architecture. The directory structure is as follows:

```
urs-deployed-main/
|-- app/
|   |-- __init__.py      # Initializes the Flask application
|   |-- models.py        # Defines database models
|   |-- routes.py        # Handles application endpoints
|   |-- static/
|   |   |-- css/
|   |   |   `-- style.css  # Styling for frontend pages
|   |   |-- js/
|   |   |   |-- dashboard.js  # Manages dashboard interactivity
|   |   |   `-- firebase_config.js  # Configures Firebase authentication
|   |-- templates/
|   |   |-- business_model.html  # Vendor registration page
|   |   |-- business_model_confirmation.html  # Confirmation page after registration
|   |   |-- check_customer.html  # Validates customer accounts
|   |   |-- dashboard.html  # Displays vendor dashboard
|   |   |-- export_pdf.html  # Handles PDF exports
|   |   |-- login.html  # User authentication page
|   |   `-- register.html  # Vendor registration form
|-- firebase_config.py  # Configuration script for Firebase Firestore
|-- firebase-auth.json  # Stores Firebase authentication credentials
|-- index.py  # Entry point for running the Flask application
|-- Procfile  # Configuration file for deployment (e.g., Heroku)
|-- README.md  # Project documentation
|-- requirements.txt  # List of dependencies for Python packages
`-- test.py  # Unit tests for the application
```

3 Key Components

3.1 Initialization

The application initializes in the `app/__init__.py` file. This script sets up the Flask application instance, connects to Firestore, and registers blueprints for modular routing. The initialization ensures **a structured startup process** and **seamless integration with Firebase services**.

3.2 Models

Defined in `app/models.py`, these classes represent essential data structures. The **Vendor, Transaction, Customer, and VendorType** models allow structured data storage, making it easier to **fetch, update, and process records dynamically**.

3.3 Routes

The `app/routes.py` file contains all API routes that define how different parts of the application interact. These endpoints **handle vendor registration, login authentication, transaction processing, reward point management, and analytics retrieval**.

4 Detailed Description

4.1 Initialization

Upon launching the Flask server, the initialization script **configures Firestore, sets up security layers, and loads necessary modules**. This process ensures **smooth integration with authentication and database services**.

4.2 Models

- `Vendor` – Stores vendor details such as business type, UPI ID, and total transactions.
- `Transaction` – Logs customer purchases, including timestamps and reward points.
- `Customer` – Maintains customer profiles, tracking accumulated reward points.
- `VendorType` – Defines business categories and corresponding reward structures.

4.3 Routes

- `/login`: Handles user authentication and login validation.
- `/register`: Registers a new vendor and stores the details in Firestore.
- `/dashboard`: Displays metrics, reward points, and transaction trends.
- `/api/transactions`: Fetches transaction details for vendors.
- `/api/analytics`: Generates reports and insights based on sales data.
- `/api/export`: Exports data in CSV or PDF formats for business analysis.

5 Database Configuration

Database operations are configured in `firebase_config.py`. The script establishes **secure connections with Firestore, initializes database instances, and provides helper functions for CRUD operations**. All data is stored using Firestore's NoSQL document-based structure, ensuring **fast and scalable performance**.

6 Authentication

User authentication is managed through **session-based login mechanisms**. Passwords are securely stored using bcrypt hashing, and authentication tokens are used to maintain user sessions. The authentication system is designed to be **scalable, secure, and compliant with modern security standards**.

7 Dashboard

The dashboard acts as the **central hub for vendors**, offering real-time insights into sales, customer engagement, and reward point issuance. The dashboard queries the Firestore database to display:

- Total transactions processed
- Reward points issued and redeemed
- Revenue trends over a specified period
- Customer retention and loyalty metrics

8 API Endpoints

The web application provides **several RESTful API endpoints**, returning data in **JSON format** for seamless integration with other services. These APIs allow vendors to:

- Retrieve sales analytics
- Access customer transaction history
- Export business reports for financial tracking

Each endpoint follows best practices, ensuring **fast responses and high availability**.

9 Export Functionality

For vendors needing transaction reports, the application offers **export options in CSV and PDF formats**. The CSV export allows for **bulk data analysis**, while the PDF option generates **formatted summaries for quick reference**. This functionality is implemented using Python's `csv` and `pdfkit` libraries.

10 Conclusion

The URS web application serves as a **fully functional, scalable solution for managing customer loyalty programs**. Built with Flask and Firebase Firestore, it offers vendors **real-time analytics, secure transactions, and a streamlined business workflow**. By providing **seamless reward point tracking, transaction management, and business insights**, the system ensures an **efficient and rewarding experience** for both vendors and customers.