HOUSEHOLD SERVICES APP

Here is a link to my demo video of the app - link

Project Report: Role-Based Access Control (RBAC) Service Management Application

---

1. Project Overview

The RBAC-based Service Management application allows users to interact based on their designated roles (Admin, Service Professional, and Customer). The system ensures data privacy, efficient role management, and robust service operations through dedicated dashboards and automated jobs.

---

2. Objectives

- Implement role-based authentication and access control.
- Facilitate service creation, request handling, and user management.
- Automate notifications and reporting through backend jobs.
- Enhance API performance using caching techniques.

---

3. Features

3.1 User Management

Admin

- One Admin with elevated privileges based on the role.
- Manage customers and service professionals.
- Approve service professionals after profile verification.
- Block/unblock users based on activities or reviews.

Professional & Customer

- Register/Login using username, password, and role.
- JWT-based token authentication ensures secure access.

3.2 Admin Dashboard

- Access granted after Admin login.
- Manage all users and their statuses.
- Search professionals for blocking/unblocking/reviewing.
- Create, update, and delete service listings.

3.3 Service Management

Admin Actions

- Create new services with fields like name, base price, and required time.
- Modify service attributes (name, price, etc.).
- Remove existing services.

Customer Actions

- Search services based on location, name, or PIN code.
- Create/edit service requests with details like request date, remarks, and status.
- Close service requests when no longer needed.

Professional Actions

- View all customer service requests.
- Accept, reject, or close service requests after completion.

3.4 Notifications and Reports

Daily Reminders

- Send reminders to professionals who haven't accepted/rejected a request.
- Trigger reminders in the evening using Email.

Monthly Activity Report

- Generate HTML-based activity reports for customers on the 1st of every month.
- Include requested/closed service details and send via email.

CSV Export

- Professionals can download closed service requests as CSV.
- Admin can trigger this batch job and get notified on completion.

3.5 Caching and Performance

- Cache frequently accessed data (e.g., services and user roles).
- Implement cache expiry for data consistency.
- Optimize APIs to reduce latency.

---

4. Technology Stack

Backend

- **Framework**: Flask
- **Authentication**: JWT or Flask-Security
- **Database**: PostgreSQL/SQLite

Frontend

- **Framework**: React.js or plain HTML with templates

Scheduling and Jobs

- **Task Queue**: Celery
- **Scheduler**: Celery Beat/cron jobs
- **Mail/Alerts**: SMTP for emails, Google Chat webhooks for reminders

Performance

- **Caching**: Flask-Caching (Redis)

---

5. Implementation Plan

1. **Authentication and Role Setup**

    - Set up JWT-based authentication.

2. **Admin Dashboard**

    - Build user and service management features.
    - Integrate service listing APIs.

3. **Service Requests**

    - Create request handling for customers and professionals.

4. **Notifications & Reports**

    - Schedule reminders and monthly reports using Celery.
    - Integrate with Google Chat or SMTP.

5. **Performance Optimization**

    - Cache frequently accessed endpoints.
    - Monitor API response times and optimize.

6. **Testing and Deployment**

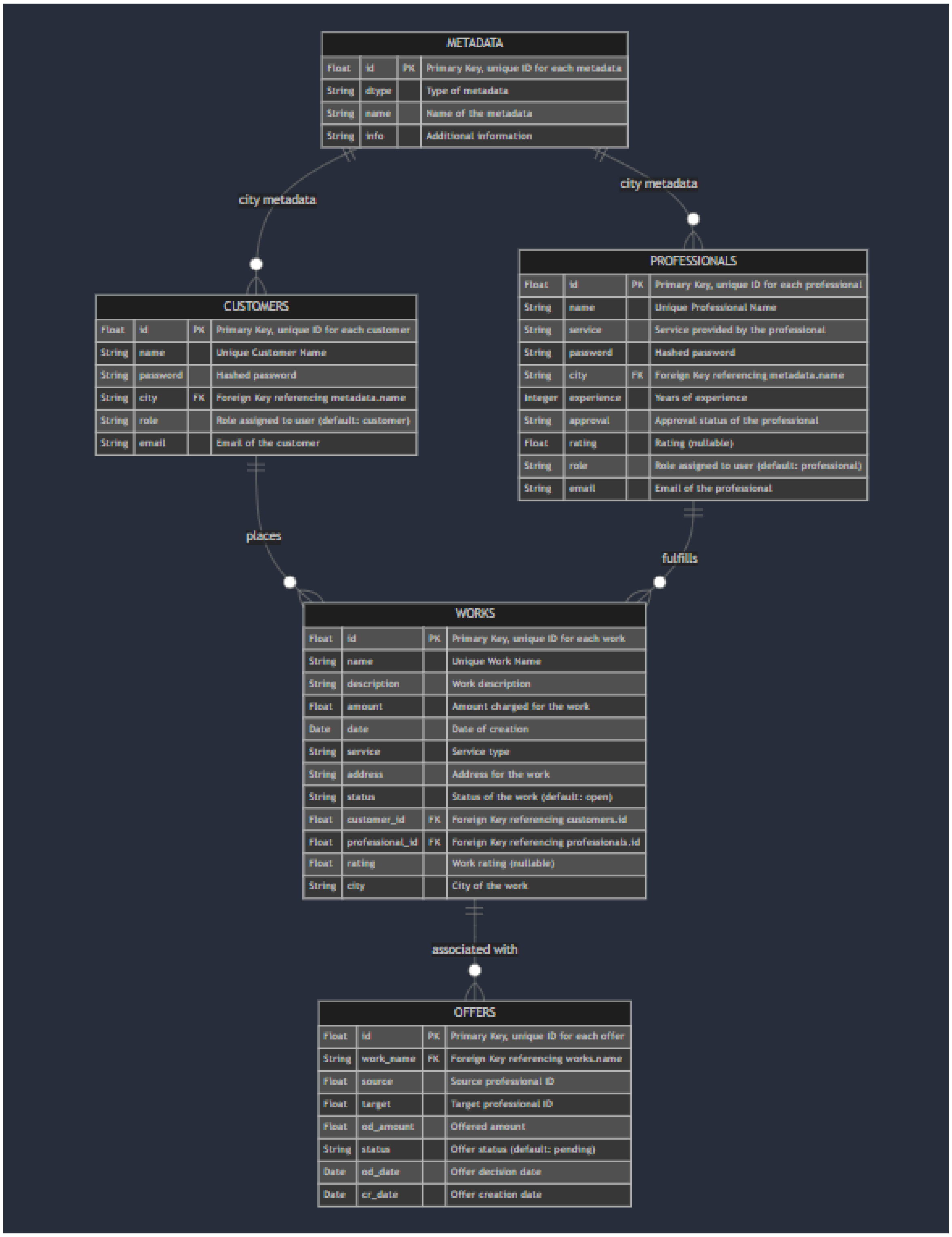
    - Perform unit and integration testing.
    - Deploy on a platform like AWS, Heroku, or GCP.

---

8. Conclusion

This RBAC-based application ensures secure, efficient, and streamlined service management for admins, customers, and service professionals. Automation features reduce manual effort while caching and performance enhancements improve user experience.

ER Diagram



METADATA

Float	id	PK	Primary Key, unique ID for each metadata
String	dtype		Type of metadata
String	name		Name of the metadata
String	info		Additional information

CUSTOMERS

Float	id	PK	Primary Key, unique ID for each customer
String	name		Unique Customer Name
String	password		Hashed password
String	city	FK	Foreign Key referencing metadata.name
String	role		Role assigned to user (default: customer)
String	email		Email of the customer

PROFESSIONALS

Float	id	PK	Primary Key, unique ID for each professional
String	name		Unique Professional Name
String	service		Service provided by the professional
String	password		Hashed password
String	city	FK	Foreign Key referencing metadata.name
Integer	experience		Years of experience
String	approval		Approval status of the professional
Float	rating		Rating (nullable)
String	role		Role assigned to user (default: professional)
String	email		Email of the professional

WORKS

Float	id	PK	Primary Key, unique ID for each work
String	name		Unique Work Name
String	description		Work description
Float	amount		Amount charged for the work
Date	date		Date of creation
String	service		Service type
String	address		Address for the work
String	status		Status of the work (default: open)
Float	customer_id	FK	Foreign Key referencing customers.id
Float	professional_id	FK	Foreign Key referencing professionals.id
Float	rating		Work rating (nullable)
String	city		City of the work

OFFERS

Float	id	PK	Primary Key, unique ID for each offer
String	work_name	FK	Foreign Key referencing works.name
Float	source		Source professional ID
Float	target		Target professional ID
Float	od_amount		Offered amount
String	status		Offer status (default: pending)
Date	od_date		Offer decision date
Date	cr_date		Offer creation date