Household Services Application - Project Report

Author

Vagadeeshwar G
21f1005494
21f1005494@ds.study.iitm.ac.in

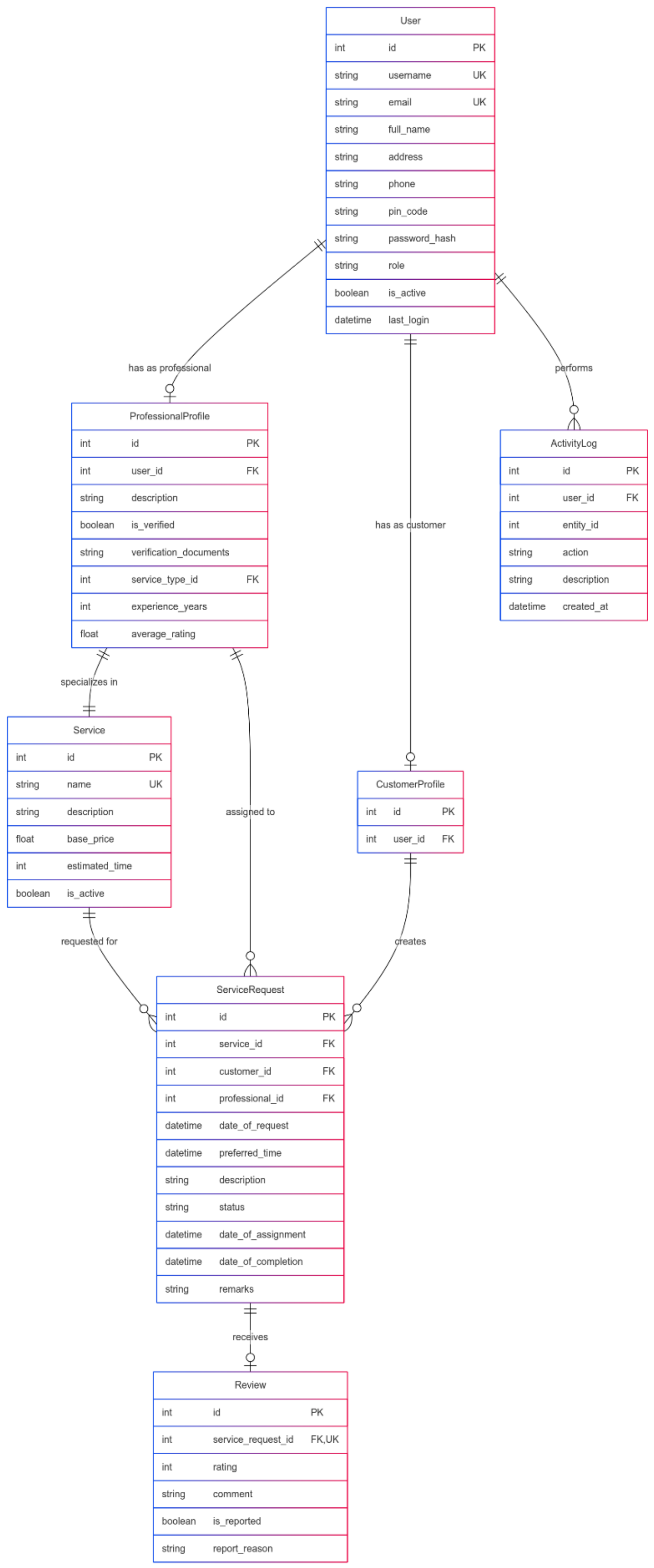
Description

The Household Services Application is a multi-user platform connecting customers with service professionals, with admin oversight. It manages the complete service lifecycle from booking to completion, includes professional verification, and provides comprehensive reporting features.

Technologies used

- **Flask (v3.1.0)**: Core web framework chosen for its lightweight nature and flexibility in API development
- **Flask-SQLAlchemy (v3.1.1)**: ORM for database abstraction, providing simplified database operations and model relationships
- **Flask-Marshmallow (v1.3.0)**: Schema-based serialization/deserialization for API request validation and response formatting
- **Flask-Caching (v2.3.1)**: For response caching to improve API performance on frequently accessed endpoints
- **Flask-CORS (v5.0.1)**: Handling Cross-Origin Resource Sharing for frontend integration
- **Flask-Mail (v0.10.0)**: Email notification service integration
- **PyJWT (v2.10.1)**: JSON Web Token implementation for secure authentication
- **Celery (v5.4.0)**: Asynchronous task queue for handling background jobs (email notifications, report generation)
- **Redis (v5.2.1)**: Used as message broker for Celery and for caching
- **SQLite**: Development database (configurable for production environments)
- **Python-dotenv (v1.0.1)**: Environment variable management for configuration

DB Schema Design



Design Rationale:

- **Single User table with role discrimination**: Simplifies authentication while allowing role-specific profiles through one-to-one relationships
- **Cascade deletes**: Ensures referential integrity when users are deleted
- **Soft deletion approach**: Using is_active flags instead of hard deletes for data audit purposes
- **Comprehensive logging**: ActivityLog provides complete audit trails for all significant actions
- **Strategic constraints**: CHECK constraints and unique indexes ensure data integrity
- **Status tracking**: Request status flow (created → assigned → completed) with timestamps for each stage
- **Review isolation**: Separate Review table with unique constraint to service request ensures one review per service
- **Professional verification**: Two-step process (document verification then account activation) for quality control

API Design

The API follows RESTful principles with resource-oriented endpoints organized by entity domains:

1. **Authentication API**: JWT-based authentication with token generation and validation
    - Implemented in routes/auth.py with middleware protection in utils/auth.py
2. **User Management APIs**: Customer and Professional registration, profile management
    - Customer APIs (routes/customer.py): Registration, listing, blocking/unblocking, dashboard
    - Professional APIs (routes/professional.py): Registration, verification, service type updates, dashboard
3. **Service Management APIs**: CRUD operations for services
    - Implemented in routes/service.py with admin-only access for critical operations
4. **Request Management APIs**: Service request lifecycle
    - Request creation, assignment, completion, and cancellation (routes/request.py)
    - Review submission and reporting system
5. **Admin Dashboard API**: Statistical data for platform management
    - Comprehensive metrics endpoint with optional filtering by time period, service type, location
6. **Export API**: Data export functionality
    - Celery-powered asynchronous CSV generation with status checking and download endpoints

All APIs use consistent request validation via Marshmallow schemas, standardized response formats through the APIResponse utility, and proper error handling. The implementation includes role-based access control via decorators and a caching layer for performance optimization.

Architecture and Features

The project follows a modular architecture with clear separation of concerns. Core components are organized as follows:

- **Routes**: Controllers are in the routes/ directory, separated by domain (auth, customer, professional, service, request)
- **Models**: Database models in models.py defining the schema and relationships
- **Schemas**: Request/response validation schemas in schemas/ directory, organized by entity
- **Utils**: Reusable utilities in utils/ for auth, caching, notifications, API response formatting
- **Templates**: Email templates in templates/emails/ for various notifications
- **Tasks**: Background jobs defined in tasks.py using Celery for async processing
- **Configuration**: App configuration in app.py with environment-based settings
- **Static**: File storage for uploads in static/uploads/ with security measures

Implemented Features:

1. **Core Features**:
    - **User Management**: Registration, authentication, profile management for all user types
    - **Service Management**: CRUD operations for service types with validation
    - **Request Lifecycle**: Complete flow from creation to completion with status tracking
    - **Professional Verification**: Document upload, admin review, and verification process
    - **Review System**: Customer reviews with reporting mechanism for professionals
2. **Enhanced Features**:
    - **Role-Based Dashboards**: Customized statistics for each user type with filtering options
    - **Caching Layer**: Redis-based caching with fallback mechanism when Redis is unavailable
    - **Asynchronous Processing**: Background tasks for email notifications and report generation
    - **Comprehensive Logging**: Activity tracking for audit and analytics purposes
    - **Geographic Filtering**: Service and professional filtering by location (pin code)

    - **Performance Optimization**: Database indexing and query optimization for high traffic endpoints

3. **Additional Features**:

    - **Data Export**: CSV generation for completed service requests with email notification
    - **Monthly Reports**: Automated reports to users with their activity statistics
    - **Daily Reminders**: Automated notifications for professionals with pending requests
    - **Smart Service Scheduling**: Time validation ensuring services can be completed within business hours
    - **Rating Analytics**: Trend analysis for professional ratings and customer satisfaction

Video

https://drive.google.com/file/d/1Pv5d0VAH6FbrjQVCcmnk7ojckCLxQBwd/view?usp=sharing