Project Report for Household Services Management System
Modern Application Development - I

Sanskar Pandey
23F3003478

Description

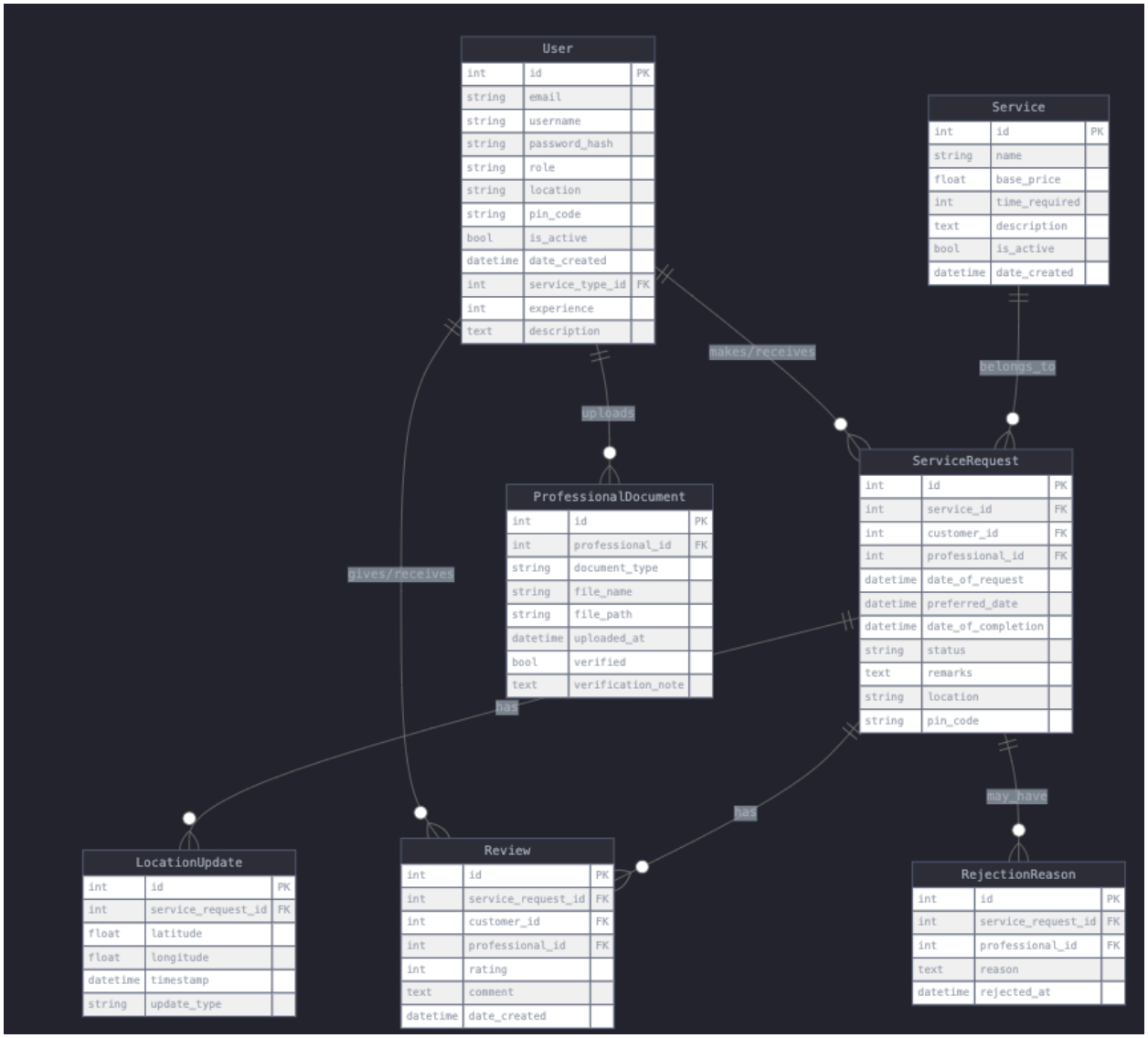
In this project, I used HTML, CSS, Bootstrap, Flask, RESTful APIs, SQLAlchemy, and other necessary modules to build a household services management application. I built a multi-role system (Admin/Professional/Customer) with login/signup functionality. Users can request services, professionals can accept/reject requests, and admins can manage the overall system. The application includes document verification for professionals, service request management, and a review system. The system tracks service status through various stages: requested, assigned, completed, and cancelled.

Technologies used

1. Flask: Core web application framework
2. Flask-SQLAlchemy: Database ORM and management
3. Flask-RESTx: API development and documentation
4. Flask-Login: User session management and authentication
5. Flask-Migrate: Database migrations
6. Flask-WTF: Form handling and CSRF protection
7. JWT: Token-based authentication
8. Bootstrap: Frontend styling
9. Datetime: Timestamp management
10. SQLite: Database options

DB Schema Design Relations:

- User Model: id, email, username, role, password_hash, location, pin_code, is_active
- Service Model: id, name, base_price, time_required, description, is_active
- ServiceRequest Model: id, service_id, customer_id, professional_id, status, preferred_date
- Review Model: id, service_request_id, rating, comment, customer_id, professional_id
- ProfessionalDocument Model: id, professional_id, document_type, file_path, verified
- LocationUpdate Model: id, service_request_id, latitude, longitude, timestamp



Architecture and Features Project Structure:

The project code is organized based on its utility in different files. I named my project **code**. Inside the app folder, we have several key components:

1. **init.py**: Contains the Flask application factory and configuration.
2. **api folder**: Houses all API-related files
    - auth_routes.py: Authentication endpoints
    - service_routes.py: Service management endpoints
    - models.py: API schemas and models
3. **models folder**: Database models

    - user.py: User model with roles (Admin/Professional/Customer)
    - service.py: Service and request models
    - documents.py: Professional document management
    - security.py: Security and audit models
4. **views folder**: Route handlers for different modules
    - admin.py: Admin dashboard and management
    - customer.py: Customer service requests
    - professional.py: Professional service management
    - profile.py: User profile management
    - auth.py: Authentication views
5. **static folder**: Contains static assets
    - css/style.css: Custom styling
    - images: Background images and icons
6. **templates folder**: HTML templates organized by module
    - admin/: Admin dashboard templates
    - customer/: Customer interface templates
    - professional/: Professional interface templates
    - auth/: Login and registration pages
    - base.html: Base template with common layout
7. **utils folder**: Helper functions and decorators
    - decorators.py: Role-based access control
    - helpers.py: Common utility functions
    - upload.py: File upload handlers
    - validators.py: Input validation
8. SQLite database is stored in the instance folder for development.

The app.py file is situated outside the app folder and serves as the application entry point. The HTML pages include landing page, signup/login pages, role-specific dashboards, service request forms, and review management interfaces. All templates are rendered through corresponding API endpoints ensuring proper separation of concerns between frontend and backend.

All interactions with the database are handled through SQLAlchemy models, and API endpoints provide a RESTful interface for data operations.

The controllers used are:

● Admin Routes:

- @bp.route('/admin/dashboard')
- @bp.route('/admin/services')
- @bp.route('/admin/users')

● Customer Routes:

- @bp.route('/customer/dashboard')
- @bp.route('/customer/services/search')
- @bp.route('/customer/service-request/create/int:service_id')

● Professional Routes:

- @bp.route('/professional/dashboard')
- @bp.route('/professional/requests')
- @bp.route('/professional/reviews')

● Auth Routes:

- @bp.route('/login')
- @bp.route('/register/<role>')
- @bp.route('/logout')

API Design

Auth API:

- POST /api/v1/auth/login: User authentication
- POST /api/v1/auth/register: User registration

● Service API:

- GET /api/v1/services: List all services
- GET /api/v1/services/<id>: Get service details
- POST /api/v1/services: Create new service (admin)

● Request API:

- GET /api/v1/requests: List service requests
- POST /api/v1/requests: Create service request
- PUT /api/v1/requests/<id>: Update request status

● User API:

- GET /api/v1/users: Get user profiles
- PUT /api/v1/users/<id>: Update user profile

Key Features:

1. Multi-role user system
2. Service request lifecycle management
3. Professional verification workflow
4. Review and rating system
5. Location-based service matching
6. Document upload and verification
7. Analytics and reporting

This project demonstrates a complete service management system with secure authentication, role-based access control, and professional verification workflows, providing a platform for connecting service providers with customers.

Video Link:
https://drive.google.com/drive/folders/1KsYNP85AqXUug4wqzusm1bT0Kkey8UQa