Modern Application Development II

Project Statement

Household Services Application - V2

It is a multi-user app (requires one admin and other service professionals/ customers) which acts as platform for providing comprehensive home servicing and solutions.

Frameworks to be used

These are the mandatory frameworks on which the project has to be built.

- SQLite for data storage
- Flask for API
- VueJS for UI
- VueJS Advanced with CLI (only if required, not necessary)
- Jinja2 templates if needed (Not to be used for UI)
- Bootstrap for HTML generation and styling (No other CSS framework is allowed)
- SQLite for database (No other database is permitted)
- Redis for caching
- Redis and Celery for batch jobs

Note: All demos should be possible on your local machine.

Roles

The platform will have **three** roles;

1. **Admin - root access** - it is a superuser of the app and requires no registration.

- Admin login redirects to the admin dashboard
- Admin will monitor all the users (customers/service professionals)
- Admin will create a new service with a base price
- Admin will approve a service professional after verification of profile docs
- Admin will block customer/service professionals based on fraudulent activity/poor reviews

- Other operations*

2. Service Professional - An individual that provides the service

- Login/Register
- Service professionals will accept/reject a request
- One professional is good at one of the services only
- He/she can accept/reject an assigned service request
- Professional profiles are visible based on customer reviews
- The professional will exit the location after the service is closed by the customer

Each professional may have;

- ID
- Name
- Date created
- Description
- service_type
- Experience etc.

3. Customer - an individual who has to book a service request

- Login/Register
- View/Search the service by the name/location pin code
- Open/close a service request
- He/she can post reviews/remarks on the closed service
- Others

Terminologies

1. **Service** - It refers to the type of service that the customer is looking for e.g AC servicing, plumbing etc.

Each service may have;

- ID
- Name
- Price
- Time required
- Description etc.

2. **Service Request**: A customer creates a service request providing the type of service the customers is looking for, when is it required etc.
   A service request may contain the following attributes:

- id - primary key
- service_id(foreign key-services table)
- customer_id(foreign key-customer table)
- professional_id(foreign key-professional table)
- date_of_request
- date_of_completion
- service_status(requested/assigned/closed)
- remarks (if any) etc.

Note: the above fields are not exhaustive, students can add more fields a/c to their requirements

Application Wireframe

A-Z Household Services

Note: The wireframe is provided only to get the flow of the application and what should appear when a specific user navigates from one page to another. It is NOT mandatory to exactly replicate the views given in the wireframe. Students can work on their front-end ideas.

Core Functionalities

1. Admin login, professional and user login (RBAC)

- A login/register form with fields like username, password etc. for professional, user and admin login
- The application should have only one admin identified by his role.
- You can either use Flask security (session or token) or JWT based Token based authentication to implement role-based access control
- The app must have a suitable model to store and differentiate all the types of users of the app.

2. Admin Dashboard - for the Admin

- Admin login redirects to admin dashboard
- Admin will manage all the users (customers/service professional)
- Admin will approve a service professional after verification of profile docs
- Admin will block customer/service professional based on fraudulent activity/poor reviews

3. Service Management - for the Admin
    - Create a new service with a base price.
    - Update an existing service - e.g. name, price, time_required and/or other fields
    - Delete an existing service
4. Service Request - for the customers
    - Create a new service request based on the services available
    - Edit an existing service request - e.g. date_of_request, completion status, remarks etc
    - Close an existing service request.
5. Search for available services
    - The customers should be able to search for available services based on their location, name, pin code etc.
    - The admin should be able to search for a professional to block/unblock/review them.
6. Take action on a particular service request - for the service professional
    Ability to view all the service requests from all the customers
    Ability to accept/reject a particular service request
    Ability to close the service request once completed*

7. Backend Jobs

a. Scheduled Job - Daily reminders - The application should send daily reminders to service professionals on g-chat using Google Chat Webhooks or SMS or mail

- Check if a professional has not visited/has pending service request
- If yes, then send the alert asking them to visit/accept/reject the service request
- The reminder can be sent in the evening, every day (students can choose the time)

b. Scheduled Job - Monthly Activity Report - Devise a monthly report for the customer created using HTML and sent via mail.

- The activity report can include service details, how many services were requested/closed etc.
- For the monthly report to be sent, start a job on the first day of every month → create a report using the above parameters → send it as an email

c. User Triggered Async Job - Export as CSV - Devise a CSV format details for the service requests closed by the professional

- This export is meant to download the service details (service_id, customer_id, professional_id, date_of_request, remarks etc.)
- Have a dashboard from where the admin can trigger the export
- This should trigger a batch job, and send an alert once done

8. Performance and Caching
   - Add caching where required to increase the performance
   - Add cache expiry
   - API Performance

Recommended Functionalities

- Well-designed PDF reports for Monthly activity reports (Students can choose between HTML and PDF reports)
- External APIs/libraries for creating charts, e.g. ChartJS
- Single Responsive UI for both Mobile and Desktop
- Unified UI that works across devices
- Add to desktop feature
- Implementing frontend validation on all the form fields using HTML5 form validation or JavaScript
- Implementing backend validation within your APIs

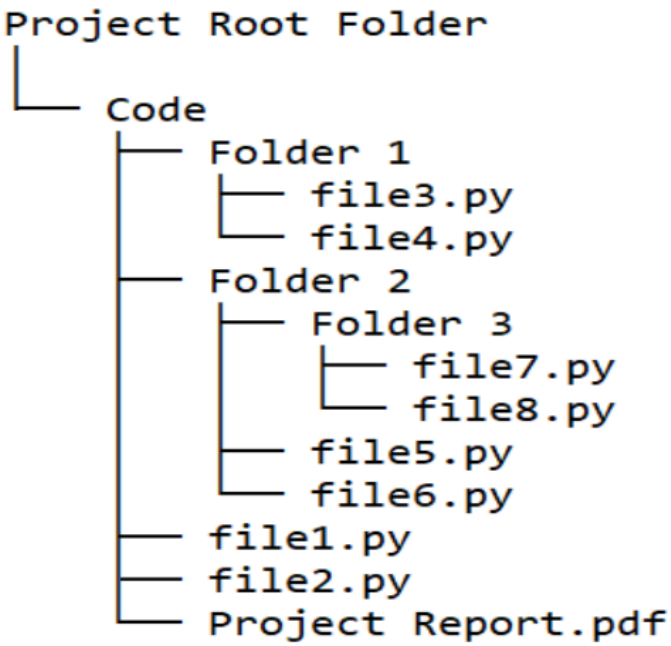
Optional Functionalities

- Provide styling and aesthetics to your application by creating a beautiful and responsive front end using simple CSS or Bootstrap
- Incorporate a proper login system to prevent unauthorized access to the app using Flask extensions like flask_login, flask_security etc.
- Implement a dummy payment portal (just a view taking payment details from customers for a service request
- Any additional feature you feel is appropriate for the application

Evaluation

- Students have to create and submit a project report (not more than 2 pages) on the portal, along with the actual project submission
- The report must include the following things;
- Student details
- Project details, including the question statement and how you approached the problem statement
- Frameworks and libraries used
- ER diagram of your database, including all the tables and their relations
- API resource endpoints (if any)
- Drive link of the presentation video

- The project report must be included as a PDF **inside** the root submission folder and NOT along with it.



- All code is to be submitted on the portal in a single zip file (zipping instructions are given in project document - **Project Doc T32024**
- Students have to create a brief (5-10 minute) video explaining how you approached the problem, what you have implemented, and any extra features
- The video must be uploaded on the student drive with **access to anyone with the link** and the link must be included in the report
- This will be viewed during or before the viva, so it should be a clear explanation of your work
- Viva: after the video explanation, you are required to give a demo of your work, and answer any questions that the examiner asks
- This includes making changes as requested and running the code for a live demo
- Other questions that may be unrelated to the project itself but are relevant to the course

Instructions

- This is a live document and will be updated with more details (wireframe)
- We will freeze the problem statement on or before **19/09/2024**, beyond which any modifications to the statement will be communicated via proper announcements.
- The project has to be submitted as a single zip file.