Author

Name : Midhil Katta
Roll number : 21f1006129
Student email : 21f1006129@ds.study.iitm.ac.in
Brief Introduction: I am a full time student at IITM BS Degree program, I come from a statistics background so I'm hoping Data Science is my next step to Upskill myself.

Description

The Household Service web application is a platform where users can essentially book a household service, but to enable that we have an admin who creates services, approves service professionals, who will accept requests and fulfil them, on top level we can split it into Admin functionalities, Service professional functionalities and User functionalities.

Technologies used

Flask Family: Flask, flask_security, flask_cors, flask_sqlalchemy, flask_caching, Blueprint, request
Celery Family: celery, Task, shared_task, schedules, schedules.crontab, AsyncResult
Werkzeug.security, datetime, csv.DictWriter, io.StringIO, io.BytesIO, jinja2, json, smtplib, os, email.mime.text, email.mime.multipart.

The Main crux was **flask** to create the application, **flask_security** was used to implement RBAC: role based authentication and login, **flask_sqlalchemy** was used to deply database models and query for data and update it, **flask_cors** is used to send requests to back end from front end, both of which are running on two different servers.
Celery, **Task**, **shared_task**, schedules, **schedules.crontab**, **AsyncResult** was used for backend schedules jobs like daily reminder and monthly report and to trigger an export.
Werkzeug.security is used to hash passwords, datetime was used to create and manipulate datetime objects.
csv.DictWriter, **io.StringIO**, **io.BytesIO**, **jinja2**, **json**,were used to create the html, csv used in backend jobs. **email.mime.text**, **email.mime.multipart** are used to send emails.
Smtplib is used to access a dummy **Mailhog** to simulate the backend jobs.
os is used to locate/create a folder where we store service professionals verification pdfs.

DB Schema Design

Database Scheme:
Tablename: 'user'
Columns: id (Integer, primary key),name(String), username(String), password(String),fs_uniquifier(String), active(Boolean)

Tablename: ‘role’
Columns: id(Integer, primary key), name(String), description(String)
Relationships: roles() with role table

Tablename: ‘role_users’
Columns: id(Integer, primary key) , user_id(Integer, Foreign key), role_id(Integer, Foreign key)

Tablename: ‘customer’
Columns: id(Integer,primary key), name(String), username(String), active(Boolean)

Tablename: ‘service_professional’
Columns: id(Integer,primary key), name(String), username(String), pincode(Integer), date_created(date), service_type(String), experience(Integer), requests_completed(Integer), cumulative_rating(float), active(Boolean)

Tablename: ‘service’
Columns: id(Integer,primary key),name(String), price(Integer), description(String)

Tablename: ‘service_request’
Columns: id(Integer, primary key), service_id(Integer), service_name(String), user_id(Integer), professional_id(Integer), total_amount(Integer), address(String), pincode(Integer), date_of_request(date), date_of_completion(date), service_status(String), rejected_by(list,json obj), rating(Integer), feedback(String)

To Implement RBAC(Role based authentication) we primarily store all user details in our user table which is linked to ‘role’ table and ‘role_users’ table, and ‘customer’ and ‘service_professional’ tables are used to store customers and service professionals details respectively, ‘service’ table is used to store services, and ‘service_requests’ is used to store all the service requests created. We make sql queries to get, create, update and delete data from the database.

API Design

The Entire backend in this project works like an API, all the routes defined perform CRUD operations on a specific data, be it user object, customer object, service professional object, service object, service request object. We use backend routes to Get, Update, Create, Delete an Object. We also implement Async CSV export for service requests via the api routes.

Architecture and Features

The project is broken into frontend and backend, in the frontend we have all the Vue js files which are UI pages a user interacts with, primarily we have Admin Views, Service Professionals Views and User views and respectively components for all these views . We have a frontend router and vuex store javascript files. Apart from this we have a Login view, Sign up view, and Service professional view.

In the backend all the routes act as Apis used to retrieve, update, create and delete data, the main files include app.py, routes.py, models.py, worker.py, cache.py we also have template folder which has html files used to produce daily reminder, monthly report files for backend jobs, and uploads folder which stores the verification pdf a service professional uploads while signing up and an instance folder which has the sqlite database file.

The Core features are RBAC login with token based authentication, all the users can login through the same page, this is made possible by flask_security, Rolemixing and UserMixing, login_user, app.security.datastore once a user logins we generate a token and store it in the frontend vuex store and using this token we authenticate the user whenever need to make fetch api calls, we retrieve data about the role of this user and accordingly route the user to admin or user or service professional dashboards.

Admin can create a service, edit a service and delete a service, this is done in the front end and the data is sent via fetch request to the backend to store in the database or update the existing data or deleting accordingly. Admin can Deactivate or Activate customer accounts or service professionals accounts this is done by sending a patch request to the backend to toggle user.active value and Admin can export a csv file containing details about service requests, this is done as a Async backend job.

Customers once logged in can book a service, search for a service and once they book a service they can cancel the service and if a service request is rejected by all service professionals the customer is intimated and the customer can Book again which will cancel the service request and redirect to book a service, or they can just cancel a service request, once a request is finished the customer has the option to close it upon which we take rating and feedback for the service request.

Service professional can view their service requests and for all the new requests made by customers, the professional can choose to either accept or reject a request and if rejected the request is assigned to another service professional and if they choose to accept it, they have the option to close it once they have finished the service upon which the customer is intimated to rate this closed service and provide feedback.

The Frontend router helps to route all the different views a user has access to, and in the backend all the Api calls made for CRUD operations ask for authentication token and each Api will have a role required to perform that Api call, since we store the user authentication token in our vuex store in the front end we send it during our fetch request and this token grants the permission to run the Api call and retrieve data.

All the Post operations are mainly done via frontend forms and validation is done at the form stage, and then once they submit, we validate the date before making a fetch call and also in the backend before performing our action.

Celery is used to perform two scheduled tasks one : sending daily reminder to the service professionals who have pending requests asking them to either accept or reject a request, and two: A monthly report for the customer activity with details of all the service request they made last month, which are pending and completed via email.

Video

https://drive.google.com/file/d/13oCItPRHylj9oBa2uInS6RG2gJQxm0Rp/view?usp=sharing