Video

https://drive.google.com/file/d/1CWtCSBZ8FRAYKItWZlMmQ_uZCmK0Vlf4/view?usp=sharing

Author

Sayantan Das
Roll Number: 21F1002905
21f1002905@student.onlinedegree.iitm.ac.in
I am a Finance-enthusiast who, along with this course, is pursuing a Bachelor's in Management Studies (specialization in Finance) from St. Xavier's College, Kolkata. I plan on to leverage the skills gained from this course in developing and solving financial problem statements.

Description

This project requires the development of a digital Kanban Board that can be accessed as a Web Application. It serves a similar purpose as a regular Kanban Board but also brings in additional features like periodic reminders and automated monthly reports.

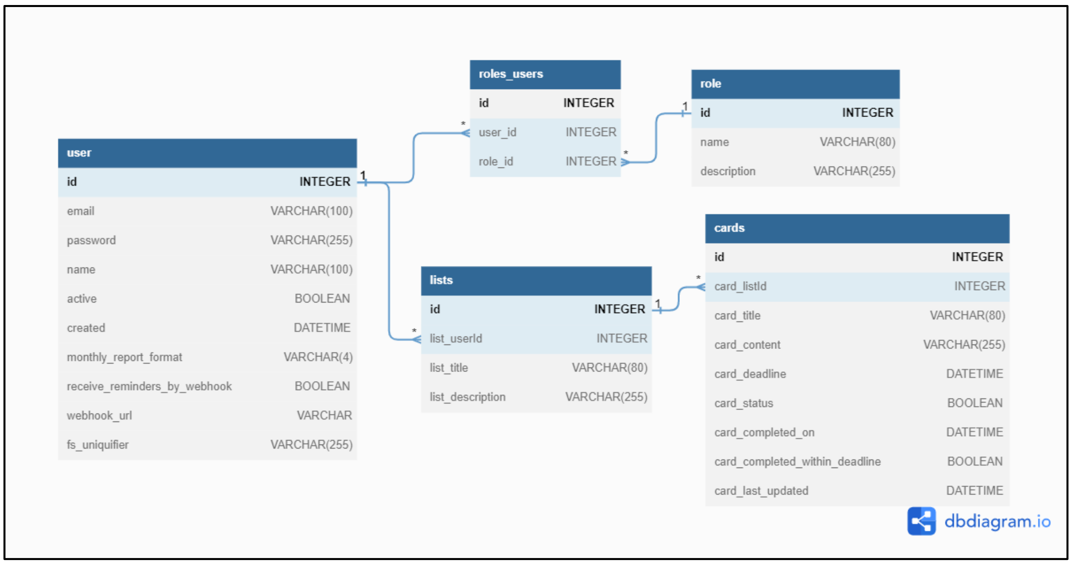
Technologies used

- **Front-end:** VueJS, Bootstrap, HTML & CSS
- **Backend:** Coded using Python with several packages as given below
- **Database:** SQLite 3
- **Technologies to be separately installed:** Redis (for celery job queuing & backend caching), Mail-Hog (SMTP Server Simulator for testing e-mail features)

Python Packages	Reason for use
Celery[redis]	for using Redis as a task queue database for Celery
Flask	For building the backend
Flask-Caching	Caching in the backend
Flask-Cors	To resolve CORS issues
Flask-RESTful	To develop the backend APIs
Flask-Security	For Token-based Authentication
Flask-SQLAlchemy	For establishing connection with the database & creating models
Httplib2	In sending reminder over Webhook
Jinja2	For creating E-mail body content & PDF content of the monthly reports
Matplotlib	For generating trendlines
Pandas	For reading imported CSV files
Weasyprint	For generating PDF monthly reports

DB Schema Design

The primary tables in the database are the users, lists & cards table that store the respective values. They have been associated with each other using foreign keys.



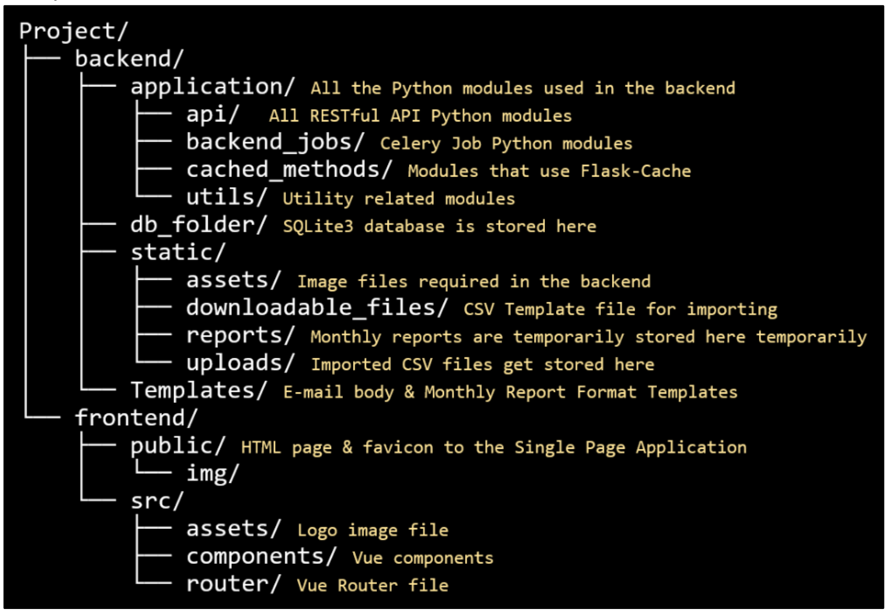
API Design

All the backend operations are handled via custom APIs except the token-based security authentication. The custom APIs are designed with the routes containing the word **/api/** as a part of the whole link. Example: *{{ base_url }}/api/user* or *{{ base_url }}/api/summary*

The *API.yaml* file in the backend folder contains details of all the implemented APIs.

Architecture and Features

The Project is organized in 2 parts – backend & frontend. The tree-diagram of the folder structure is as follows (along with the purpose pertaining to which modules have been placed there):



```
Project/
├── backend/
│   ├── application/ All the Python modules used in the backend
│   │   ├── api/   All RESTful API Python modules
│   │   ├── backend_jobs/ Celery Job Python modules
│   │   ├── cached_methods/ Modules that use Flask-Cache
│   │   └── utils/ Utility related modules
│   ├── db_folder/ SQLite3 database is stored here
│   ├── static/
│   │   ├── assets/ Image files required in the backend
│   │   ├── downloadable_files/ CSV Template file for importing
│   │   ├── reports/ Monthly reports are temporarily stored here temporarily
│   │   └── uploads/ Imported CSV files get stored here
│   └── Templates/ E-mail body & Monthly Report Format Templates
└── frontend/
    ├── public/ HTML page & favicon to the Single Page Application
    │   └── img/
    └── src/
        ├── assets/ Logo image file
        ├── components/ Vue components
        └── router/ Vue Router file
```

There is a README.md in the Project folder that describes how to execute both the parts.

All the core features from the Problem Statement have been incorporated. This includes the ability to register as Users, maintain multiple lists & cards, move, edit & delete the same, receive daily reminders & monthly reports, check summary & trendlines, export & import the data and caching for faster data access.