Name: Shreya Y
Roll Number: 21f1002768
Student email: 21f1002768@student.onlinedegree.iitm.ac.in
About me: I am a Chartered Accountant pursuing the BS in Data Science and Programming degree full time and am currently in the final term of the diploma level. I have built upon the Kanban application by introducing reactivity and additional features taught to us in the Modern Application development-II course, for a better user experience.

Description

The problem statement requires us to build a KanBoard, a typical task management application which allows users with a login to have a dashboard, where they can make and manage to-do lists, along with provision to add sub-tasks under each list. The user should have access to task summaries and a provision to export them in a csv format. The application should send daily reminders of pending tasks to users, and monthly reports summarising their tasks.

Technologies used

The technologies used in building the application are as follows:
VueJS, HTML, CSS, Bootstrap - Frontend: The application is reactive and responsive through the use of the VueJS library(CDN), facilitating easy navigation and implementing front-end validations. Requests for data are made through fetch calls to the API with appropriate headers and information sent.
In combination with this, HTML, Bootstrap and CSS were primarily used to render columns and cards containing task information, to implement functionalities such as buttons and dropdowns and for styling.
Flask-Restful - Middleware: Flask-restful was used to build APIs necessary to return data to the front-end functions fetching data to be rendered to the user. Data is passed in the form of Json/Blobs to the client.
Celery and Redis - Middleware: Celery in combination with Redis has been used to perform asynchronous jobs on the server side, such as processing multiple csv requests and scheduled email reminders. Redis has also been used for caching data for better performance.
Flask-SQLAlchemy: Flask SQLAlchemy was primarily used for building models which connect the database to the APIs for all CRUD operations. Information from arguments is passed on to the back-end through sessions.
SQLite - Backend: The storage of data is on a database created in SQLite3. Database constraints and foreign key relationships have been implemented to link users, lists and cards.
Flask-security - Authentication: Token authentication feature of Flask-security layered with other username and password authentication, has been used to ensure that the access to every API is authenticated.

DB Schema Design

The database comprises of 3 main tables as follows:
User: Information about registered users. The columns are:

- id *(integer)*: Unique id assigned to each user - the *primary key*.
- Username *(text)*: Username chosen user.
- email *(text)*: User email id for login.
- password *(text)*: Stores user password using ‘bcrypt’ and ‘salt’ techniques.
- report_template*(text)*: Stores user choice of monthly report template - HTML/PDF.
- active *(integer)* : Column specified by Flask-security “User” model.

List: Details of all lists across users. Columns are:

- list_id *(integer)*: Unique list ID - *primary key*.
- list_name *(text)*: Unique name for every user list.
- id *(integer)*: Foreign key linking the list table to the user table.
- description *(text)*: List description.

Cards: Details of all cards across users. Columns are

- card_title (text): Unique title for each card in a particular list.
- list_id (integer): Foreign key mapping the card to the List table.
  (card_title, list_id) form the primary key.
- card_content (text): Brief description.
- created, card_due_date, status, completed_date, last_modified (text): Date of creation, due date, complete/incomplete status, date of completion and last modified date of the card.

API design

The APIs fall into 6 broad categories - UserAPI, ListAPI, CardAPI corresponding to the models and used to implement CRUD operations using the relevant HTTP verb methods. ExportAPI and Import API are used for csv exports/imports. ListSummaryAPI, CardSummaryAPI, CompletionSummaryAPI and LastUpdateAPI return summaries. The information as requested by views through get requests is sent.

Architecture and Features

Templates: Comprises all the html home page rendered to the user.
Static: Comprises the VueJS application file and the relevant components. Contains CSS styling and relevant images.
Application: The database.py file imports the database and passes it on to models.py which creates the SQLAlchemy ORM objects for database tables. The objects from models.py are imported into api.py, config.py specifies the database path and sets the various configurations, including for Flask-security, Celery and SMTP server. cache.py contains asynchronous tasks which are called by api.py as needed.
main.py and __init__.py : Creates the application and imports the settings from config.py. Cache is created and passed into the application context in __init__.py.
Report Templates: Templates - Contains html templates for user email body, monthly reports and sample template of the upload file. Generated - Stores the PDF and csv files created when a request is triggered.
Database: Comprises the SQLite database file.
Application documentation: Contains the yaml file for the API.
requirements.txt, local_setup.sh, local_workers.sh, local_beat.sh, local_run.sh: Used to install the required libraries and consolidate the multi-line shell script codes.

The key features of the application are a board with capabilities to create and edit lists in-place, create and edit cards in-place, drag and drop cards across lists, set task due dates, mark tasks complete, delete lists/cards. It provides facilities to export csv reports, view summaries and send scheduled reminders and reports. The application is protected by token authentication.
The additional features include the option to bulk upload data to populate a dashboard and the choice between HTML and PDF reports .

Video link

https://drive.google.com/file/d/1fB_rfgY6KnShoRoFvhxH-AXjqMx3Pe57/view?usp=sharing