Author

Pranav P Dave
21f1005187
21f1005187@student.onlinedegree.iitm.ac.in
I am currently doing my Btech in electronics and communication engineering at PES University(RR).

Description

In this project a user must be able to create trackers. These trackers belong to either a numeric type or Multiple-choice , the time series is modified to be numeric type, theymust be configurable and editable. The user can plot graphs to get a visual representation of the data. The user must be able to trigger exports of their csv to their emails and also receive reminders for logging if not logged yet. At the end of each month the user must receive a pdf report.

Technologies used

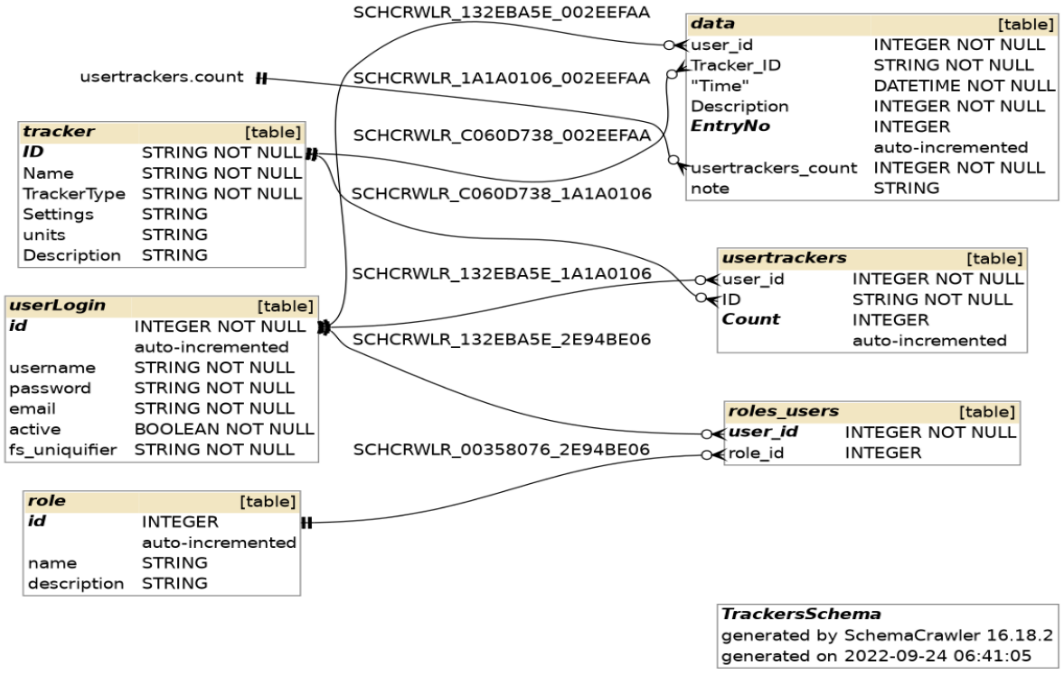
Backend

- **Flask restful** for the **Api**
- **datetime** to use **datetime** and **timedelta** for filtering based on date.
- **HTTP from werkzeug.exceptions and flask make_response** to define custom errors.
- **json** to use **json.dumps**
- **flask marshall_with** to output as json
- **flask fields** to set datatype of value of each json key
- **flask reqparse** to get data from the user.
- **flask_sqlalchemy** to import **SQLAlchemy** for the model
- **sqlalchemy func** to get max value of field
- **celery** to schedule and execute jobs like sending mails, generating pdfs and csv etc.
- **flask_security** to set up **token based authentication**
- **flask_caching** to set up endpoint caching
- **flask_cors** to allow cross origin requests
- **secrets** to generate a random hexadeciman for fsuniquefier
- **redis** used to create a database for **celery** and **caching**
- **mailhog** for testing mails
- **smtplib** and **email** to send mails
- **jinja2** to render templates
- **weasyprint** to render pdf
- **csv** to generate csvs
- **contrab** to schedule celery tasks

FRONTEND:

- **bootstrap-vue**
- **chart.js for charts**
- **CSS AND BOOTSTRAP**
- **VUE Router** for routing
- **Local storage to store auth token**

DB Schema Design



UserLogin:

Column Name	#	Data Type	Length	Not Null	Auto Increment	Default	Description
email	1	String		true	false		
id	2	STRING	[NULL]	true	false	[NULL]	INTEGER
password	3	STRING	[NULL]	true	false	[NULL]	[NULL]
active	4	BOOLEAN	[NULL]	true	false	[NULL]	True if user is logged in
fs_uniquefier	5	STRING	[NULL]	true	false		Used to generate token

Tracker:

Column Name	#	Data Type	Length	Not Null	Auto Increment	Default	Description
ID	1	STRING	[NULL]	true	false	[NULL]	PRIMARY KEY,Tracker ID
Name	2	STRING	[NULL]	true	false	[NULL]	Tracker Name
TrackerType	3	STRING	[NULL]	true	false	[NULL]	Numerical or MC
Settings	4	STRING	[NULL]	false	false	[NULL]	Feeling/moods etc
units	5	STRING	[NULL]	false	false	[NULL]	[NULL]
Description	6	STRING	[NULL]	false	false	[NULL]	Notes

User Trackers:

Column Name	#	Data Type	Length	Not Null	Auto Increment	Default	Description
User	1	INTEGER	[NULL]	true	false	[NULL]	FOREIGN KEY,Table UserLogin
ID	2	STRING	[NULL]	true	false	[NULL]	FOREIGN KEY,Table Trackers
Count	3	INTEGER	[NULL]	false	true	[NULL]	PRIMARY KEY

Data:

Column Name	#	Data Type	Length	Not Null	Auto Increment	Default	Description
user_id	1	INTEGER	[NULL]	true	false	[NULL]	Foreign key,User ID
Tracker_ID	2	STRING	[NULL]	true	false	[NULL]	Trackers ID
Time	3	DATETIME	[NULL]	true	false	[NULL]	[NULL]
Description	4	INTEGER	[NULL]	true	false	[NULL]	Value to be entered
EntryNo	5	INTEGER	[NULL]	false	true	[NULL]	Entry index, PRIMARY KEY
usertrackers_count	6	INTEGER	[NULL]	true	false	[NULL]	UserTrackers table ,FOREIGN KEY
note	7	STRING	[NULL]	false	false	[NULL]	Not for log

Default implementation of Roles and User Roles as seen in schema for implementation of flask security.

The DB was designed this way to **maintain** one to one and many to one relationships and no many to many relationships. This helps **CASCADING DELETES**. Hence a relationship table User Trackers. This helps in **keeping track** of which **user** has which **Tracker**.

API Design

I have made APIs for basic **CRUD** i.e (get,post,put,delete) for each table.(LoginAPI, TrackerAPI, userTrackerAPI, DataAPI)
Also I have made APIs (**LogAPI**) to get data from a **certain date** and another (**DailyLogAPI**) to collapse all data points from **different times on the same day into one point**.
The API parses the input json and uses it to perform operations as needed(Mostly CRUD in this case). Then the sqlite data object is converted into json using marshal_with. The jsons have a predefined structure which is set with the help of fields.
All endpoints except get and post from LOGIN API require an authentication token to access. Commonly used endpoints like
The UsersTrackers API and Log and Daily log api are cached for 10s for demo purposes but ideally would be longer.
The cache is on a redis server running on port 6379.

Architecture and Features

The controller is the front end which written in vue.js which sends requests to the server. The main file is App.vue which is the front end. All views are present in the **views** folder, Components, mainly **linechart.vue** is present in the components folder. The router **index.js** is in the router folder.

Api in file **API.py** in backend folder performs the required operations on the model named **models.py**. File **main.py** initializes all requiered flask components and celery. It also provides the endpoint to start async task to generate a csv. The database file is present in the same directory as the python files. File **tasks.py** is used to schedule and execute celery tasks like sending mails generating reports etc. It has a templates folder for pdf reports

Default features:

The web apps styling is implemented using css bootstrap and html docs use jinja.
Note: All CRUD done using API.

- The frontend files use bootstrap for styling.
- The login system generates a token since token based authentication using flask security has been implemented. The user can edit his password or delete the user.
- The dashboard of a user has a list of trackers currently created by the user with links on the list to edit or getmore info. The top right corner of the page has buttons to create new trackers, edit password and logout.
- When clicking more, a graph shows datapoints if any are any datapoints present. Also below it are all the logs in order of their creation. These logs can be updated or deleted.
- Graphs are plotted as line plots using v u e chart.js. The datapoints are based on dates. The values of all get added(numeric) or averaged(multiple-choice) for the same day.
- Reminders, pdfs and csv are sent by mail as required and scheduled.
- Cache has been implemented using flask caching.

Video

https://drive.google.com/file/d/1tlls80hhnkaxmlc7htNEuMZKeV7o64AX/view?usp=sharing