Documentation of Assignment 2

Author: *Ali* R. Nik (arnik@tutanota.com)

main.py:

- **Libraries**
  - google.oauth2.id_token
    - Parse and verify an ID Token issued by Google OAuth 2.0 authorization server
  - google.cloud
    - Access datastore module from App engine
  - flask
    - Access Flask Framework which is a simple handler of HTTP methods
  - google.auth.transport
    - This library simplifies using Google various server-to-server authentication mechanisms to access Google APIs
- **Models and data structures**
  - User:
    - Description: User Model is used to save information about users fetched from firebase. Also it save the primary keys of calendars that is shared with it. It consists of the following columns:
      - name/ID: the name of user as primary key to access user.
      - name: the name of user.
      - shared_me: a dictionary which the key is equal to the name of calendar and value with “No” or “Yes”. First the calendar is shared with the user and {“CALNAME”: “No”}. After some time when user accept sharing calendar, the value change to Yes, ie {“CALNAME”: “Yes”}.
    - Reason: Having user information in firebase is not enough as I need to modify the the shared_me data-structure . So I create this model to easily update information especially

I need it as shared_me field is always changing and its not a good idea to use firebase as saving “always modifying” fields.

- cal:
  - Description: This Model represents Calendar information with the following fields:
    - name/ID: the name of calendar + the name of creator of calendar.
    - Name: the name of calendar
    - owner: the name user which is the creator of calendar
    - shared: A map of {“USERNAME”: “USERNAME”} which save users that the calendar is shared with them.
  - Reason: Its obvious that we need calendar information to saved in database. The reason I added “shared” columns is that to have a list of usernames that the calendar is shared with them . In fact its not a list and instead its a dictionary to decrease the access time when implementing the code.
- event:
  - Description: This Model represents the events that is created for each calendar and it has the following fields:
    - name/ID: the name of event + the name of calendar + the name of creator of event + the name of user of this event.
    - name: the event name
    - creator: the creator of event
    - start_date: the date and time of start of event saved as timestamp.
    - end_date: the date and time of end of event saved as timestamp.
    - cal: the name of calendar that the event is belongs to it.
    - user: the user that the event is shared with it. Remember that this only used when the event is directly shared with user not by calendar sharing.
  - Reason: The reason I designed event model in this way is each event is defined for a calendar and also events could also shared between users . So each event is hardly linked within its user and calendar.

• Functions

Before I start documenting functions, I should mention there are many functions from previous assignment that I also use them in this assignment. So I do not re-document them in this document.

- create_row_from_data((kind, name, data):
  - Description: In previous assignment there is a function called create_row and the difference of this function with that is this function also gets an object called “data” and its type is an Entity object (dict) and create row with that data instead of getting data from form.
- create_cal_for_user(user, calname):
  - Description: check for the existence of calendar the user named and if that calendar does not exist, it create a calendar for the specified user.
- add_user_if_not_added(claims):
  - Description: check for the existence of user and if the user does not exist in db , it create the user with the its initial calendar called personalCalendar.
- def get_week(day):
  - Description: return the week and all days in it for the day passed to it. It also return the dominant month which the week located in it. By dominant month, I mean the month that most of the days of that week is located in it. The tuple (weeks, dominant month) is returned which weeks is a list of dicts which each element represent a day.
- get_my_cals(claims):
  - Description: retrieve all calendars for the signed-in user. After that retrieve all calendars that are shared with this user and then checks for selected calendars by user and only return those which selected.
- add_selected_field(result):
  - Description: read the response of GET data and try to fetch the list of selected calendars which user want to see in calendar view. Also this function has the list of all calendars which user has. So it iterate through all of user calendars and mark them as selected if it is in the selected list and return both the updated list of calendars (which has been marked as selected) and selection list.
- get_shared_cals_list(claims):
  - Description: it only iterate through calendars that is shared with user(iterate shared_me column of User table) and return that list.
- get_and_set_cal_week():
  - Description: read the response of GET data and try to fetch information about the day that user selected and calls get_week(fetched day) function with the fetched information about day (read get_week() function description) . And finally return the data returned by get_week() functions and today and a random day of selected week.
- add_or_edit_calendar(claims) and add_or_edit_event(claims):

        - Description: fetch the data of GET and check the route is correct or not. Validate the data and decide to create a new calendar or update the calendar .
    - fill_mat(my_cals, week_info, selected_cals, claims):
        - Description: create a matrix with 48 rows and 7 columns of a week (48 because there are 24 hours and I decide to split it to 30 mins chuncks instead of 1 hours chuncks). Fill that matrix with necessary data for showing selected calendars in calendar view.
    - delete_cal(claims):
        - Description: read the data of GET response to see the route is correct and if the route is for deletion, it first remove all the occurrences of calendar in shared_me list of all users . Then delete all events in that calendar. Because events are hard linked with calendars. And finally delete the calendar from calendars table safely.
    - delete_event(claims):
        - Description: check the correctness of route by checking the response of GET, and then validate authority and remove the event from event table.
    - acc_or_dec_shared_cal(dec_or_acc, sharedby, calname):
        - Description: Change the status of shared_me field of user table based on decision of user to accept sharing of calendar or to reject sharing of calendar. Also it update the shared field of calendar depend on the decision of the user.
    - stop_sharing_event(eventname, calname, username):
        - Description: remove the direct sharing of event for specified user by modifying event table. It **does not affect on that event** if it is **shared by calendar**.