Cloud Computing

Assignment 2

Building an Application for Task Management amongst multiple Users

Motivation

This web application was developed for the requirement of Master's in Big Data Management and Analytics, Semester 2- Cloud Computing module. The aim of this assignment is to develop a task management system for multiple users as a PaaS application.

1. Description

'Tasked' is a PaaS (Platform as a Service) application that is developed using Google App Engine with Python version 2.7. The application serves as a task management system for multiple users. This application provides users with login-logout and task creation services. In the application users can add task boards, which are collection of tasks and users to the database and modify details of existing task boards. The users can also invite other users to see and interact with boards they have created. The functionality to delete the entire board is available only for the created users. Users can also edit and delete tasks and mark them as complete. This application is taking advantage of parent-child relationships, KeyProperties of Google App Engine. Retrieval using keys is much faster than access using queries. Hence this increases the response time while retrieving values from the database. Models are designed and structured to allow for direct key access.

2. File organization and naming conventions

All the image files are included in the 'img' folder and the CSS stylesheet is included in the 'css' folder. These folders with the other yaml, python and html files are zipped together for submission. All files are named according to their actual functionalities making them easier to understand. Similarly, data model names and variable names are also self- explanatory.

3. Images

All images included to enhance the aesthetics of the User Interface are designed using the online platform Canva. Images are thus not violating any copyright laws and are solely created for the purpose of this application.

4. Reference materials used for learning

In order to learn the basics of Google App Engine, the 'Google App Engine with Python' pdf provided as part of the module was utilised. To deeply understand, the concepts of entity, parent-child relationships, KeyProperties etc., Google App Engine documentation was referenced. All of the references are provided in the end of this document.

5. Code Documentation

5.1. Python classes and methods

1. app.yaml

This file contains all the configuration details including name and version information about the library used (jinja2) and the static directory for 'img' and 'css' folders. Handlers transfers requests for all the URLS by using the variable 'app' which is defined in python file (main.py) which has the routing table for this web application.

2. get() in all Classes

All classes contain the get method for handling HTTP get request and accepts self as argument. This means that this method may only be called on an instance of the class. It will render the result of a GET request to this class. [1] This functionality of get() method is common for all classes. Other functionalities of get() methods specific to classes are documented in appropriate sections.

3. Routing table in main.py

All webapp2 applications are instances of a WSGIApplication which has all of the required code to transfer requests to the appropriate classes. Application routing table is defined as a list in main.py. Tuples are used in the list that states that any requests to the root of the application at '/' calls the respective methods of an instance of the 'MainPage' class. [1]

4. MainPage class

4.1.1. get()

This method creates a URL string as 'login'. The login URL and URL string is rendered as hash map to the index.html which is the Login page. When a user is logged in, they are redirected to the dashboard. Google App Engine's default login- logout service is used for this assignment.

5. Logout class

5.1.1. get()

This method creates a logout URL with Google App Engine's default login-logout service. The user is redirected to the login page, when the user clicks the Logout button in the navbar from anywhere in the dashboard. The URL and URL string will be rendered to login page (index.html).

6. Dashboard class in dash.py

6.1. get()

This method is used to render data to the dashboard, when a user logs in. It checks for task boards created by the user and the task boards the user is invited to, if it exists, the task boards are appended to lists which will be further displayed on the dashboard. The tasks completed in the task boards that a user is a part of is also rendered to dashboard.html. Task board data is retrieved by exploiting parent-child relationship. Direct key access is used to retrieve this data.

7. ViewTask class in dash.py

7.1. get()

This method is used to render task boards to the 'View' page, when a user goes to the page from the dashboard. This method renders all the task boards to viewAllBoards.html page using jinja2.

8. AddBoard class in dash.py

8.1. get()

This method provides the necessary functionalities to allow users to create task boards. List of users in the system will be rendered, so that users can be invited to the task board. Inviting users while creating task board is not mandatory. Users can also be added in later stages.

8.2. post()

When the users select the 'Add Board' option, name, description and list of invited users which are checkboxes will be retrieved from the form and stored in the 'TaskBoard' model. Users will only be allowed to enter task board title as maximum of 100 characters. This is enforced to ensure integrity and is achieved using html form 'maxlength' option.

8.3. add_invited_users()

This method accepts an argument of self, user names and 'TaskBoard' object. This method is used to store the list of invited users in the model.

9. ViewBoard class in dash.py

9.1. get()

get() method in ViewBoard class is used to display the details in individual task boards. Title and due date of all tasks are appended to lists to be rendered to the table in ViewBoard.html page. This method also renders, the name of assigned users of each tasks and the completion status of each tasks. There is also an option to remove users from the task board, for this a select box is used in the html form. The get() method also renders the list of invited users for this functionality. Once a username is selected and confirmed, the data in the TaskBoard Model will be modified.

10. AddTask class in dash.py

10.1. get()

get() method in AddTask class renders the information about the task board to the addTask.html page by passing the task board id and list of users. While a task is created, there is an option to assign users which is implement using select box. For displaying the list of users to choose from, this list is also passed to addTask.html.

10.2. post()

When the action is "Add Task", post() method puts the task details such as name, due date in the datastore if the task name is unique. In the html form, the minimum value for due date is set to be '01-04-2020' for integrity of the application and validation purposes. If there are tasks with the same name in the dashboard, 'Task name already exists' error message will be displayed. This is achieved by running a loop through all tasks in this method. Users will only be allowed to enter task title as maximum of 100 characters. This is enforced to ensure integrity and is achieved using html form 'maxlength' option. If the user cancels the add task operation, the user will be redirected to the information page of the task board.

11. EditTask class in dash.py

11.1. get()

This method considers four situations; Assigning of Tasks, Edit, Delete, Completion of tasks. This is performed by assigning values to 'mode'. When task is set to complete, status is set to True. The timestamp of completion is persisted and stored in the database. This date and time is also rendered to information page of each task board. When the mode is 'DEL', task will be removed from the task board information page and the data will be deleted from the datastore. For 'ASSI' and 'EDIT' mode, action is set to the respective options. This action, task board lists and users are rendered to addTask.html page.

11.2. post()

When the action is "Assign Task", the user value is retrieved and stored in the datastore, and the status of the task will be marked as assigned. For 'Update Task' action, the updated task name is obtained and the datastore is checked if there are any other tasks in the task board with the same name. If it exists, then a message will be displayed asking user to change the task name. If the updated task name is unique, it will be stored in the datastore. When the action is "Delete User", the task boards will be checked to see if the user is a part of them, if yes, the user will be removed from the task boards. If there are tasks assigned to user, those tasks will be marked unassigned and highlighted. For this a loop is ran through the tasks. The details of those tasks will be modified accordingly. If action is "Invite User", the user object will be obtained from the select box and the list of user attributes in the TaskBoard will be appended with the new user object.

12. EditBoard class in dash.py

12.1. post()

The post() method in EditBoard class is used for operations on the task board. When the action is 'Update', updated name and description of the task board will be stored in the datastore. For 'Delete' action, the task board will be deleted after getting its key.

5.2. JavaScript functions

5.2.1. Functions in viewBoard.html

1. **deleteUser**

deleteUser() JavaScript function is used for both remove user and invite user operations. This method is written to ask appropriate confirmation to the user before an invite or remove user operation is carried out. Some hidden fields are also set here to pass the attribute values.

2. **complateUser**

This JavaScript function is used to ask confirmation to the user before a task is marked complete in the datastore.

3. **deleteTask**

This JavaScript function is used to ask confirmation to the user before a task is deleted from the datastore, when the user clicks the delete icon for each task.

4. **renameTaskBoard**

This function is used to update the name and description of the task board, when the user changes these values in the form and selects the ‘Update’ option. Hidden fields are also set to use the form values.

5. **checkDeleteUpdate**

checkDeleteUpdate() function in the script is used to ensure that all tasks and users are removed before the task board is deleted fully. It alerts the users about this condition when they attempt to delete the task board. This function also invokes the checkUserExist() to see if there are users present in the task board. The count of tasks is also checked for equality with zero.

6. **checkUserExist**

The count of invited users in task board is retrieved for checking if it is zero, i.e., if there are no users in the task board it returns Boolean False and Boolean True if there are users in the task board.

7. **totalTask**

totalTask() returns the number of tasks in the task board. This count is taken from the table in the User Interface itself.

8. **activeTask**

activeTask() returns the number of active tasks in the task board. This count is also obtained from the table in the User Interface itself.

5.2.2. Functions in addTask.html

1. validateData

This function is used to ensure that name and due_date of tasks are not null. If the user do not input these values, a alert message will pop up.

2. removeMessage

removeMessage() function is used to remove the validation message banner.

6. Data Structures and Models

6.1. Models

6.1.1. MyUser in myuser.py

The MyUser class extends the ndb Model.

a. 'emailaddress': Email address contains alphabets, numbers and characters like @. Thus, string is the best choice for it. It also allows for the usage of Google App Engine's default login-logout service**.**
b. 'created_task_board': This attribute stores the list of task boards created by the user with associated email address. It is set to be a KeyProperty with kind='TaskBoard' because it should allow direct key access for data retrieval without requiring query.
c. 'invited_task_board': : This attribute stores the list of task boards the user is invited to . It is set to be a KeyProperty with kind='TaskBoard' because it should allow direct key access for data retrieval without requiring query. Many-to-Many relationship is designed

6.1.2. TaskBoard in task.py

The TaskBoard class extends the ndb Model.

a. 'name': This attribute stores the name of the task board as StringProperty which can contain alphanumeric characters.
b. 'inv_users'=This attribute stores the list of invited users as StringProperty. List of strings is chosen because names can contain any characters.
c. 'tasks'= This attribute stores the list of tasks in that task board. It is set to be a KeyProperty with kind='Task' because it should allow direct key access for data retrieval without requiring query. One-to-Many Relationship is utilised here.
d. 'created_by' = This stores the name of the creator of the task board as a StringProperty.

6.1.3. Task in task.py

The Task class extends the ndb Model.

a. 'name': This attribute stores the title of the task as StringProperty which can contain alphanumeric characters.
a. 'due_date': This attribute stores the due date of the task as DateProperty.
b. 'assign': This attribute stores the name of the user the task is assigned to.

c. ‘status’: This attribute stores the completion status of the task and the default is set to False.
d. ‘completedDateTime’: This attribute stores the timestamp when a task is completed. In order to set date and time to current values, auto_now_add is set to True.
e. ‘assignRemoved’ : This attribute stores the assigned status of the task and the default is set to False. This is created for using it to display tasks in red, if the task assigned to the user is removed.

6.2. Data Structures

6.2.1. Dictionaries

Dictionaries offer powerful functionalities like hash maps. Throughout this project, hash map of all the values that is to be rendered in the template, are used. This is particularly useful with jinja2 to render content, message, entity objects, user, tasks and task board lists etc.

6.2.2. Lists

Lists are used in many classes to store the tasks and task board names, dates, and other details. Lists were chosen because it is easy to implement and it is easy to append values to list. Application routing table is also provided in the form of a python list.

6.2.3. Tuples

Tuples are given in the application routing table list to ensure that requests to the root of the application invoke the respective methods of an instance of the MainPage class. [1]

7. Design Decisions

The application is named ‘Tasked’, as it conveys the use cases of the application in a single word. The logo is designed using Canva, and the logo narrates the purpose, the application is developed for. A task management system should be user friendly and convenient, as it will be utilised by multiple users of an organization. Hence, the design of this web application is implemented by keeping these factors in mind. Bootstrap is used for improving the aesthetics of the application. The layout is designed with one side navigation bar and the top navigation bar, with the content below changing as different links are being selected by the user, this choice was made to develop a consistent web application. Consistent template is really important for systems like these, as it retains the focus of the users on the tasks rather than the distracting and varying templates. Background is kept to be plain white for the same reason. Bringing in too many colours or images in the background confuses the user.

In order to introduce fun and vibrancy to the application, the login page is developed with animated images designed using Canva. Convenience of the users was the main concern while developing the application. For easy access, logout link is provided in the top navigation bar separately from other links in the side bar. A dashboard with all the task boards the user is part of, is included so that users can see this list in one glance. The dashboard is an innovative approach which is solely meant to increase the ease of UI. The counters including statistics of tasks completed, tasks completed today by that user are also provided in separate bordered boxes, so that users can easily know about these values every day, without clicking through the task boards which can be time consuming. Apart from these, the ability for users to see the task statistics in each task board is included so that there is more transparency. The webpages are responsive, as users of an organization may access the application through multiple devices to login to the task management system and to engage with their colleagues and perform tasks. Another useful feature, is an icon to assign a task to a user later and not necessarily with task creation. This feature was provided along with the edit and delete buttons, because the creator of the task may not have chosen the user while creating the task.

Colour choices for the application are finalised by referencing *The Principles of Beautiful Web Design* by Jason Beaird. Dark green and red colours are avoided as much as possible because they are the most common forms of colour blindness. A task management system should be accessible to everyone, including people with colour blindness. Buttons are utilising multi-colour palette; this is for the users who may feel that white colour is boring. For some use cases, icons with tooltip text are utilised instead of text for the buttons, as visualizations are comprehensible enough and looks more appealing for frequently performed actions. Tables were selected to display the details of task boards, because it is an easy way to display properly formatted content and it looks neat and clean, when contrasted to the white background. A font stack is used in order to cater to a situation of unavailable fonts with different browsers. The fonts that are chosen are Tahoma and Verdana. This is to maintain the professional yet attractive look of the application.

References

1. Denby, B., 2017. Google App Engine By Example.
2. Google Cloud. 2020. Python On Google App Engine | App Engine Documentation | Google Cloud. [online] Available at: <https://cloud.google.com/appengine/docs/python>
3. George, J., 2014. Principles of Beautiful Web Design, 3Rd Edition.