Cloud Computing

Assignment 2

Anooz Prasai
3014202
prasaianooz@gmail.com
anooz.prasai@student.griffith.ie

Documentation

Table of Contents

Google App Engine app.yaml

Runtime environment of Python2.7 is used.

Url handlers

    /favicon.ico is handled with static file favicon.ico under /
    /static is handled with static directory “public”
    Anything else is served by routes in main.py WSGI Application initializers.

Webapp2 version 2.5.2 and the latest version of jinja2 templating engine is used.

Libraries Imported

urlparse - to parse request url for getting base_url
google.appengine.api users : for default users login and logout functionality
google.appengine.ext ndb : DB Datastore library
webapp2_extras sessions: sessions library provided by webapp2
Jinja2 : View templating engine
Webapp2 : python web framework compatible with google app engine for WSGI applications
os : to get cwd.

Architecture

MVC pattern followed.
Code organised into Handlers, Models and View packages.

Handlers: Routes to url are handled by handlers. Handlers gets requests and maintains data flow between view and model to achieve output
Model: Datastore classes and helper methods to handle business logic
View: Class to render view with

Basic Code Flow

- BaseHandler is a parent class of all the Handler classes
- Constructor in BaseHandler sets all parameters and handles user logged in authentication
- If user is logged in, constructs a user in AppUser Datastore
    - If user's email already exists in datastore, get user object
    - If user's email doesn't exist, insert in datastore and get user object
- Creates a new ViewHandler object with parameter self.template_values
    - Also initialises jinja2 environment with view files on views folder

Datastore Models

Summary

- ➢ Each logged in user should be an instance of AppUser.
- ➢ Each task board should be associated with AppUser so Taskboard Model holds created_by AppUser key
- ➢ Taskboards can have more members added to it. This is one to many relation from Taskboard to User.
- ➢ Also each user can be added to more than one taskboards. This is one to many relation from User to taskboard
- ➢ This sums up to many to many relation from Taskboard to AppUser
- ➢ This brings up TaskboardMember table, which holds Taskboard's Key associated members AppUserKey
- ➢ Finally Task should be associated with one taskboard, and can be created by any user and assigned to any user.

- ➢ So Tasks model should have Taskboard Key and Assigned_to AppUser's key and Created_by AppUser's key

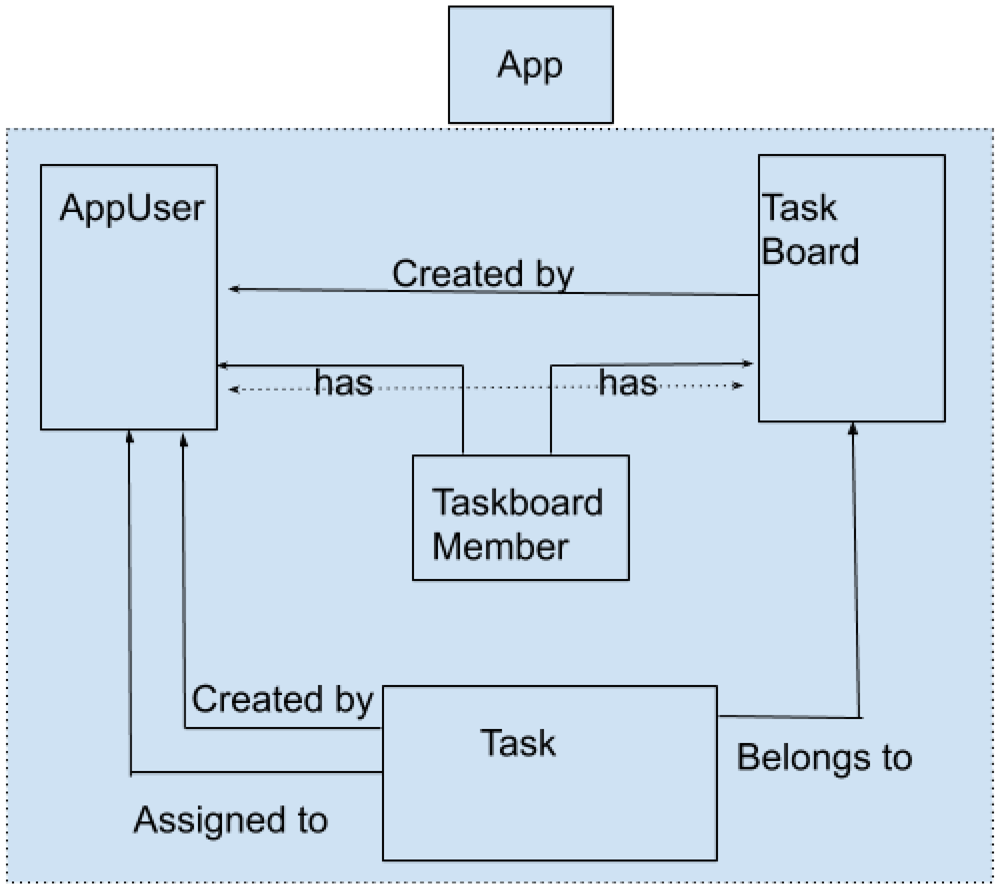

Fig. App Models Overview

AppUser

- App Users datastore model
- Email; datatype String; implies email of logged in user

Taskboard

- Taskboard datastore model
- title: datatype: String; implies title of taskboard,
- created_by: datatype: Key ; implies creator of taskboard,
- created_date: datatype: datetime.datetime; implies date of taskboard creation
- updated_date: datatype: datetime.datetime; implies date taskboard was updated

TaskboardMember

- TaskboardMember association table
- This table contains associaion within taskboard and User key
- Each association/record implies that user is member of the taskboard
- taskboard; datatype: Key; implies key of taskboard
- app_user: datatype: Key; implies key of associated member object from AppUserModel
- created_date: datatype: Key; implies of association made.

Task

- Tasks datastore model
- Taskboard; datatype: ndb.KeyProperty; implies the Task board which task belongs to
- title;datatype: ndb.StringProperty; implies title of task
- description; datatype: ndb.TextProperty; implies description of task
- due_date; datatype:ndb.DateTimeProperty; implies task's due date
- assigned_to; datatype:ndb.KeyProperty; implies user who the task is assigned to
- status; datatype:ndb.BooleanProperty; implies completed or ongoing task
- created_by; datatype:ndb.KeyProperty; implies user task was created by
- created_date; datatype:ndb.DateTimeProperty; implies date task was created on
- updated_date; datatype:ndb.DateTimeProperty; implies date task was updated on
- completed_date; datatype:ndb.DateProperty; implies date task was completed on

Backend API

Home Page

- GET /
- Returns homepage main view master.html
- Served by get method of handler MainHandler from handlers.MainHandler
- From MainHandler's get method using the view object created from BaseHandler, master.html file is passed into View's render method.

Show Task Boards

- GET /taskboards HTTP/1.1
- Dispatched to handlers.TaskboardHandler index method
- Gets all task boards authorised to current user.
    - Using models.TaskBoardModel.TaskBoardMethods get_all_authorised_taskboards() method
        - Get all membered taskboards objects
        - Fetch keys using python's map and lambda function
        - Get user's all created task boards
            - Taskboard datastore schema holds keys of taskboard's creator
        - Merge objects (created + membered)
        - Get all objects
    - Map each objects to the form required by view
        - Created_by key to email of user
        - Created_date to Y-m-d format
        - Count statistics of tasks in taskboard

Add/Edit Task board

- POST /taskboards HTTP/1.1
- Dispatched to handlers.TaskboardHandler post method
- Deserialize request payload with JSON string to python's object
- Check if add or edit operation is permitted to user using handlers.TaskboardHandler.is_post_authorised() method
    - If operation is Add operation; distinguished by 'id' key in params - authorised
    - If operation is Edit; distinguished by 'id' key in params - unauthorised
- If authorised
    - Insert if add operation using models.TaskboardMethods insert_taskboard()
    - Update if update operation using models.TaskboardMethods.update_taskboard()
- Send response in JSON string format implying success flag which is being parsed in view.

View Taskboard

- GET /taskboards/5733953138851840 HTTP/1.1
- Dispatched to handlers.TaskboardHandler get() method

- Check if operation permitted using handlers.TaskboardHandlers.is_get_authorised()
    - Get list of created and membered taskboard of user
    - Check if current task board in use is in the list
- Get Taskboard object by id
- Transform into format applicable for view
- Send this transformed data as a JSON object.

List Tasks in Task Board

- GET /taskboards/<taskboard_id>/tasks, HTTP/1.1
- Dispatched to handlers.TaskHandler.get_all_taskboard_tasks() method
- handlers.TaskHandler.get_all_taskboard_tasks() accepts taskboard id
- Operation authorised with handlers.TaskHandler.is_get_authorised(taskboard_id)
    - Get all created and membered task boards of current user
    - Check if used taskboard is in this list
- Get all tasks in taskboard with models.TaskMethods.get_all_tasks_by_taskboard(taskboard_id) method
    - Queries Task datastore with taskboard Key = key with provided id
- Transforms all task objects into format accepted in view
    - Id: numerical id of task
    - Taskboard_id: numerical id of taskboard
    - Title: task title
    - Due_date: in y-m-d format
    - Due_date_remaining: remaining or overdued days
    - Overdue: if current date is greater than or equals to due_date
    - Assigned_to_email: email of AppUser task is assigned to
    - Assigned_to: numerical id of user task is assigned to
    - Status: in boolean if completed or not
    - Status_text: either Completed or On-Going
    - created/updated/completed dates in Y-m-d format
    - Completed_date_text in number of days before or after due date
- Print all this data with response success.

List Members in Task board

- GET /taskboard_members/<taskboard_id>, HTTP/1.1
- Dispatched to handlers.TaskboardMemberHandler.index()
- Check handlers.TaskboardMemberHandler.is_get_authorised() by taskboard id
    - Get all membered and created taskboards of current user
    - Check if current taskboard is in list

- Fetch data objects from TaskboardMemberModel where TaskboardKey = current taskboard key.
- Create response dictionary object representing success flag and data
- Send response as json string.

Save/Update Task in Task board

- POST /tasks, HTTP/1.1
- Dispatched to handlers.TaskHandler.post()
- Load request params to python object using json.loads from json package
- Check if add task to task board is permitted to current user by taskboard id
    - Get all membered and created tasks of current app user
    - Check if taskboard is in membered and created task list
- Validate all input fields by handlers.TaskHandler.validate(params)
    - Check if taskboard_id is set
    - Check if title, description field is not empty
    - Check if Assigned_to has a user in member list
    - Check if due_date has valid date format
- Save or update with task id flag
- Create response dictionary with success flag and data
- Send response

Mark task as Completed

- POST /tasks/mark-complete
- Dispatched to handlers.TaskHandler.mark_complete()
- Load request payload to python dictionary using json.load
- Get task
- Set Task.status = true
- Save task
- Create dictionary with success flag
- Send dictionary as json response

Mark task as On-Going

- POST /tasks/mark-ongoing
- Dispatched to handlers.TaskHandler.mark_ongoing()
- Load request payload to python dictionary using json.load
- Get task
- Set Task.status = false

- Save task
- Create dictionary with success flag
- Send dictionary as response

Add member in task board

- POST /taskboard_members, HTTP/1.1
- Dispatched to handlers.TaskboardMemberHandler.post() method.
- Load request payload into python object with json.loads()
- Validate with handlers.TaskboardMemberHandler.validate() method
    - Check if user is selected
    - Check if user is already a member
- Insert in TaskboardMember
- Build response dictionary with success flag and data
- Send response

Delete member in task board

- POST /taskboard_members/delete, HTTP/1.1
- Dispatched to handlers.TaskboardMemberHandler.delete() method
- Load request payload into python dictionary with json.loads()
- Delete record with TaskboardMemberMethods.delete_taskboard_member()
    - Fetch the record with taskboard_id and app_user_id association
    - Get key of record
    - Execute delete operation
- Get all task with taskboard_id and app_user id
- Update tasks as unassigned
- Create dictionary with success flag and data
- Send dictionary as json response to request

Delete tasks

- POST /tasks/<task_id>/delete, HTTP/1.1
- Dispatched to handlers.TaskHandler.delete_task() method
- Load request payload into python dictionary using json.loads()
- Get task object from Task datastore using TaskMethods helper function get_by_id()
    - Get task from Task datastore using Task.get_by_id()
- Get Taskboard key from task object
- Check if taskboard is authorised for current user

    - Get all membered and created taskboards for current user
    - Check if current taskboard is in the list
- Run delete operation for task
- Create response dictionary object with success flag
- Send response as json string

Delete Task board

- POST /taskboards/delete, HTTP/1.1
- Dispatched to handlers.TaskboardHandler.delete_taskboard
- Load request payload into python dictionary using json.loads()
- Validate request
    - Check if taskboard_id parameter is set
    - Check if there is any task in taskboard
    - Check if there are any members in taskboard
- Build validation dictionary object
- If validation object has error
    - Create dictionary with success false
- Delete Task board instance from datastore
- Create dictionary with success flag
- Send json as response

View Architecture

Summary

- ➢ AngularJS is used to build single page application for TaskManagement App
- ➢ Architecture is can be segmented as Component Controllers, Model and Partials
- ➢ Component Controllers
        - acts as controller for each portions of view
        - Sidebar, Taskboard, TaskboardInfo components are 3 components
- ➢ Model
        - Are dynamic data that changes in Dynamic HTML
        - Separate models for sidebar, Taskboard, TaskboardInfo
        - Models in scope are changed in Component Controllers
- ➢ View
        - Views are partials with UI where models are reflected to user

        - Views are again manipulated with component controllers

- Routing with ngRoute is carried out to route through components

Root App

- Whole HTML start tag is taken as root app
- Root app is initialised as angular app
- This accounts for $rootScope which is base model for view
- $rootScope can be injected into any sub components with Dependency Injection that is done within angularJS internals

Sidebar Component

- Sidebar is positioned in left and contains all navigation link
    - In our case only one which is Taskboards and links to /taskboards
- Room for scalability is achieved
- Sidebar component is rendered within custom tag <side-bar>
- Font-awesome icons are used for icons

Taskboard Component

- Taskboard component is displayed in right main content.
- Requests GET /taskboards, HTTP/1.1
- It has Taskboards array as model, and is overridden with json object response from server
- Task boards are populated in tabular format with view edit and delete operations.
    - Taskboard task's figures are shown in blue, green, yellow and grey badges respectively for total tasks, completed tasks, incomplete tasks, and total completed today tasks.
- Add Taskboard button on right top of table is binded with Modal Popup.
- Modal pops up with taskboard form to save.
- Same goes with Edit Taskboard, which sends XHR request to server
    - Requests GET /tasksboards/<taskboard_id>, HTTP1.1
- Delete button is binded to taskboard.creator in reponse object
    - Delete button is also binded with confirmation modal.
    - On confirmation,
        - POST /taskboard {taskboard_id:<taskboard_id>} is requested

Taskboard Info Component

- Taskboard Info component comprises detailed view of taskboard
- Displayed on right main content within <ng-view> tag
- Requests GET, /taskboard/<taskboard_id>/tasks, HTTP/1.1
- Accepts text/json object
- Populates response data into table with operation detail, edit, mark as ongoing, mark as complete, delete operations
    - Status is represented with spinner font-awesome icon in red color
        - To represent task in incomplete
    - Due date is represented with Y-m-d value
        - And number of days remaining or overdue in green and red badges respectively
    - Status complete is represented with font-awesome check icon in green color
        - And number of days before or after due date is shown below icon to represent completion date.
- Detail button is binded with popover to show detailed view of task
- Add/Edit Task button is binded with popover to add/edit and submit task
    - Status option is set only on edit operation
    - As task when created by default is in incomplete operation
- Mark as ongoing
    - Requests GET /tasks/mark-ongoing, HTTP/1.1
    - Accepts text/json with mark-ongoing success flag
    - Changes button to mark as complete on success response
- Mark as complete
    - Requests GET/tasks/mark-complete, HTTP/1.1
    - Accepts text/json with mark-complete success flag
    - Changes button to mark as ongoing on success response

- Add Member button is binded with popover to add/edit and submit member
    - Requests GET /app_users, HTTP/1.1
    - Accepts text/json object with all users
    - Shows all users in multiselect dropdown
    - User can ctrl + select multiple options
    - Form submit triggers
        - POST /taskboard_member HTTP/1.1
        - Accepts text/json success object
- Requests GET /task-members, HTTP/1.1
    - Accepts text/json object with all members in taskboard
    - Shows in list format with delete button. Font awesome trash icon
    - Delete button is binded with taskboard.creator value

        - Shows only when taskboard.creator = true
    -