

## CINS 465: Project 1: Organizer

This is an individual assignment. **Do your own work. Do not copy / paste anything from any outside source** You may brainstorm with or get help on technical problems from others, but all submitted work must be your own. It is highly recommended that you use Piazza as a tool for remote group collaboration. In addition to asking questions and reading answers on Piazza, please read other's questions and post answers if you know them.

### Objectives

- Implement a full stack web service from scratch using all of the tools and technologies we have learned this semester: HTML, CSS, Javascript, JQuery, Bootstrap, Django & Python.

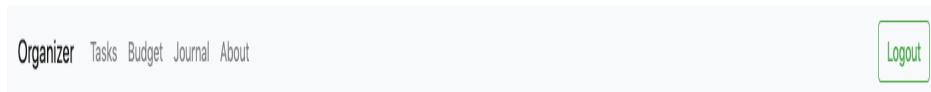
### Design / UI Requirements

- The name of the service is Organizer. There are three main features of Organizer: Tasks (a personal to-do list), Budget (a personal expense tracker), and Journal (a personal journal entry manager). Organizer also supports Join, Login / Authentication, Logout, About and a shared navigation UI. The Organizer home page is a Dashboard with a visualization of the state of the Tasks, Budget and Journal.
- Break down by Django Application.

- Core

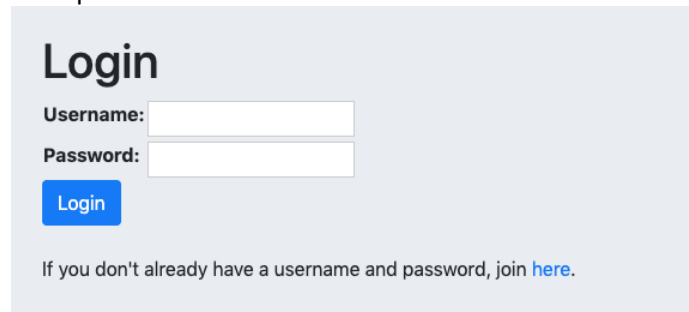
- **Navigation Bar**

- Shared navigation bar with items for **Organizer** (home page), **Tasks**, **Budget**, **Journal** and **About**, with a **Logout** button (only visible when user is logged-in).
    - Example UI:



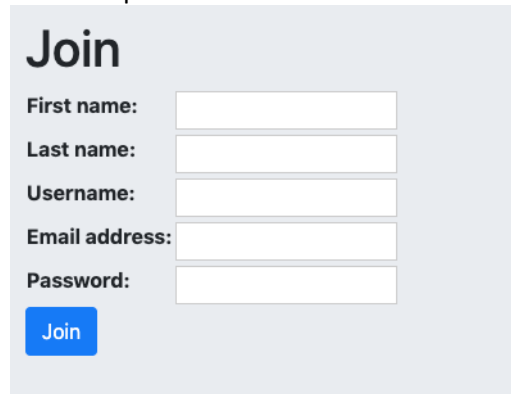
- **Login Page**

- A page that is displayed when navigating to any site link (other than **About** or **Join**) when the user is not logged in. This page must have a link to the **Join** page.
    - Example UI:

The image shows a login form with a light gray background. At the top, the word "Login" is written in a large, bold, dark gray font. Below it, there are two input fields: "Username:" and "Password:". Below the "Password:" field is a blue button with the text "Login" in white. At the bottom, there is a line of text: "If you don't already have a username and password, join [here](#)."

- **Join Page**

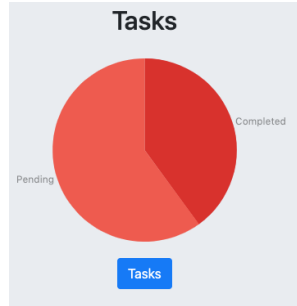
- A page that is accessible from a link on the **Login** page.
    - Example UI:

The image shows a join form with a light gray background. At the top, the word "Join" is written in a large, bold, dark gray font. Below it, there are five input fields: "First name:", "Last name:", "Username:", "Email address:", and "Password:". Below the "Password:" field is a blue button with the text "Join" in white.

- Dashboard (home page)

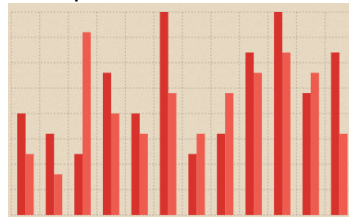
- The home page displays a dashboard for each of the main Organizer applications: Tasks, Budget and Journal in a three column format, using the bootstrap grid feature. Each dashboard summarizes the current data for each application.
- The Tasks dashboard area is a pie chart showing completed vs uncompleted tasks.

- Example UI:



- The Budget dashboard displays a bar chart showing actual vs. budget for each budget item.

- Example UI:



- The Journal Dashboard displays the total number of journal entries and the number of days since the last journal entry.
- Each application dashboard has a button that allows the user to navigate to that application (this is only shown for the Tasks dashboard above but each dashboard should have a similar button).
- It is suggested to use chartist.js (<https://gionkunz.github.io/chartist-js/>) to render the dashboard charts.

- Tasks

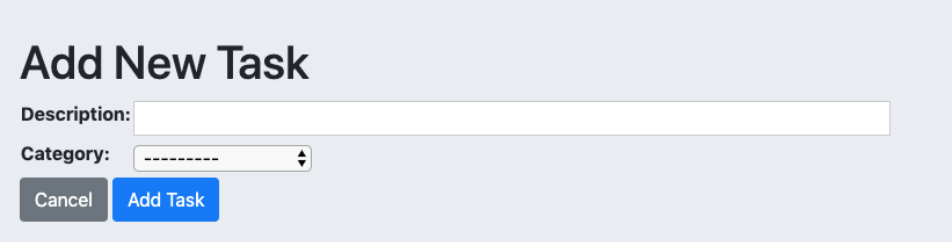
- View Task List (page)

- A page containing a table showing all defined tasks. Each task consists of: **Description**, **Category**, **Completed**. For this project we will use a static set of task categories: **Home**, **School**, **Work**, **Self Improvement** and **Other**. There is one row in the table for each currently defined task. Each row displays buttons to Edit or Delete that task. There is a button just below the table to Add a new task. There is a checkbox control just above the table to show / hide completed tasks. Note: This show/hide setting must be stored in the user model data so that it persists whenever the Tasks page is visited.

- Example UI:


Tasks			
Hide Completed Tasks: <input type="checkbox"/>			
Description	Category	Completed	Action
Yoga	Self Improvement	Yes	<button>Edit</button> <button>Delete</button>
Meditation	Self Improvement	Yes	<button>Edit</button> <button>Delete</button>
Practice Piano	Self Improvement	No	<button>Edit</button> <button>Delete</button>
Laundry	Home	No	<button>Edit</button> <button>Delete</button>
CINS: 465: Start Project 1	School	No	<button>Edit</button> <button>Delete</button>
<button>Add Task</button>			

- Add Task (page)
  - When the user selects the Add Task button, the Add Task page is displayed.
  - Example UI:



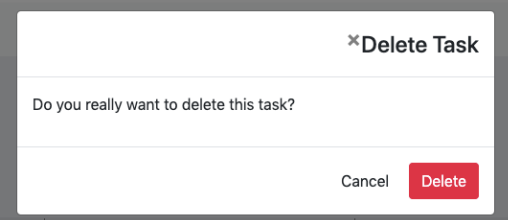
The 'Add New Task' form features a title 'Add New Task' at the top. Below it is a text input field labeled 'Description:'. Underneath the description field is a dropdown menu labeled 'Category:' with a dashed line indicating a selection. At the bottom of the form are two buttons: a grey 'Cancel' button and a blue 'Add Task' button.

- Edit Task (page)
  - When the user selects the Edit button for a given task, the Edit Task page is displayed.
  - Example UI:



The 'Edit Task' form has a title 'Edit Task'. It contains a text input field for 'Description:' with the value 'Laundry'. Below it is a dropdown menu for 'Category:' with the value 'Home'. At the bottom are two buttons: a grey 'Cancel' button and a blue 'Update Task' button.

- Delete Task (button in each task list table row)
  - When the user selects the Delete button for a given task, the Delete Task page is displayed. A confirmation dialog box is displayed.
  - Example UI:



The 'Delete Task' dialog box has a title '\*Delete Task'. The main text asks 'Do you really want to delete this task?'. At the bottom right are two buttons: a grey 'Cancel' button and a red 'Delete' button.

- Note: The dialog shown above is a custom implementation using bootstrap. See <https://getbootstrap.com/docs/4.0/components/modal/> for details.
- Show / Hide Completed Tasks (checkbox above task list table row)
  - Selecting this checkbox should refresh the page, update the user model with this setting, and re-render the table showing or hiding completed tasks accordingly.

- Budget

- View Budget List (page)

- A page containing a table showing all defined budget items. Each budget item consists of: **Description**, **Category**, **Projected**, **Actual**. **Projected** and **Actual** are in whole dollar amounts. Only allow positive integers. Use the following static set of budget categories: **Food**, **Clothing**, **Housing**, **Education**, **Entertainment**, **Other**. There is one row in the table for each currently defined budget item. Each row displays buttons to **Edit** or **Delete** that item. Implement a similar UI as shown for the View Task List page shown above. Display the projected surplus or deficit (in whole dollars) just below the budget table.

- Add (page)

- When the user selects the Add button, the Add Budget Item page is displayed. Implement a similar UI as shown for the Add Tasks page shown above.

- Edit Budget Item (page)
  - When the user selects the Edit button for a given item, the Edit Budget Item page is displayed. Implement a similar UI as shown for the Edit Tasks page shown above.
- Delete Budget Item (button on each budget list table row)
  - When the user selects the Delete button for a given task, the Delete Task page is displayed. A confirmation dialog box is displayed. Implement a similar UI as shown for Delete Task, shown above.
- Projected Budget Surplus or Deficit (display this dollar amount below the budget list table)
- Journal
  - Apply the same general principles and techniques described above to design your Journal application pages. Each journal entry must have **Date, Description & Entry** fields. You don't need to implement categories for journal entries. You only need to display the **Date** and **Description** fields on the View Journal List page. Django provides support for date form and model fields. Automatically fill-in today's date for the user when they add a new journal entry. Allow them to change the date on the **Edit Journal Entry** page. When adding or editing journal entries, use a textarea control (to provide lots of room to type the entry). Use a custom handwriting style font for the journal entry text area see <https://fonts.google.com/?category=Handwriting>
  - View Journal List (page)
  - Add Journal Entry (page)
  - Edit Journal Entry (page)
  - Delete Journal Entry (button on each journal list table row)
- About
  - A few paragraphs about project 1, how you implemented it, and what tools / technologies were used.
  - A paragraph about you, including a recent photo.

#### General Requirements

- All persistent data is stored on the backend using Django models. For this project, use the default backend database, sqlite.
- Choose your own bootstrap & CSS styles / colors to make your project 1 site your own. Make sure your choices are functional for the user (all text and UI controls must be easy to see and use for example).
- Form and table controls must be aligned and centered as shown in the screenshots above. I've found the best way to do this is via table layouts for forms.
- Each user has their own private set of Task, Budget & Journal entries (when I login, I only see my entries). This mostly means you need to include the user id in each of your Task, Budget and Column models.
- You must use git and github as your source control management system for Project 1.
- **You must submit a single zip or tar file** containing your entire Django project such that when I open it I can navigate into your assignment6 project directory and start your web server using manage.py.

#### Suggested "Plan of Attack"

- The following is a suggested sequence for getting started on this project. Much of the information provided here can be derived from the posted lecture content. These are essentially the notes I took when I implemented my version of this project and are meant you get you off to a quick and good start.
- Join Github if needed, at <https://github.com/>. Note that to move on to the next project you must have Project 1 in git, remoted to github. This will be needed for deployment.
  - Make sure you have git installed on your dev machine.
  - Clone your new git repository to your dev machine using the 'git clone' command. If you're new to GitHub & Git, see <https://help.github.com/en/github/creating-cloning-and-archiving-repositories/cloning-a-repository>
- Create a new Django project called Project 1 in your new local git directory.
  - django-admin startproject project 1
- Within your Project1 Django project, create each of these Django applications: core, tasks, budget, journal.
  - python3 manage.py startapp tasks
  - ...

- Configure INSTALLED\_APPS in settings.py to reference each app you just created.
- Create main view for each app, in the views.py files, which prints a basic http response, e.g. for the journal view you might do this as a starting point:
  - `return HttpResponse("Main journal page")`
- Note: It is recommended that the core app handle the home page (dashboard), join, login, logout and about so you can go ahead and add views for each of those now, or do it later when ready.
- In urls.py, connect your views to entrypoints, e.g.
 

```
from core import views as core_views
from journal import views as journal_views
from tasks import views as tasks_views
from budget import views as budget_views
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", core_views.home, name='home'),
    path('journal/', journal_views.journal, name='journal'),
    ...
```
- Start project (python manage.py runserver) and test each end-point that you defined in urls.py. You should have one for each of the four views that you created, one for each app.
- CHECKPOINT 1: Good time to commit to git and push to github.
- Create templates directory under **outer** project directory
- Create a sub-directory for each app, within the new templates directory
- In settings.py, define TEMPLATES\_DIR after BASE\_DIR
- In settings.py, add TEMPLATES\_DIR to TEMPLATES dictionary's DIRS list
- Create one html template file for each app, within the appropriate template sub-directories
- Modify each view to render from the template files
- Retest each URL endpoint
- CHECKPOINT 2: Good time to commit to git and push to github.
- Create static sub-directory under **outer** project directory
- Create CSS, JS and IMG sub-directories under new static directory
- Configure STATIC\_DIR in settings.py, below TEMPLATES\_DIR
- `STATIC_DIR = os.path.join(BASE_DIR, "static")`
- Create STATICFILES\_DIRS list at the bottom of settings.py
- `STATICFILES_DIRS = [`
- `STATIC_DIR`
- `]`
- Create shared static CSS, JS and IMG content files as needed, include from templates where needed.
- Create shared links template in core app, include in head section of all view templates
- Create shared navbar content, include at top of body of all view templates
- Test navbar
- CHECKPOINT 3: Good time to commit to git and push to github.
- Implement join, login & logout. You can peek at my Sudoku implementation to see how I did this: Course Content > Lectures > Week 8 > project1.withLogin.tgz Download that file in a clean directory somewhere, unpack it, and then open up the main project directory in Atom. See app2 > forms.py, views.py.
- Add forms.py to your core app
  - `from django import forms`
  - `from django.contrib.auth.models import User`
  - Define join and login forms tied to build-in User model.
- Run model migration commands
  - From main project directory
    - `python manage.py migrate`
    - `python manage.py makemigrations`
    - `python manage.py migrate`

- Add logout button to navbar, only visible when user is not logged-in
- Redirect all end-points to login end-point when user is not authenticated, except for join and about end-points.
  - Hint: Use python decorators like I did in my Sudoku views.
- Implement view functions for join, login, logout
- Test join, login, logout
- CHECKPOINT 3: Good time to commit to git and push to github.
- Next I'd implement the Tasks pages, then Budget, then Journal, then the Dashboard on the home page and finally the About page. Each of these apps will need their own forms and models.
- I'll be showing examples in-class that will illustrate how to implement the table views and the add, edit, delete functionality.
- Budget your time wisely. Work on this project every day for at least an hour or two. You can probably expect about 2 to 4 hours to get to CHECKPOINT 3 above. Implementing the Tasks pages will likely take another 4 to 8 hours, especially if you haven't done these sorts of web based interfaces before. Then after that the rest should fall into place more quickly as you will apply the same techniques for the Budget and Journal pages. The dashboard will likely take another 2 to 4 hours. I'll show examples in class on how I did the charting using chartist.js.
- Create your demo video.
- Submit your project zip/tar file and video demo to Course Content > Projects > Project 1 in blackboard.
- Just FYI: My plan for Project 2 is to add a few enhancements to the project 1 features, like custom categories for Tasks and Budgets for example, and then deploy Project 2 into a cloud provider like Google cloud.
- Good luck! Be sure you use Piazza to collaborate and problem solve with your classmates (no code sharing allowed but small snippets / examples are fine).