CS 416

Homework 4

1. Objectives

The purpose of this assignment is to get you familiar with creating data-driven web app and improve your skills in the areas below:

- Using HTML, and CSS, Bootstrap to create web pages that are responsive for various screen sizes
- Using Django to create a data-driven web app

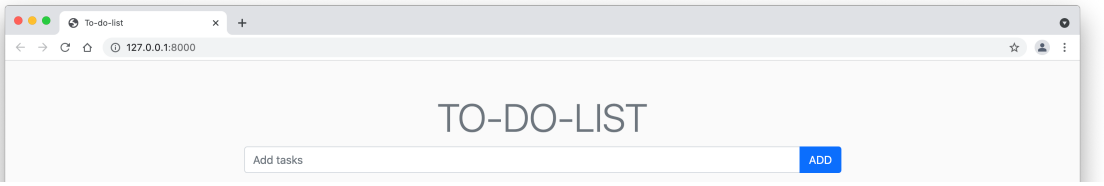
2. What to do?

In this assignment, you will create an app named **Todo** that can be used to create and store a to-do list. Using **`Django`**, you will implement the required functionality. Additionally, using **`Bootstrap,`** you will design the web pages in this app exactly as shown in the provided screenshots and video below.

2.1 Overall Behavior

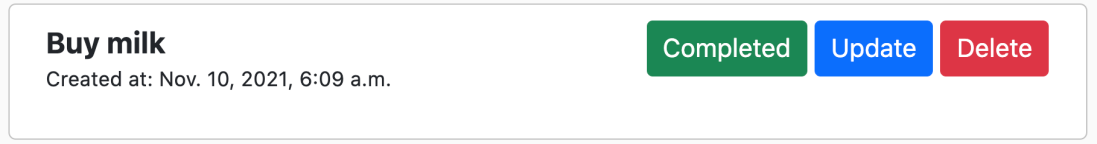
The app must have 3 different pages: **`index.html, update.html`** and **`delete.html`** (using Django's Template inheritance, you can extend each of these three pages from another page such as **`base.html`**).

On the index page, the user is presented with a textbox and a button (e.g., ADD) to enter a new task. When the add button is clicked, the entered task should be saved into a database on the backend.



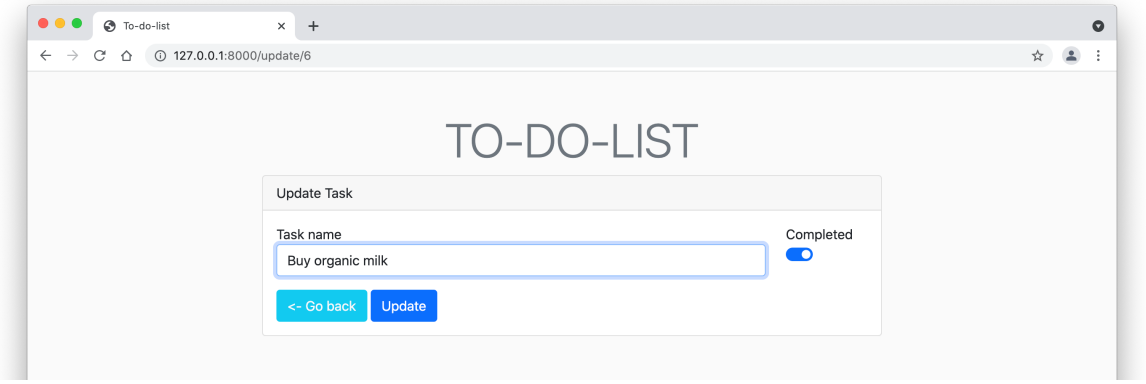
When a task is created, the task status (e.g., Boolean type) will be "not completed" on the backend. However, the user will be able change the status of the tasks as "completed" later on the index page by clicking "Complete" button for each task.

Each task should be displayed on the index page as shown below. There should be 3 buttons (completed, update and delete) for each task that is not "completed".

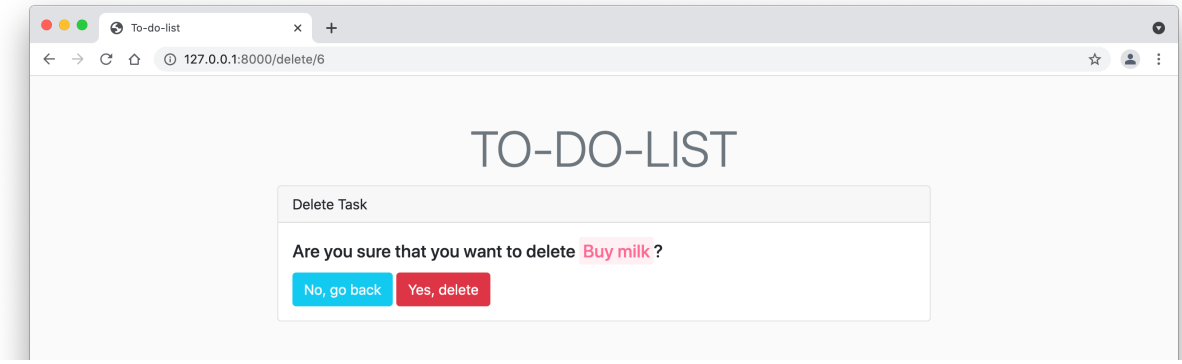


1. Completed button marks the task as “completed” and the appearance of this task should be updated as shown below. Additionally, “Completed” button should not be shown for a completed task (see the image below).

2. Update button opens the update page where the user can update the task that was clicked as shown below. In particular, there should be a textbox loaded with the text that the user clicked on the index page. On the update page, the user can edit the text as well as change the completed status using a switch. When the update button is clicked, the task is updated and the user is redirected to the index page. When the “Go back” button is clicked, the user is redirected to index page **without updating the task**.



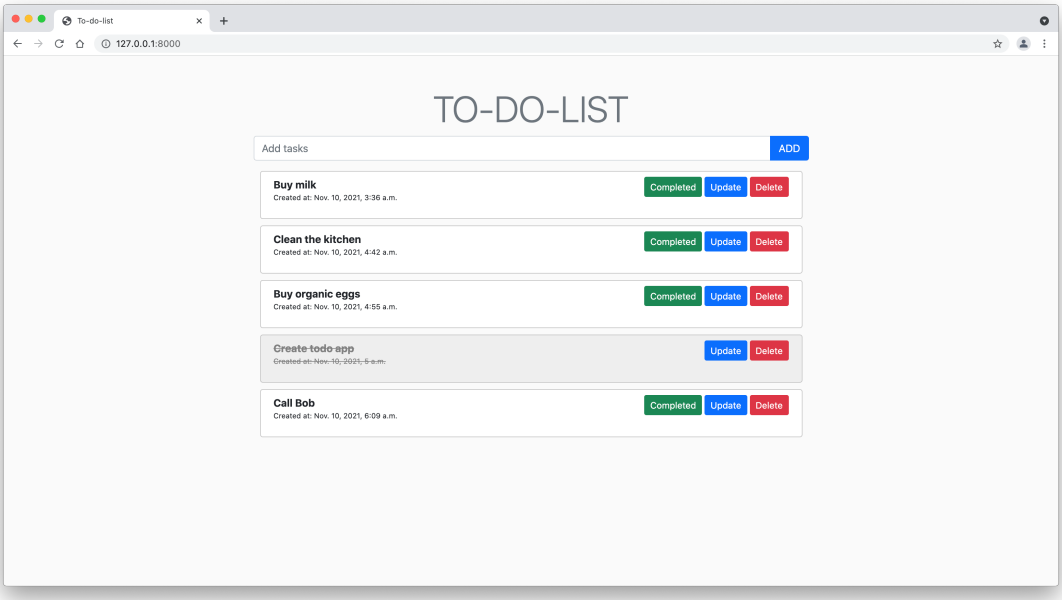
3. Delete button opens the delete page where the user can delete the task permanently as shown below. When the delete button is clicked, the task is deleted and the user is redirected to the index page. When the “No, go back” button is clicked, the user is redirected to index page without deleting the task.



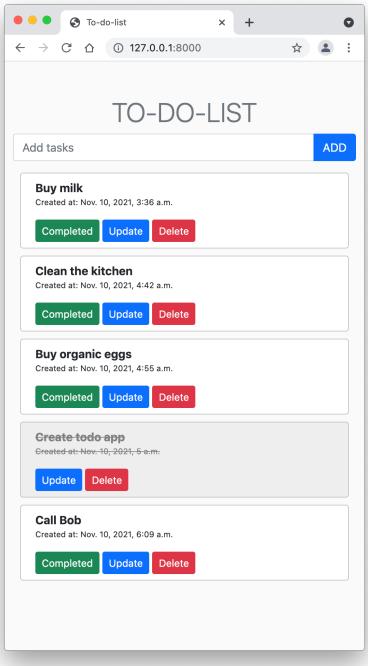
2.2 Overall appearance

The screenshots below show the website you need to create in this assignment. You can **watch the video** showing more detailed look and behavior of this page (see the video on Blackboard).

Large Screen:



Small Screen:



2.3 Appearance Details

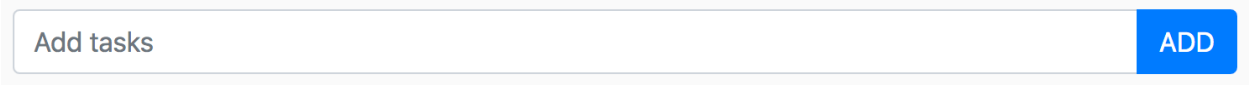
Overall site (the styles below should apply to all of the three pages: index, update and delete):

- Each page **body** should have a background-color of `#fafafa`.
- The content in each page should be justified to the center.
    - Hint: Bootstrap grid system can be used with a container, row and column classes.
        - https://getbootstrap.com/docs/5.1/layout/grid/
- There should be a margin of 3rem from the top and bottom
    - Hint: `my-5` class from Bootstrap can be used for the container.
- All the pages should show a header: **TO-DO-LIST** as shown in the screenshot above
    - This header (e.g., h1) should be gray color and centered
        - Hint: `display-4` class for a heading to stand out, `text-center` class for centering it, and `text-second` class for changing the color to gray can be used.

Index page:

Add Task:

- There should be a textbox and add button to add new tasks. The button should appear right next to the textbox as shown below. The textbox should have a placeholder text (e.g., Add tasks).
    - Hint: `input-group` class from Bootstrap can be used to achieve functionality described above.
        - https://getbootstrap.com/docs/5.1/forms/input-group/#button-addons

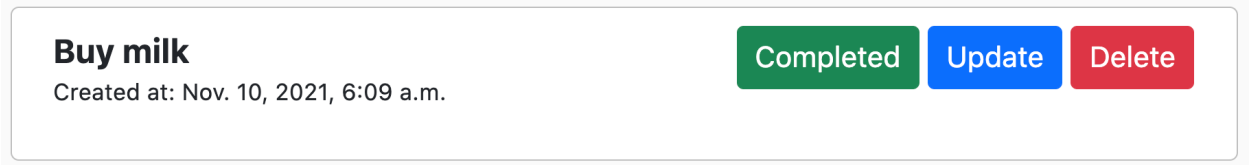


Note: On the client side (i.e., index.html), you can create a form with a button that sends the entered data to a URL that your server can process. If you are using a form that includes a button tag, you should change type attribute of your button tag from “button” to “submit” so that when the button is clicked the form data is sent to the server, otherwise button type will not trigger submitting the form.

```
<button class="btn btn-primary" type="submit">ADD</button>
```

Tasks:

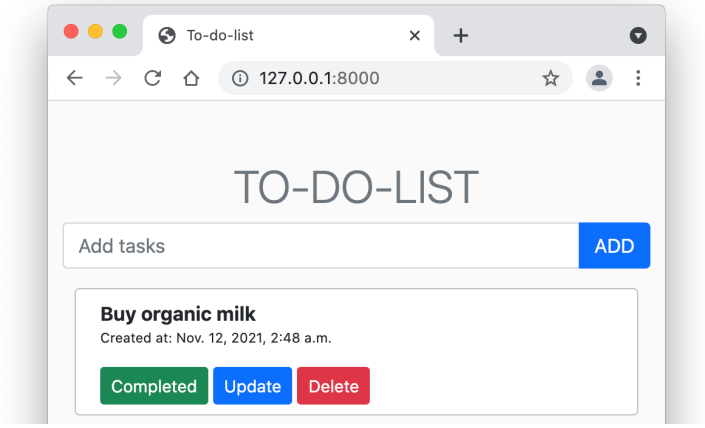
- Each task that the user enters should be displayed on the index page as shown below.



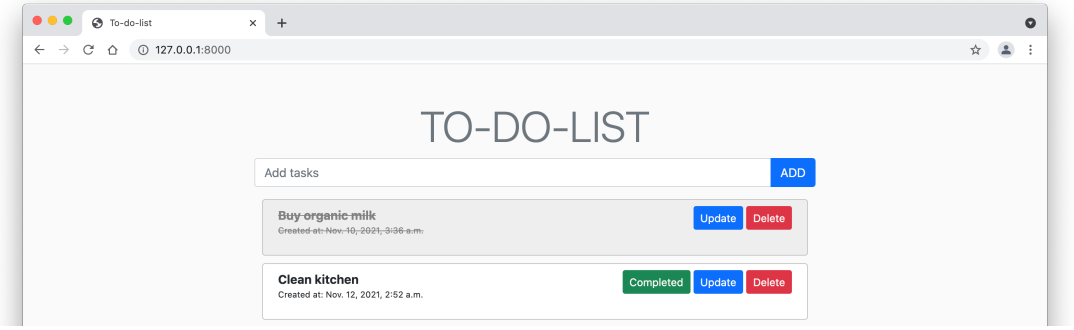
- There should be 3 buttons (Completed, Update and Delete) for each task that are not completed.

- The three buttons should be aligned to the right, and task name and creation date should be aligned to the left.
    - Hint: For each task, you can create a row that consists of two columns (e.g., two divs each with col-md-6). The first column can contain task name and creation date, and the second column can contain the three buttons.
- Task name should be bold.
- "Created at" should be a small text (0.7rem) with gray color.
- Each task div should have the following styles:
    - a silver solid border that is 1px thick
    - 0.25rem border-radius
    - White background color
    - 10px margin and padding from all sides

- When the screen size is less than 768px (i.e., medium screen size breakpoint), the buttons should appear underneath of the task and date. The buttons should be aligned to the left when the screen size is small, and anything above a small screen size the buttons should be aligned to the right.
    - Hint: You can take a look at the usage of responsive text alignment classes at the link below:
        - https://getbootstrap.com/docs/5.1/utilities/text/#text-alignment



- Completed task should be shown as below. In particular, when the user clicks the "Completed" button, all text (i.e., task and date) should be strikethrough and the background color should change to a grayish color (e.g., #eeeeee), and there should not be a "Completed" button for a completed task.



Hint: If you have a column in your table called completed, you can use this boolean variable in your template as well. Also, you can create two different classes in your CSS file. These classes can be given to div tags depending on the completed status as shown below. Please note that this is not a complete code, you need to fill the rest of the code, but hopefully it will give you an idea of how this can be accomplished.

index.html

```
{% for task in tasks %}
        {% if task.completed %}
           <div class="row completed" >
           ...


       {% else %}
           <div class="row not-completed" >

           ....
         {% endif %}
      {% endfor %}
```

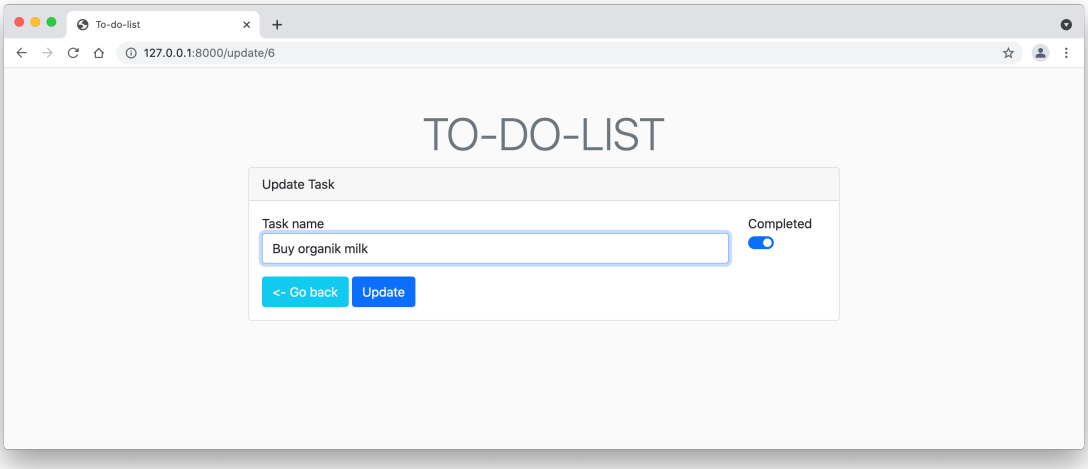
CSS

```
.completed {
  background-color: gray;
  text-decoration: line-through;
  ...
}

.not-completed {
  background-color: white;
   ...
}
```

Update Page:

- In the update page, there should be a textbox, switch and two buttons: “Go back” and “Update” as shown below.
    - `Hint:` `card` class from Bootstrap can be used to style as shown below
        - https://getbootstrap.com/docs/5.1/components/card/#header-and-footer

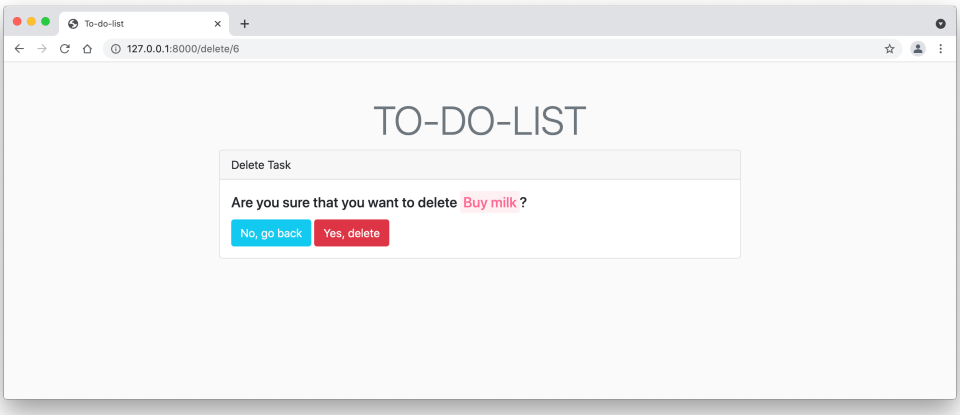


Hint: Using Django’s form can make the implementation of this part easier. On your template, you might need to render the fields manually to make “task name” and “completed” switch appear side by side (e.g., use Bootstrap’s grid system) (you can see an example in Lecture#11, slide title with “Rendering fields manually”). The following can give you an idea of how a form field can be accessed manually in your template.

```
<div class="form-check form-switch">
    {{ form.completed }}
</div>
```

Delete Page:

- Similar to update page, `card` class from Bootstrap can be used to style as shown below.
- This page should have a statement asking the user delete the task that was clicked on the delete button on the index page. There should be two buttons: go back and delete.
- The task that is being deleted (e.g., Buy milk) should be styled as follows:
    - text color and background color of #fe6993, rgba(254, 105, 147, 0.1), respectively.
    - 3px border radius
    - 0.2em padding
    - line-height of 2em



Some helps creating the app in Django

Models

You can create a table (e.g., Todo) with 3 columns as follows:

1. **`task_name`:** should be a charField with max length 200
2. **`completed`:** should be a BooleanField with default value False

```
completed = models.BooleanField(default=False)
```

3. **`created_at`:** should be DateTimeField with auto_now_add value True to keep track the creation date of the task

Forms

You can create a form class to handle generating form elements. For example, the following code can be placed in `forms.py` (create a python file (forms.py) and then place the code in it), and TaskForm (it can be a different if you want) can be used in `views.py` to generate, save and validate form elements. You can also use widgets in this class to handle rendering the widget as HTML and style the form elements (see the following link for more information).

- https://docs.djangoproject.com/en/3.1/ref/forms/widgets/#styling-widget-instances

```python
class TaskForm(forms.ModelForm):
    class Meta:
        model = Todo
        fields = '__all__'

        widgets = {
            'task_name': forms.TextInput(attrs={
                'class': 'form-control',
                'placeholder': "Add tasks"
            }),
            'completed': forms.CheckboxInput(attrs={
                'class': 'form-check-input',
                'role': "switch"
            })
        }
```

This represents the model (i.e., table name in the database) so make sure to import your model at the top

Column name in your model. Change it accordingly, if you use a different name

The widget allows to add custom attributes for the input elements in the form. For example, you can add Bootstrap classes (e.g., form-control or form-check-input) to the generated input elements. Placeholder attribute can be changed here as well. Whenever this form is rendered on your template, input tags will have these attributes.

This can be used to style the checkbox so that it appears as a switch on the update page. However, you may still need to encapsulate the input element in a div with the following classes on your template to properly show the switch button. Please see the link below.

- https://getbootstrap.com/docs/5.1/forms/checks-radios/#switches

```
<div class="form-check form-switch">
```

Views

You can create 5 view functions as follows:

- **index**: this function can retrieve all the tasks from the database and render the index page with the data.
    - **Hint**: If you are using Django forms, you can pass both the form instance and the tasks via context to your template.
- **add**: this function can be executed when the user enters a new task on the index page. This function can also be used to save the data into the database. Towards that, using Django forms as explained above can make it easier to validate and save form data.
    - **Hint**: On the client side, you can change the action attribute in the form tag in order to send the form data to your add method as shown below (I assume you gave a name to your add url as “add”)

```
<form action="{% url 'add' %}" method="post">
```

- **delete**: this function can take task id as an argument and get the corresponding record from the database and then delete it. On the client side, you can use a form and include the task id in the action attribute so that the delete function in your views receive the id of the task to be deleted.

    - `<form action="{% url 'delete' task.id %}" method="post">`

- **update**: this function can take task id as an argument and get the corresponding record from the database and then update it. Similar to add function, using forms in this function can make it easier to validate and save form data.
- **complete_task**: this function can take task id as an argument and get the corresponding record from the database, update its completed column as True and save it. This method does not have to render a page instead it can simply redirect to the index page.
    - **Hint**: On the client side, id of the task can be received as a parameter without needing to send it via a form. For example, you can use a link where href carries the task_id to the complete_task method in views.py.
        - `<a class="btn btn-success btn-sm" href="{% url 'completed' task.id %}">Completed</a>`

Urls

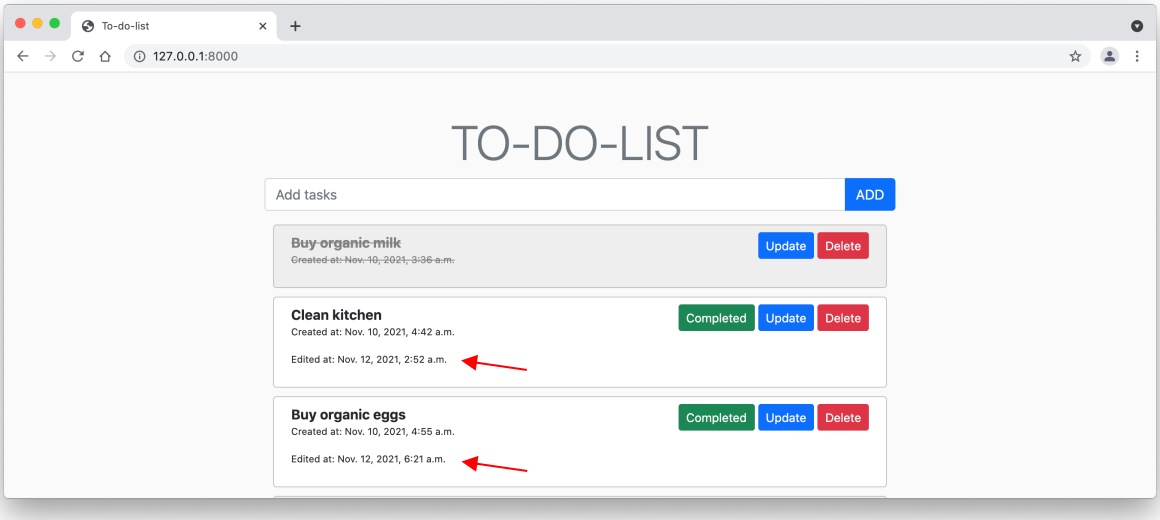
You can create 5 urls/paths one for each of the views' functions explained above.
The update, delete and complete urls/paths can capture the task id from the url so that the corresponding view functions can use the task id when updating, deleting as well as marking the task as completed.

- For instance, for delete you can use the following.
    - `path('delete/<str:task_id>', views.delete, name="delete"),`

Extra Credit (OPTINAL)

Up to **10** bonus points will be given if you complete the additional tasks below.

- **Deploy your application on a cloud hosting environment (7 points)**
    - Deploy your website on a cloud hosting environment such as PythonAnywhere. I have provided information on how to deploy your Django web app on your local computer to PythonAnywhere using GitHub (see the slides "**Cloud Deployment**" under Lectures on Blackboard). Note that PythonAnywhere is probably the easiest way to deploy your Django app. However, if you have issues with deploying your website or prefer another cloud hosting environment such as Heroku, AWS, etc., feel free to use one of these as well.
- **Show edit time (3 points)**
    - If a task is updated, the time of update should be shown in addition to the creation time as shown below. If the task is not updated, only creation time should be shown (e.g., see the first task which was not updated).
        - **`Hint:`** you can add another column in your table to keep track the update date of the task. In your template, you can check whether the update time is greater than creation time, if so, show edit time in addition to the creation time.



Assignment Checklist:

The following is a checklist that you should refer to before submitting your assignment. This will prevent you making mistakes that you would otherwise lose points.

- Make sure that the web page you submit includes the html, head, title, body, and **doctype** tags.
- Make sure that your HTML file is well formed, meaning all the html, head, title and body tags are opened must be closed. For example, if you have `<body>` tag, you must have `</body>`.
    - You should check your HTML code using "Nu Html Checker"
        - **https://validator.w3.org/#validate_by_input**
- Try to avoid redundancy in your CSS as much as possible. You should not have too many redundant styles, if there are ways to use inheritance or special selectors to concisely define style rules. Points will be deducted if styles used for this page are too redundant.
- File names (e.g., html file, images, ccs file) should be all lower-case with no space to separate words, instead use `-` to separate words (e.g., recipe-styles.css).
- It is best practice to use relative file paths when describing location of your other files such as CSS, JS, images.
- Make sure you follow the best HTML practices, which you can find under Lecture-2 on Blackboard.
- Make sure you follow the best Django practices such as:
    - Use Django inheritance for your templates
    - Use static folder to store your CSS, JS, and images
    - Use Django forms whenever possible
    - Place your templates in a directory with the same name as your application

What to submit?

Submit **a single zip file** containing the following on Blackboard (please name this zip file with your first name and last name such as **John Smith HW4.zip**):

- Archive of the entire project
- For **each** of the three pages (index, update, delete), provide 2 screenshots (in PNG, JPG, JPEG or PDF format) showing your web page in your web browser for desktop size display, and small size display. (i.e., a total of 6 screenshots)
- Fill out the attached self-evaluation form and submit it along with the rest of the files
- For extra credit, if you have successfully deployed your site, provide the URL for your **PythonAnywhere** deployment site in the comment box on Blackboard.

If for some reason you are still not able to submit your files, email your files to me before the deadline. Please attach your files to this email.

Getting help

Start your assignment early. If you need help, send an email to me or make an online appointment with me during my office hours.