COS60010 Technology Enquiry Project

Semester 2, 2021

Deliverable 4
A Review on My Contribution to the To-do List Project

Lucas Qin

Individual Assignment

Course Code: 60010
Convenor: Dr Charlotte
Tutor: Dr Eureka
Student ID: 103527269
Student Name: Lucas Qin
Word Count: 4500
Date: 29/10/2021

Executive summary

In this project, my contribution to the project can be found as follow:

1. Extract requirements from project brief.

2. Decide back-end technology stack.

3. Design and develop the structure of database.

4. All functions of to do list including creation, edition and deletion of project and its tasks in the back-end.

5. All functions of the visualization page including doughnut chart, Gantt chart, display of team members and project calendar in the back-end.

6. Worked on making all HTML pages that related to to-do list and visualization pages (9 pages) and added HTTP requests by using Django template tags.

7. Applied Bootstrap to all HTML pages in to-do list and visualization.

8. Worked with another team member to finish the assign page in programming.

9. Combined functions of wiki page, resources page and user login & logout page with my to-do list and visualization page in testing phase and debugged.

10. Cooperate closely with the front-end team for applying CSS style to pages with Django template tags.

11. Attended all meetings from the semester and actively led the conversation for the meeting.

12. Provide support to team members in programming and design ideas.

13. United the team by acknowledging and appreciating others' works.

14. Prepared the presentation slides and rehearsal as a team.

Table of Contents

1. Introduction

In this project, our team has successfully built a Kanban-style to-do list, which contains 6 main functions including to-do list, project visualization, wiki page, team assign page, resources page and user login & register. The Fig.1 shows my coding part within the 6 main functions. I used Python as a programming language and Django as a framework. I have learned many practical skills in this project, technical parts including Python programming, Django framework, Bootstrap styling, Chart.js visualization, google chart and database design in Sqlite3. Other parts I have learned include how to deal with program issues, how to cooperate with other team members and how to work as an agile team.

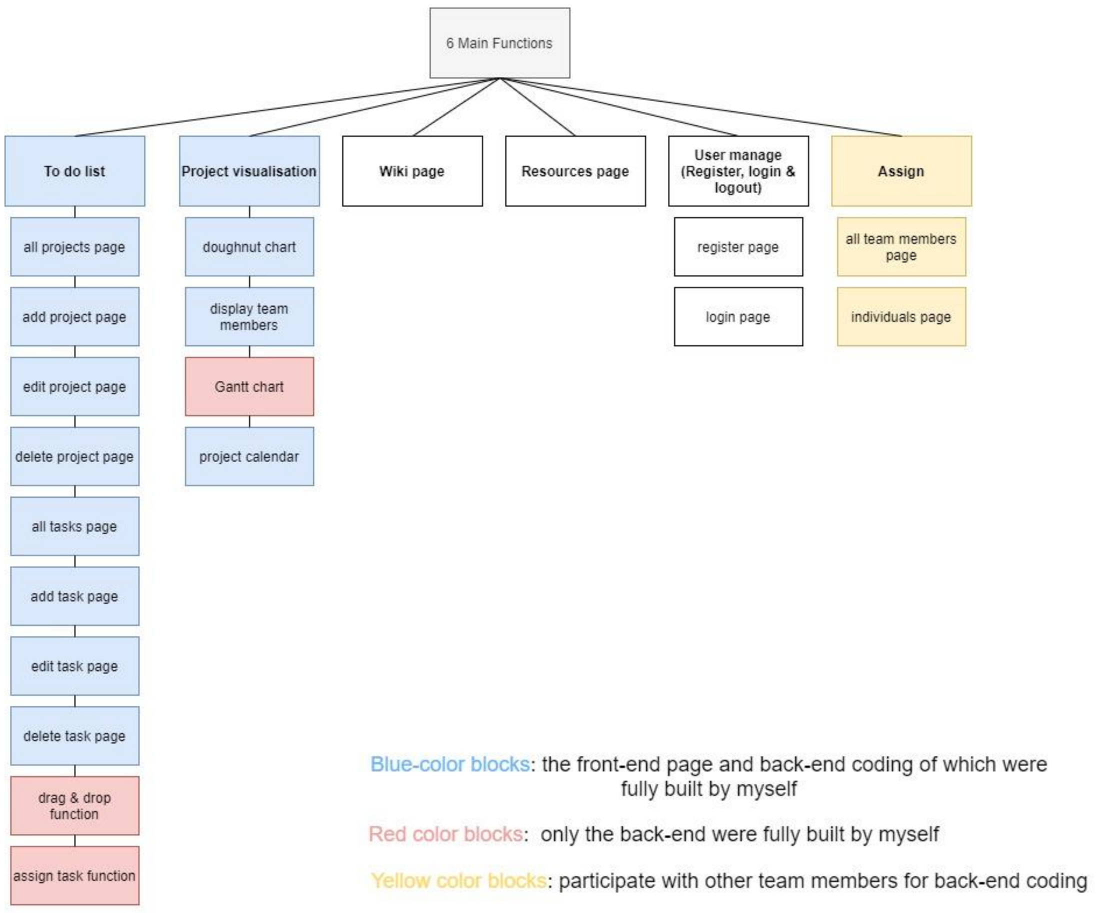


Fig.1 My coding part in the project

2. Production development

1. Extract requirements from project brief

I worked with my team to analyze the project brief and came up with user requirements. The requirements details have been presented in Deliverable 2.

2. Decide back-end technology stack

At the early stage of the project, I'm the only one that did coding in the backend, hence I realized it is important to pick a programming language and framework that suits our team. I did a lot of research during Deliverable 2. Based on Dr Eureka's opinion, I picked from Python, PHP and Ruby back then. The detailed reason has been presented in Deliverable 2. Here I summarized the reason why choose Python & Django as below:

1. Python is easy to learn.
2. Python has huge a developer community.
3. Django is popular for building web apps.
4. Django uses an MVC structure (Fig.2), it's easy to understand.

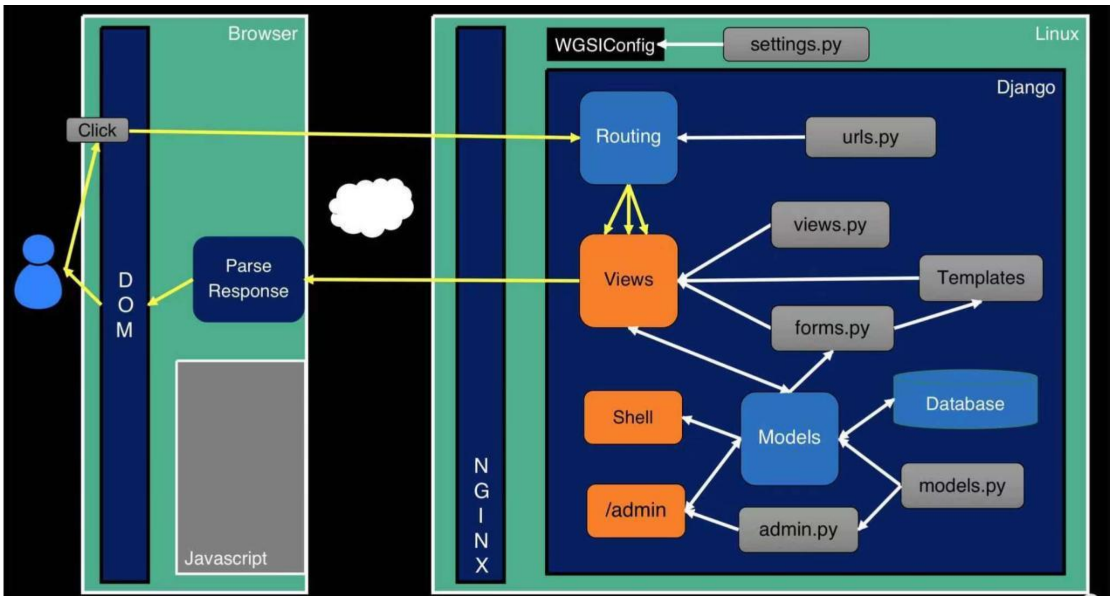


Fig.2 Structure of Django project[1]

3. Design and develop structure of database

The database structure was designed by me and Mona by collaboration.

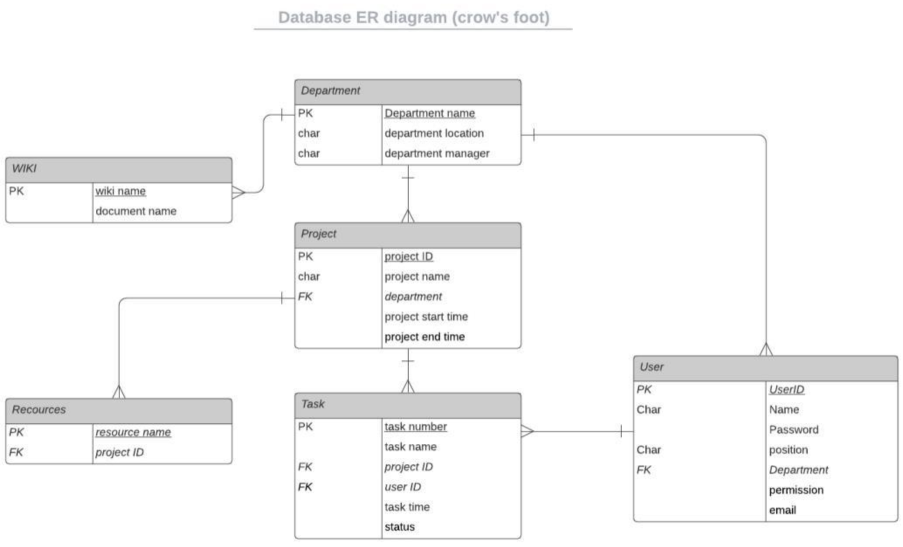


Fig.2.1 Entity-Relationship model

4. All functions of to do list

1. Introduction to to do function

In my design, the to-do list function consists of two parts, one is a project and another is to do list. The to do list is part of the project as shown on Fig. 3. When users enter into a project, they can see all the tasks within the project. The page for all the projects are shown as Fig.4, and the page for all the tasks with a project can be seen in Fig.5.

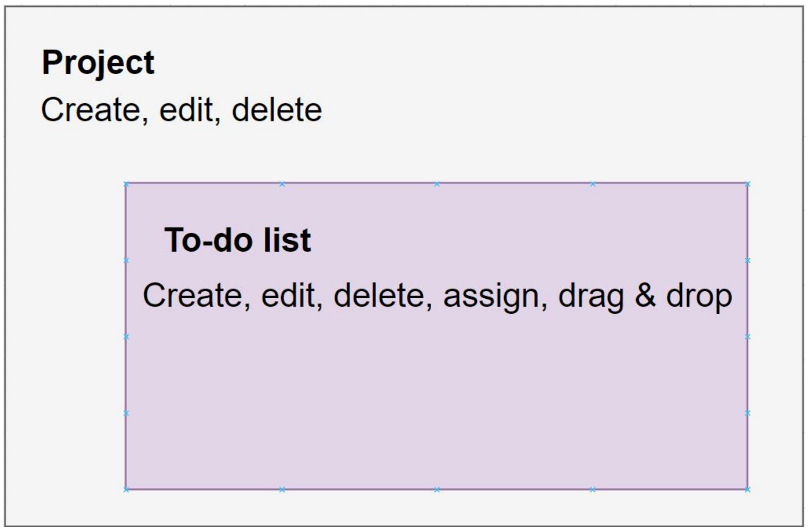
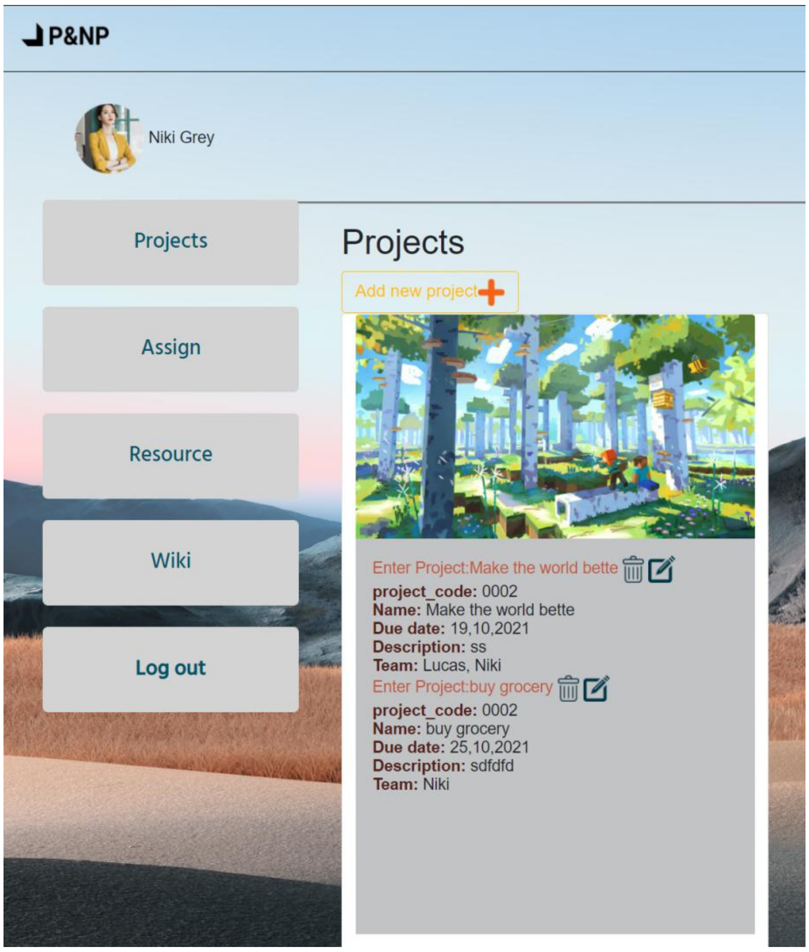


Fig. 3 Structure of to do list function

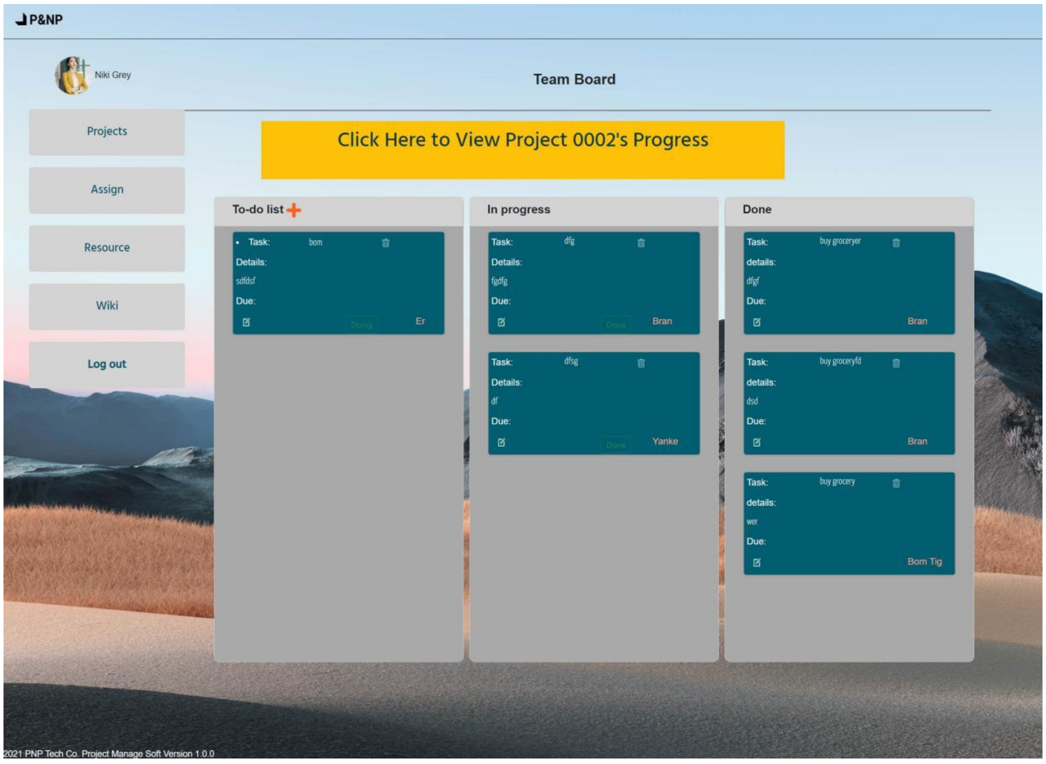


Projects

Enter Project:Make the world bette
project_code: 0002
Name: Make the world bette
Due date: 19,10,2021
Description: ss
Team: Lucas, Niki

Enter Project:buy grocery
project_code: 0002
Name: buy grocery
Due date: 25,10,2021
Description: sdfdfd
Team: Niki

Fig.4 Interface for all current projects



Team Board

To-do list

- Task: bom
Details: sdfdsf
Due:
Er

In progress

Task: dfg
Details: fgdfg
Due:
Bran

Task: dfsg
Details: df
Due:
Yanke

Done

Task: buy groceryer
details: dfgf
Due:
Bran

Task: buy groceryfd
details: dsd
Due:
Bran

Task: buy grocery
details: wer
Due:
Bom Tig

2021 PNP Tech Co. Project Manage Soft Version 1.0.0

Fig.5 Interface for all tasks

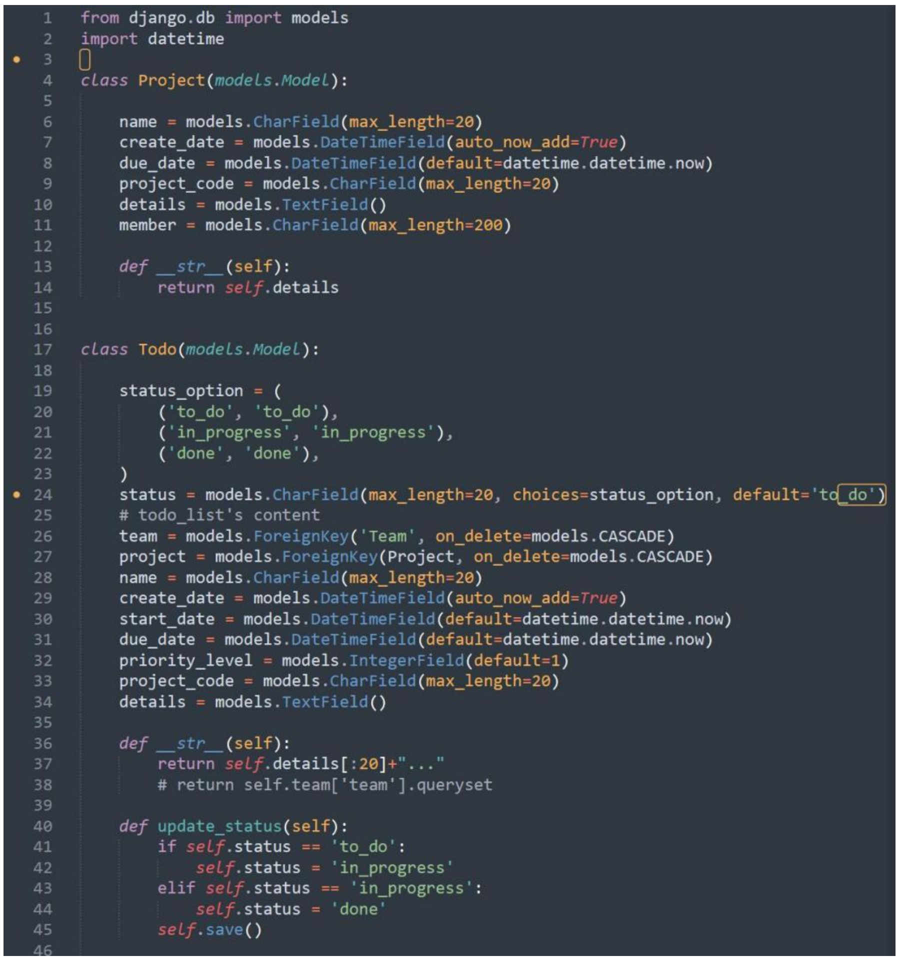
2. Explanation of my processes to create a new project and application

The detailed sequence for building to do list function can be described as:

1. Design database, make sure all the logic are clear and correct, here for project to tasks is one-to-many relation,besides task and project to team member are one-to-many relation as well.
2. Create a folder for placing to do list projects.
3. Using Command Prompt to navigate to the folder, then create a virtual environment, type in ‘*python -m venv 6_env*’, then activate ‘*6_env\Scripts\activate*’. The ‘6_env’ is the name of the virtual environment.
4. Install Django using ‘*pip install Django==3.2.7*’, version 3.2.7 was the latest.
5. Create a project within the folder using ‘*django-admin startproject todo_list .*’.
6. Using ‘*python manage.py migrate*’ to create a Sqlite3 database inside Django.
7. Run the localhost server by ‘*python manage.py runserver*’.
8. Open another Command Prompt, navigate to project’s folder, activate the virtual environment, create application using ‘*python manage.py startapp todo_lists*’, here in Django, normally different application controls over different function, this ‘todo_lists’ will be the main function of the 60010 project. In addition, the newly-created app needs to be registered in settings.py.
9. Define database in Sqlite3 file. The final version of the code can be seen in Fig.6, I will talk about how it ended up like this later, it has been through many amendments. The definitions for the database are stored in ‘models.py’ file, a class is a table, and within the class, I set datatype for every row, there are Charfield, Textfield, DateTimeField, etc.

For technical details, firstly, In Fig.6, the class ‘Todo’ is for all the tasks, I set 3 status for all the tasks, default is to_do, that means a newly created task will have a status to_do. Giving different status to tasks can help me manipulate them in HTML and moving them between different columns as shown on Fig.5. Secondly, one other thing to be noticed is the foreign key in Fig.6, by setting the project field as a foreign key, it inherits the value from another table and also inline with one-to-many relation in database design. The foreign key links it to a Team table, the team table was built together by me and Mona, so I’ll talk later. Thirdly, at the end of each class, I defined the data and data form the table return, these values can be used by algorithm in views.py directly. I’ll talk more about this in the algorithm part. All the models here need to be registered in admin.py by adding sentences like ‘*admin.site.register(project)*’.

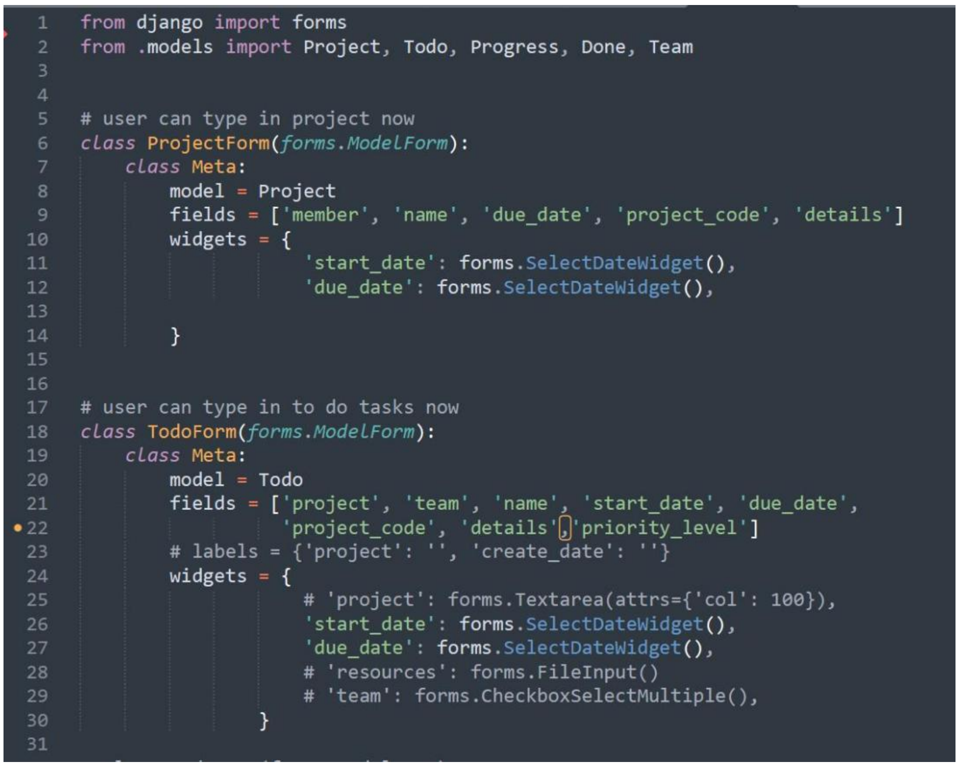
After defining the database in models.py, I use ‘python manage.py makemigrations’ to check the change that has been made to models.py, if there is no error, I use ‘python manage.py migrate’ to write models into the database. In fact, every time there’s a change to models in models.py, the processes will be needed again.



```python
from django.db import models
import datetime

class Project(models.Model):

    name = models.CharField(max_length=20)
    create_date = models.DateTimeField(auto_now_add=True)
    due_date = models.DateTimeField(default=datetime.datetime.now)
    project_code = models.CharField(max_length=20)
    details = models.TextField()
    member = models.CharField(max_length=200)

    def __str__(self):
        return self.details


class Todo(models.Model):

    status_option = (
        ('to_do', 'to_do'),
        ('in_progress', 'in_progress'),
        ('done', 'done'),
    )
    status = models.CharField(max_length=20, choices=status_option, default='to_do')
    # todo_list's content
    team = models.ForeignKey('Team', on_delete=models.CASCADE)
    project = models.ForeignKey(Project, on_delete=models.CASCADE)
    name = models.CharField(max_length=20)
    create_date = models.DateTimeField(auto_now_add=True)
    start_date = models.DateTimeField(default=datetime.datetime.now)
    due_date = models.DateTimeField(default=datetime.datetime.now)
    priority_level = models.IntegerField(default=1)
    project_code = models.CharField(max_length=20)
    details = models.TextField()

    def __str__(self):
        return self.details[:20]+"..."
        # return self.team['team'].queryset

    def update_status(self):
        if self.status == 'to_do':
            self.status = 'in_progress'
        elif self.status == 'in_progress':
            self.status = 'done'
        self.save()
```

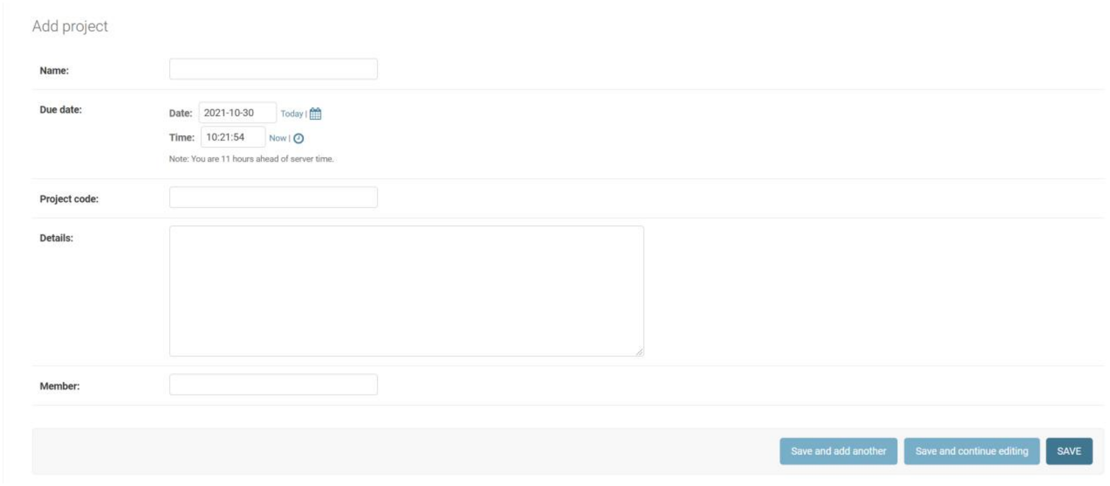
Fig.6 model.py, definition of project and task in database

10. Create a file called form.py, form.py is used to store Django's ModelForm unit, ModelForm is a method for creating a form to input data into a database. As shown in Fig.7. The ProjectForm inherits Django's forms.ModelForm, the meta inside the ModelForm tells Django to create a form base on which model in models.py, the fields allow me to choose which of the model's fields can be shown in the form for input. And the widget is to pick Django's widget type for each field, if not chosen, Django will create a default one based on the data type defined in the model. The form looks like Fig.8.



```python
from django import forms
from .models import Project, Todo, Progress, Done, Team


# user can type in project now
class ProjectForm(forms.ModelForm):
    class Meta:
        model = Project
        fields = ['member', 'name', 'due_date', 'project_code', 'details']
        widgets = {
                'start_date': forms.SelectDateWidget(),
                'due_date': forms.SelectDateWidget(),

        }


# user can type in to do tasks now
class TodoForm(forms.ModelForm):
    class Meta:
        model = Todo
        fields = ['project', 'team', 'name', 'start_date', 'due_date',
                 'project_code', 'details','priority_level']
        # labels = {'project': '', 'create_date': ''}
        widgets = {
                # 'project': forms.Textarea(attrs={'col': 100}),
                'start_date': forms.SelectDateWidget(),
                'due_date': forms.SelectDateWidget(),
                # 'resources': forms.FileInput()
                # 'team': forms.CheckboxSelectMultiple(),
            }

```

Fig.7 forms.py



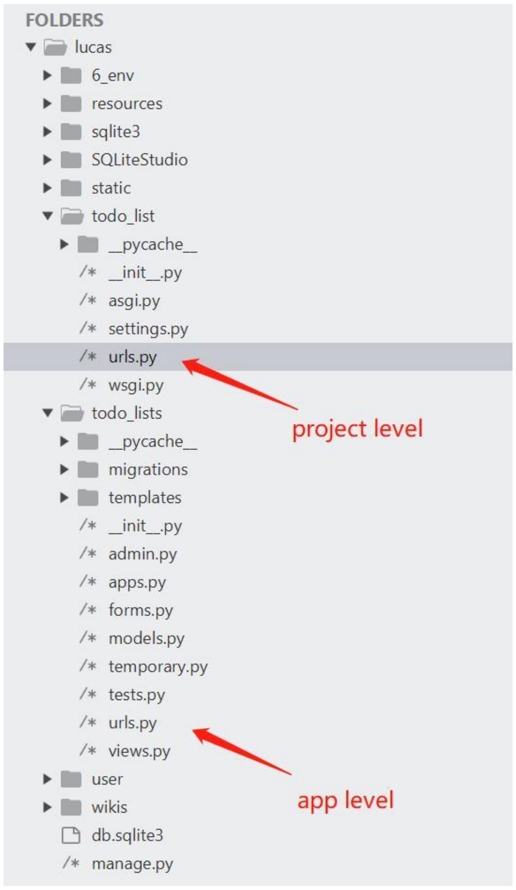
Add project

Name:	
Due date:	Date: 2021-10-30 Today | Time: 10:21:54 Now | Note: You are 11 hours ahead of server time.
Project code:	
Details:	
Member:	

Fig.8 created form in Django administration interface

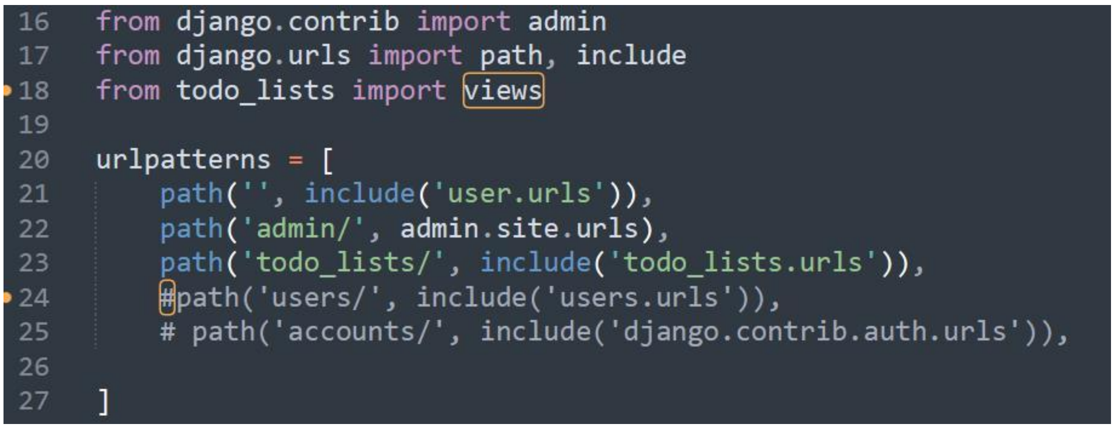
11. Create a URL path for the different pages in Localhost.

There are two steps. Firstly, the structure of the file can be seen in Fig.8, open urls.py in project-level folder, and add a root url for entering the project and then create a urls.py in app level, and add urls for different functions in Django. The project level and app level URL can be seen in Fig.9 and Fig.10. The third one is the URL for to do list application, it tells Django to move to the app I created at the beginning called ‘todo_lists’ to find for its sub-path, and in todo_lists, I have created a URL files for Django, inside the file as shown in Fig.10, I have defined all the necessary path for different function, take line 15 as an example, the path is a Django method for defining URL, it is until after Django version 2.0 that Django replaced 'URL' with 'path'. Inside the path, I inherited 'projects' from line 13, and added a '*<int:project_id>/*', it tells Django that the name of the URL after '*projects/*' should be an integer number, and it match a parameter called '*project_id*', but to look for the '*project_id*', I tell Django to go to views.py and look for a function called 'project', the function will be created in next step, which is the most challenged and time-consuming part in the whole project.



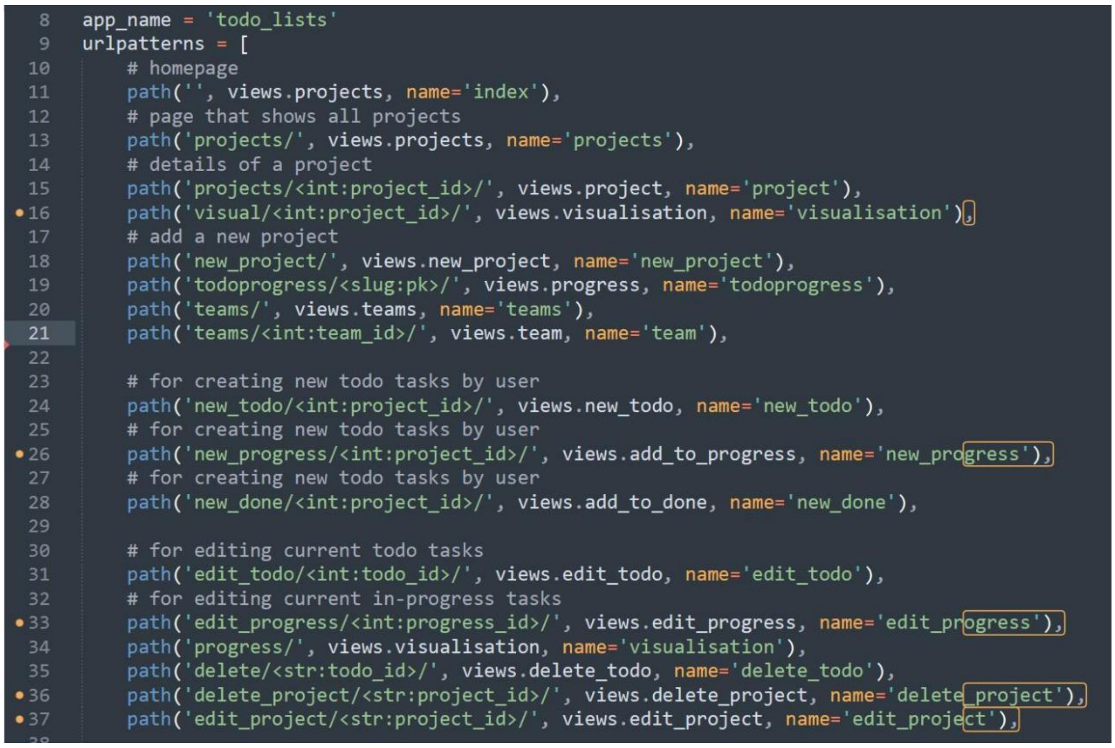
```
FOLDERS
▼ lucas
  ▶ 6_env
  ▶ resources
  ▶ sqlite3
  ▶ SQLiteStudio
  ▶ static
  ▼ todo_list
    ▶ __pycache__
      __init__.py
      asgi.py
      settings.py
      urls.py
      wsgi.py
  ▼ todo_lists
    ▶ __pycache__
    ▶ migrations
    ▶ templates
      __init__.py
      admin.py
      apps.py
      forms.py
      models.py
      temporary.py
      tests.py
      urls.py
      views.py
  ▶ user
  ▶ wikis
    db.sqlite3
    manage.py
```

Fig.8 Structure of files



```
from django.contrib import admin
from django.urls import path, include
from todo_lists import views

urlpatterns = [
    path('', include('user.urls')),
    path('admin/', admin.site.urls),
    path('todo_lists/', include('todo_lists.urls')),
    #path('users/', include('users.urls')),
    # path('accounts/', include('django.contrib.auth.urls')),

]
```

Fig.9 project-level URL



```python
app_name = 'todo_lists'
urlpatterns = [
    # homepage
    path('', views.projects, name='index'),
    # page that shows all projects
    path('projects/', views.projects, name='projects'),
    # details of a project
    path('projects/<int:project_id>/', views.project, name='project'),
    path('visual/<int:project_id>/', views.visualisation, name='visualisation'),
    # add a new project
    path('new_project/', views.new_project, name='new_project'),
    path('todoprogress/<slug:pk>/', views.progress, name='todoprogress'),
    path('teams/', views.teams, name='teams'),
    path('teams/<int:team_id>/', views.team, name='team'),

    # for creating new todo tasks by user
    path('new_todo/<int:project_id>/', views.new_todo, name='new_todo'),
    # for creating new todo tasks by user
    path('new_progress/<int:project_id>/', views.add_to_progress, name='new_progress'),
    # for creating new todo tasks by user
    path('new_done/<int:project_id>/', views.add_to_done, name='new_done'),

    # for editing current todo tasks
    path('edit_todo/<int:todo_id>/', views.edit_todo, name='edit_todo'),
    # for editing current in-progress tasks
    path('edit_progress/<int:progress_id>/', views.edit_progress, name='edit_progress'),
    path('progress/', views.visualisation, name='visualisation'),
    path('delete/<str:todo_id>/', views.delete_todo, name='delete_todo'),
    path('delete_project/<str:project_id>/', views.delete_project, name='delete_project'),
    path('edit_project/<str:project_id>/', views.edit_project, name='edit_project'),
```

Fig.10 app-level URL

12. Django has already created an empty views.py file in the application folder 'todo_lists', the file contains all the core methods for retrieving data from the database.

Take the first function in Fig. 11 as an example, I defined a '*def projects()*' to display all the current projects, the '*def project()*' accept 'request' method to call. And inside the function, I defined a variable called '*projects*' to store all objects in a model called '*Project*', which I defined in models.py. Then I pass the value of the variable to a dictionary called 'context', this is how Django works, the output must be a dictionary. Finally, I output the request, context and relevant HTML page I created for the function using Django's render method, which I used '*from django.shortcuts import render*' sentence on the top of views.py to import the method. The step I created the HTML pages explains why most of the HTML were created and added styles by me, because the HTML page specifically addressed my need to store a function, and it requires knowledge in Django's template tags to make one.

These three functions are a small part of all functions, I will explain in a later section how the code ends up like this, it has been through many iterations to be workable in the project.

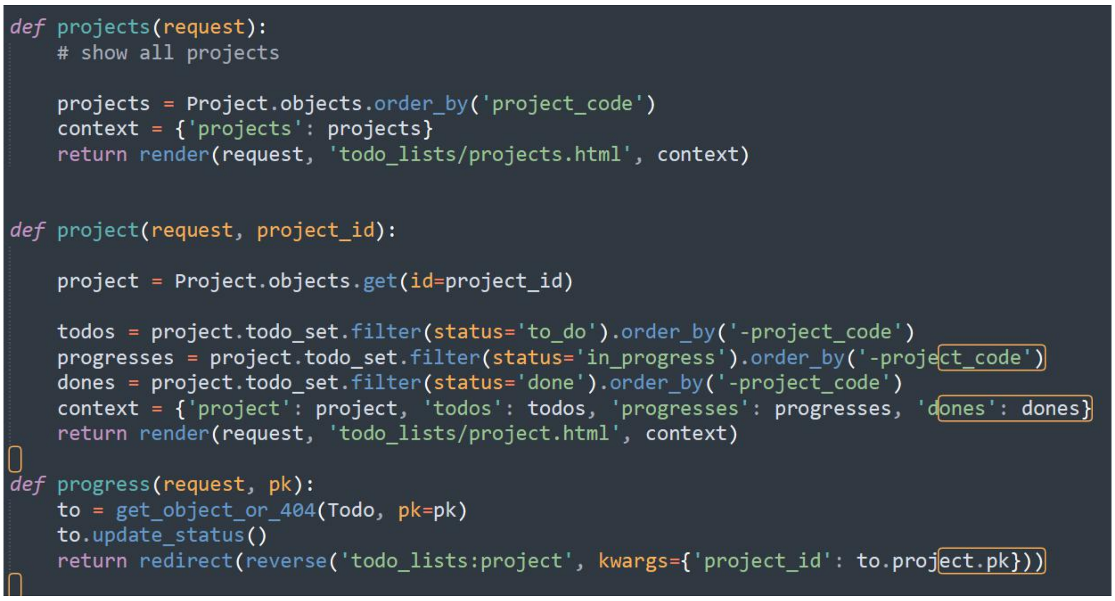

```python
def projects(request):
    # show all projects

    projects = Project.objects.order_by('project_code')
    context = {'projects': projects}
    return render(request, 'todo_lists/projects.html', context)


def project(request, project_id):

    project = Project.objects.get(id=project_id)

    todos = project.todo_set.filter(status='to_do').order_by('-project_code')
    progresses = project.todo_set.filter(status='in_progress').order_by('-project_code')
    dones = project.todo_set.filter(status='done').order_by('-project_code')
    context = {'project': project, 'todos': todos, 'progresses': progresses, 'dones': dones}
    return render(request, 'todo_lists/project.html', context)

def progress(request, pk):
    to = get_object_or_404(Todo, pk=pk)
    to.update_status()
    return redirect(reverse('todo_lists:project', kwargs={'project_id': to.project.pk}))
```

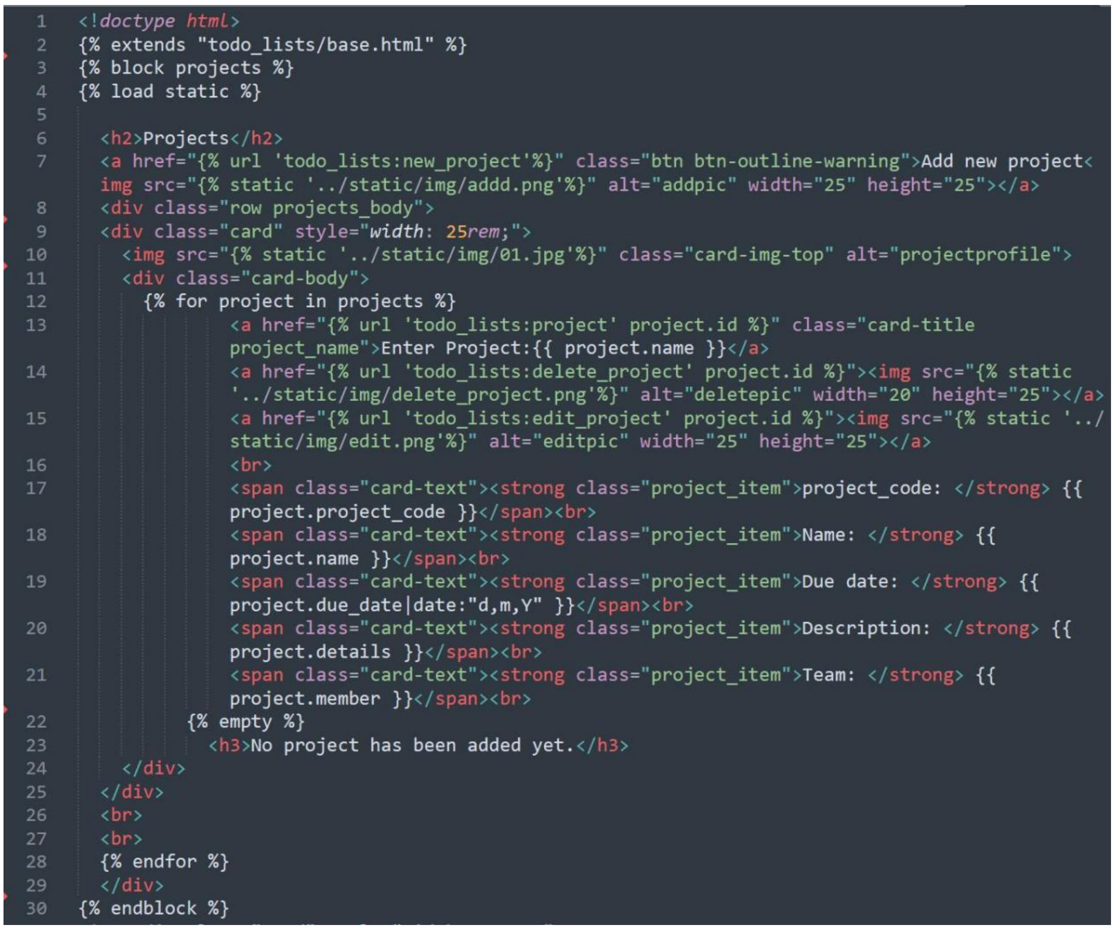
Fig.11 definition of functions of displaying all projects, each project and changing status of a task

13. In views.py, I used the render method to tell Django to return a HTML page, hence this step is for creating the HTML page. One particular HTML page can be seen as Fig.12. Most of the HTML pages in this project were created by server side as it addresses specific needs for server side output. In a real industrial environment, frontend and backend are connected through different API, whenever frontend sends a request, the request goes to API, hence HTTP request must be in a format the API can accept. Then the API will pass the request to the server, mostly in JSON format, and the server checks the database and returns a message to the API. In our project, we did not introduce an API between frontend and backend, hence the only way was that backend team use template language to create HTTP requests based on definitions in views.py.

Take Fig.12 as an example, first of all, I made all the pages created by myself as a block for better reusability and management, the '*{% entends "todo_lists/base.html" %}*' tell Django to put all the codes behind this line to base.html, which only contains common elements like header, style, fonts and JavaScript links, sidebar and footer . Take a look at base.html in Fig. 13, Django will match the '*{% block projects %}{% endblock %}*' in Fig.12 with Fig.13, and place the block in wherever the tag appear in the base.html.

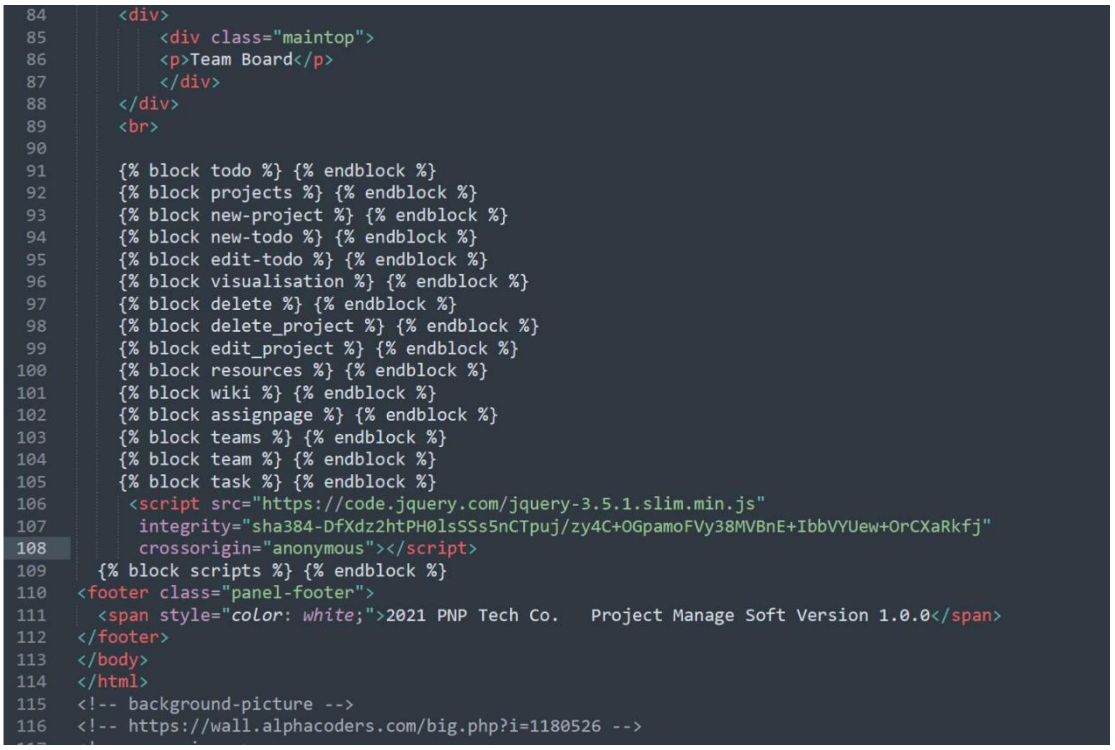
Static files include CSS, JavaScripts and images, the '*{% load static %}*' tag tells Django to load static files from the static folder. The load static method also needs to be defined in settings.py for using.

The '*{% for project in projects %}{% endfor %}*' tells Django to look for everything that is in the context I defined in views.py and for every item in the context execute the codes inside the for loop for one time. The context is a dictionary that includes names and values of Project model's object as shown in Fig.11. The double curly bracket dynamically displays context's value. The final effect can be found on Fig.4, the for loop in HTML iterated twice to display 2 current projects.



```html
<!doctype html>
{% extends "todo_lists/base.html" %}
{% block projects %}
{% load static %}

  <h2>Projects</h2>
  <a href="{% url 'todo_lists:new_project'%}" class="btn btn-outline-warning">Add new project<img src="{% static '../static/img/addd.png'%}" alt="addpic" width="25" height="25"></a>
  <div class="row projects_body">
  <div class="card" style="width: 25rem;">
    <img src="{% static '../static/img/01.jpg'%}" class="card-img-top" alt="projectprofile">
    <div class="card-body">
      {% for project in projects %}
            <a href="{% url 'todo_lists:project' project.id %}" class="card-title project_name">Enter Project:{{ project.name }}</a>
            <a href="{% url 'todo_lists:delete_project' project.id %}"><img src="{% static '../static/img/delete_project.png'%}" alt="deletepic" width="20" height="25"></a>
            <a href="{% url 'todo_lists:edit_project' project.id %}"><img src="{% static '../static/img/edit.png'%}" alt="editpic" width="25" height="25"></a>
            <br>
            <span class="card-text"><strong class="project_item">project_code: </strong> {{ project.project_code }}</span><br>
            <span class="card-text"><strong class="project_item">Name: </strong> {{ project.name }}</span><br>
            <span class="card-text"><strong class="project_item">Due date: </strong> {{ project.due_date|date:"d,m,Y" }}</span><br>
            <span class="card-text"><strong class="project_item">Description: </strong> {{ project.details }}</span><br>
            <span class="card-text"><strong class="project_item">Team: </strong> {{ project.member }}</span><br>
        {% empty %}
          <h3>No project has been added yet.</h3>
    </div>
  </div>
  <br>
  <br>
  {% endfor %}
  </div>
{% endblock %}
```

Fig.12 HTML page for displaying all projects



```html
    <div>
        <div class="maintop">
        <p>Team Board</p>
        </div>
    </div>
    <br>

    {% block todo %} {% endblock %}
    {% block projects %} {% endblock %}
    {% block new-project %} {% endblock %}
    {% block new-todo %} {% endblock %}
    {% block edit-todo %} {% endblock %}
    {% block visualisation %} {% endblock %}
    {% block delete %} {% endblock %}
    {% block delete_project %} {% endblock %}
    {% block edit_project %} {% endblock %}
    {% block resources %} {% endblock %}
    {% block wiki %} {% endblock %}
    {% block assignpage %} {% endblock %}
    {% block teams %} {% endblock %}
    {% block team %} {% endblock %}
    {% block task %} {% endblock %}
     <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
      integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
      crossorigin="anonymous"></script>
  {% block scripts %} {% endblock %}
<footer class="panel-footer">
  <span style="color: white;">2021 PNP Tech Co.   Project Manage Soft Version 1.0.0</span>
</footer>
</body>
</html>
<!-- background-picture -->
<!-- https://wall.alphacoders.com/big.php?i=1180526 -->
```

Fig.13 base.html

14. The above steps clearly show the method on how I started a project and an application in Django. For adding more functions like add new task or edit task, the steps will normally be repeated between step 11-13, and the database also will be adjusted constantly in terms of data type it returns and columns that are missing.

3. Explanations to codes' iteration

When I first start this semester, I have zero knowledge of coding, and this is my first semester. Hence I have learned a lot of knowledge of Python and Django, and went into many errors to go this far. At the very beginning, all the codes I have learned is from a book called "Python Crash Course" written by Eric Matthes, in this book, chapter 1 to 11 are about basic knowledge of Python, and then in chapter 18-20, there is small web application project, the project was about making a study note where there are title and details of the note. I read the book thoroughly at least 3 times, and for chapters 18-20, I must have read over 100 times.

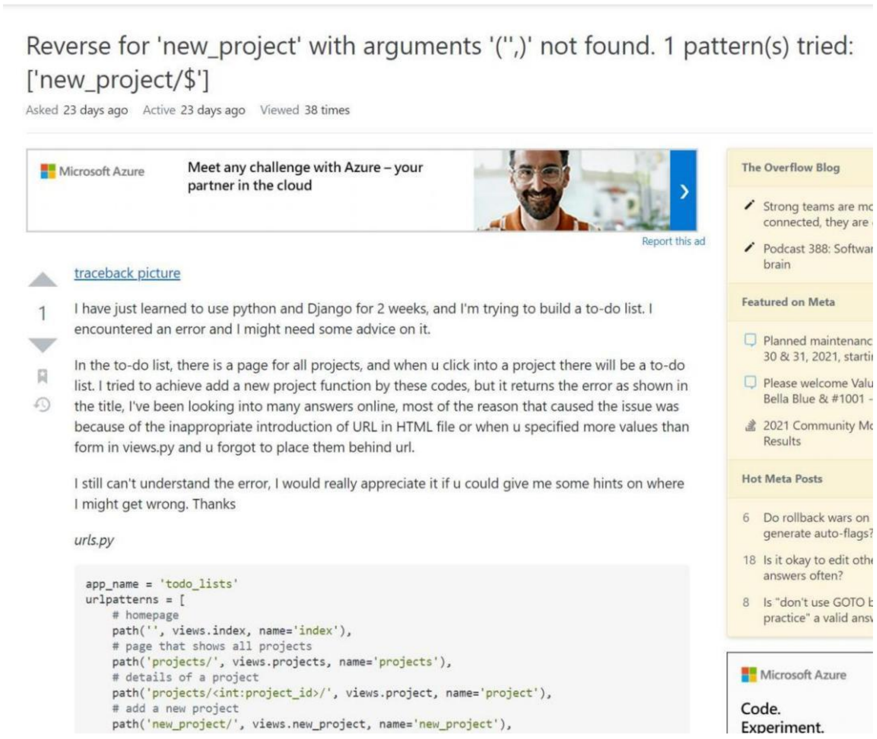
At first, I managed to get to know what those codes meant, until I could clearly understand every row of code in the book. And then I tried to follow the books' instructions to do the project, so that I can get more familiar with coding. But many errors showed up, as a beginner, it was so easy to cause errors by missing colon or something. Sometimes I have to go to stack overflow to ask a question which seems easy to me now as shown in Fig. 14. And gradually, I encountered repeated errors, and I can solve them by myself. However, one another issue popped up, the codes in the book were very old versions of code, it did not support the Django version I was using, so it seems like I can continue to follow the book's instructions to practice. I searched a lot of information on different versions of code, luckily, I found the author Eric's blog, he kept updating code based on Django's update for the book, with this latest code, I followed the book to do the whole project for 4 times.

When I felt confident enough to write some code for my to-do list project, I was stuck. I found a crude fact that understanding other's code is one thing, and writing my own code is another. As shown on Fig.15, at the very early stage, I imitated the algorithm from the book, and each task follow the similar pattern in coding, that means to a task that is 'to do', 'in progress' and 'done', they can only be created separately, there was no relation among them. The only good news for this phase was I can have title, details, due date and other information for my task, both input and output. The next step was to figure out how to connect tasks of 'to do', 'in progress' and 'done'. The book doesn't demonstrate examples like this, thus I have to find the answer myself.

I went to Stack Overflow and asked people about their opinion as shown in Fig.16, they pointed me to a whole new method I did not know, that is adding different status to a task rather than transit data among 3 tables. Because if I kept 3 table for storing 'to do', 'in progress' and 'done', I would need to send one request to delete the task in 'to do' and make a copy in ‘in progress’ simultaneously, I can’t find a way to do that in one ‘*def*’ method, normally create and delete are two ‘*def*’ in views.py.

A model with status looks like Fig.17, and based on the model, I rewritten the algorithm in views.py as shown on Fig. 18. In Fig.18, I specified the method to identify a specific task by its id, and then I created a new ModelForm called ‘*dragTodoForm*’ in form.py for inserting data into the database. I was planning to place the form in HTML to send the request to add_to_progress or add_to_done in Fig.18. When the form is valid, I use the update method to change the task status, and save the data in the form to the database. Once the status is changed, I can extract values in Todo models based on their status and display the task in different columns in HTML. Unfortunately, this method did not work.

I had to dig more about how to change a task status. Again, I browsed all the articles I could find on Stack Overflow on ‘how to change the status of a field in Django?’. Finally, after countless searches and experiments, I came up with the correct code to deal with the relation of ‘to do’, ‘in progress’ and ‘done’ as shown in Fig.8 and Fig.11, plus other adjustments in URL, settings and HTML’s http request.



Reverse for 'new_project' with arguments '('',)' not found. 1 pattern(s) tried: ['new_project/$']

Asked 23 days ago Active 23 days ago Viewed 38 times

Microsoft Azure Meet any challenge with Azure – your partner in the cloud

Report this ad

1

traceback picture

I have just learned to use python and Django for 2 weeks, and I'm trying to build a to-do list. I encountered an error and I might need some advice on it.

In the to-do list, there is a page for all projects, and when u click into a project there will be a to-do list. I tried to achieve add a new project function by these codes, but it returns the error as shown in the title, I've been looking into many answers online, most of the reason that caused the issue was because of the inappropriate introduction of URL in HTML file or when u specified more values than form in views.py and u forgot to place them behind url.

I still can't understand the error, I would really appreciate it if u could give me some hints on where I might get wrong. Thanks

urls.py

```
app_name = 'todo_lists'
urlpatterns = [
    # homepage
    path('', views.index, name='index'),
    # page that shows all projects
    path('projects/', views.projects, name='projects'),
    # details of a project
    path('projects/<int:project_id>/', views.project, name='project'),
    # add a new project
    path('new_project/', views.new_project, name='new_project'),
```

The Overflow Blog

Strong teams are mc connected, they are

Podcast 388: Softwa brain

Featured on Meta

Planned maintenanc 30 & 31, 2021, starti

Please welcome Valu Bella Blue & #1001 -

2021 Community Mc Results

Hot Meta Posts

6 Do rollback wars on generate auto-flags?

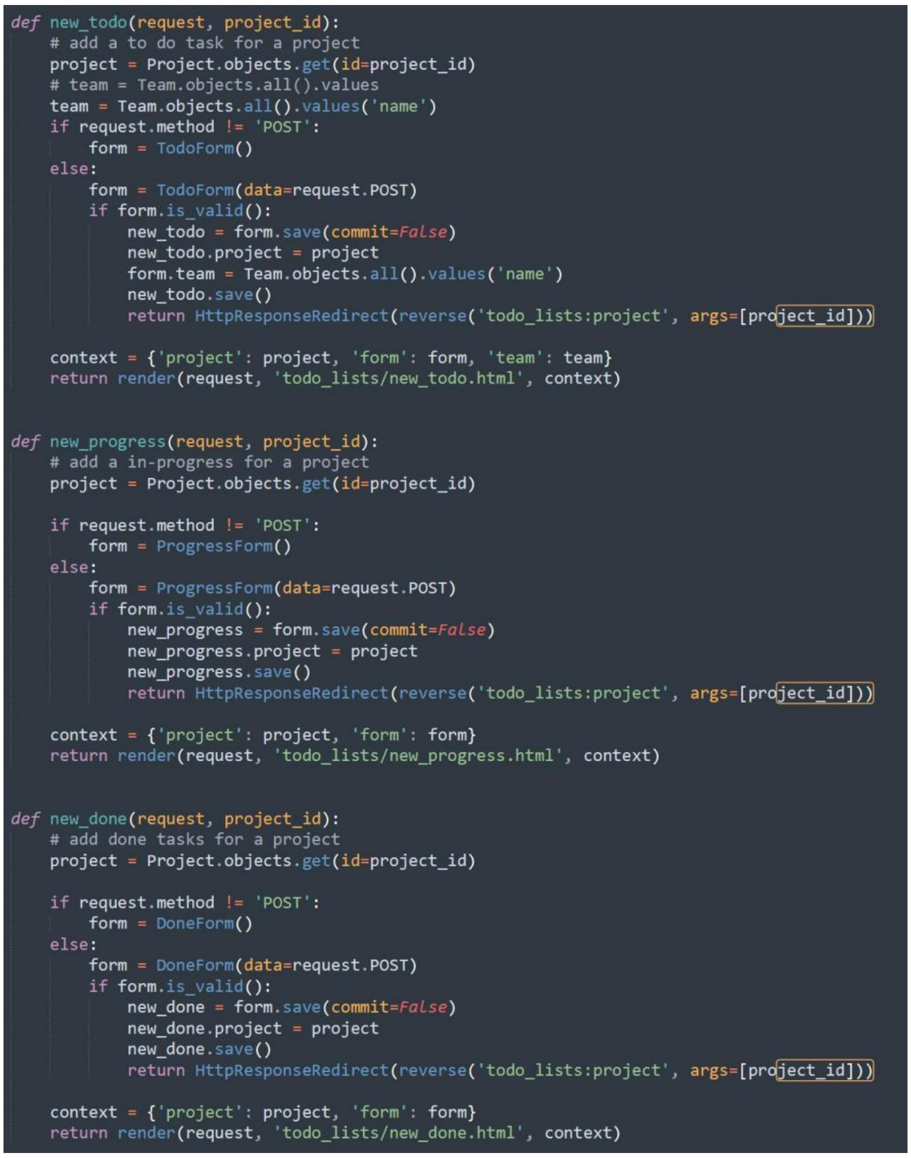
18 Is it okay to edit othe answers often?

8 Is "don't use GOTO b practice" a valid ansv

Microsoft Azure

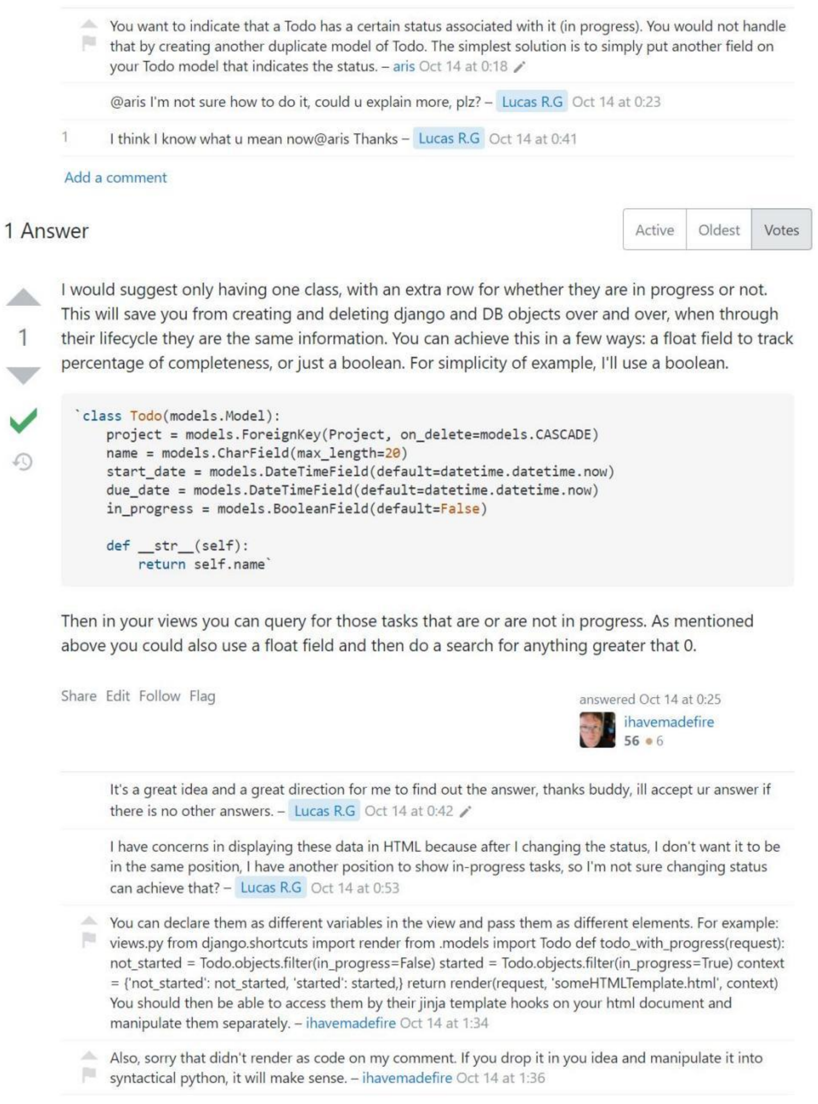
Code. Experiment.

Fig.14 I asked question in Stack Overflow during learning Django



```python
def new_todo(request, project_id):
    # add a to do task for a project
    project = Project.objects.get(id=project_id)
    # team = Team.objects.all().values
    team = Team.objects.all().values('name')
    if request.method != 'POST':
        form = TodoForm()
    else:
        form = TodoForm(data=request.POST)
        if form.is_valid():
            new_todo = form.save(commit=False)
            new_todo.project = project
            form.team = Team.objects.all().values('name')
            new_todo.save()
            return HttpResponseRedirect(reverse('todo_lists:project', args=[project_id]))

    context = {'project': project, 'form': form, 'team': team}
    return render(request, 'todo_lists/new_todo.html', context)


def new_progress(request, project_id):
    # add a in-progress for a project
    project = Project.objects.get(id=project_id)

    if request.method != 'POST':
        form = ProgressForm()
    else:
        form = ProgressForm(data=request.POST)
        if form.is_valid():
            new_progress = form.save(commit=False)
            new_progress.project = project
            new_progress.save()
            return HttpResponseRedirect(reverse('todo_lists:project', args=[project_id]))

    context = {'project': project, 'form': form}
    return render(request, 'todo_lists/new_progress.html', context)


def new_done(request, project_id):
    # add done tasks for a project
    project = Project.objects.get(id=project_id)

    if request.method != 'POST':
        form = DoneForm()
    else:
        form = DoneForm(data=request.POST)
        if form.is_valid():
            new_done = form.save(commit=False)
            new_done.project = project
            new_done.save()
            return HttpResponseRedirect(reverse('todo_lists:project', args=[project_id]))

    context = {'project': project, 'form': form}
    return render(request, 'todo_lists/new_done.html', context)
```

Fig. 15 A to-do task can only be created separately at early stage



You want to indicate that a Todo has a certain status associated with it (in progress). You would not handle that by creating another duplicate model of Todo. The simplest solution is to simply put another field on your Todo model that indicates the status. – aris Oct 14 at 0:18

@aris I'm not sure how to do it, could u explain more, plz? – Lucas R.G Oct 14 at 0:23

1 I think I know what u mean now@aris Thanks – Lucas R.G Oct 14 at 0:41

Add a comment

1 Answer

Active Oldest Votes

1

I would suggest only having one class, with an extra row for whether they are in progress or not. This will save you from creating and deleting django and DB objects over and over, when through their lifecycle they are the same information. You can achieve this in a few ways: a float field to track percentage of completeness, or just a boolean. For simplicity of example, I'll use a boolean.

```
`class Todo(models.Model):
    project = models.ForeignKey(Project, on_delete=models.CASCADE)
    name = models.CharField(max_length=20)
    start_date = models.DateTimeField(default=datetime.datetime.now)
    due_date = models.DateTimeField(default=datetime.datetime.now)
    in_progress = models.BooleanField(default=False)

    def __str__(self):
        return self.name`
```

Then in your views you can query for those tasks that are or are not in progress. As mentioned above you could also use a float field and then do a search for anything greater that 0.

Share Edit Follow Flag

answered Oct 14 at 0:25
ihavemadefire
56 ● 6

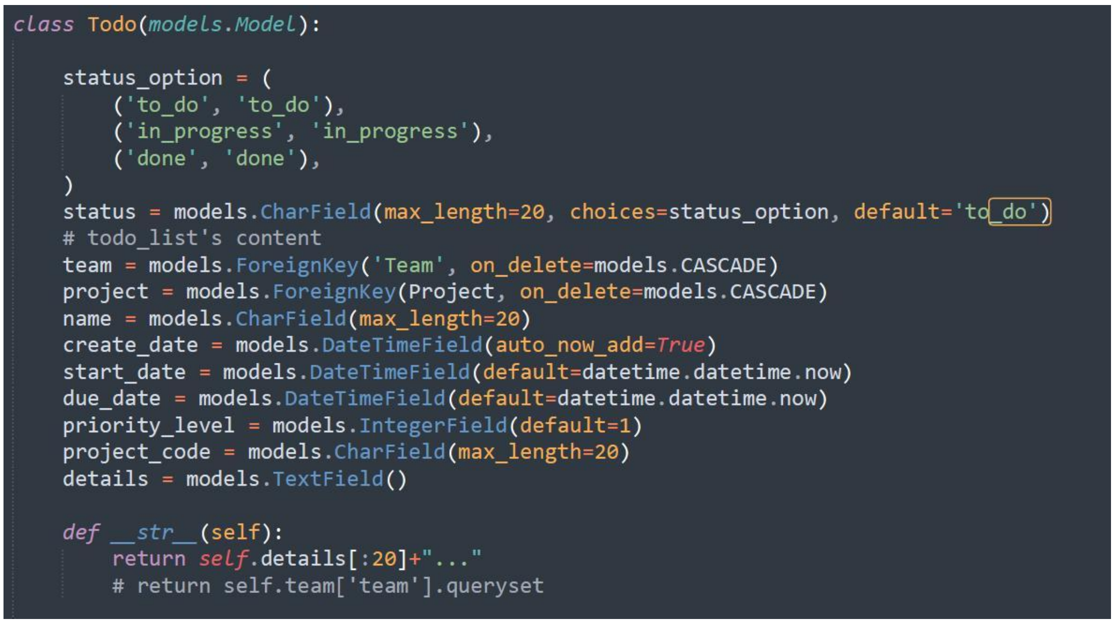
It's a great idea and a great direction for me to find out the answer, thanks buddy, ill accept ur answer if there is no other answers. – Lucas R.G Oct 14 at 0:42

I have concerns in displaying these data in HTML because after I changing the status, I don't want it to be in the same position, I have another position to show in-progress tasks, so I'm not sure changing status can achieve that? – Lucas R.G Oct 14 at 0:53

You can declare them as different variables in the view and pass them as different elements. For example: views.py from django.shortcuts import render from .models import Todo def todo_with_progress(request): not_started = Todo.objects.filter(in_progress=False) started = Todo.objects.filter(in_progress=True) context = {'not_started': not_started, 'started': started,} return render(request, 'someHTMLTemplate.html', context) You should then be able to access them by their jinja template hooks on your html document and manipulate them separately. – ihavemadefire Oct 14 at 1:34

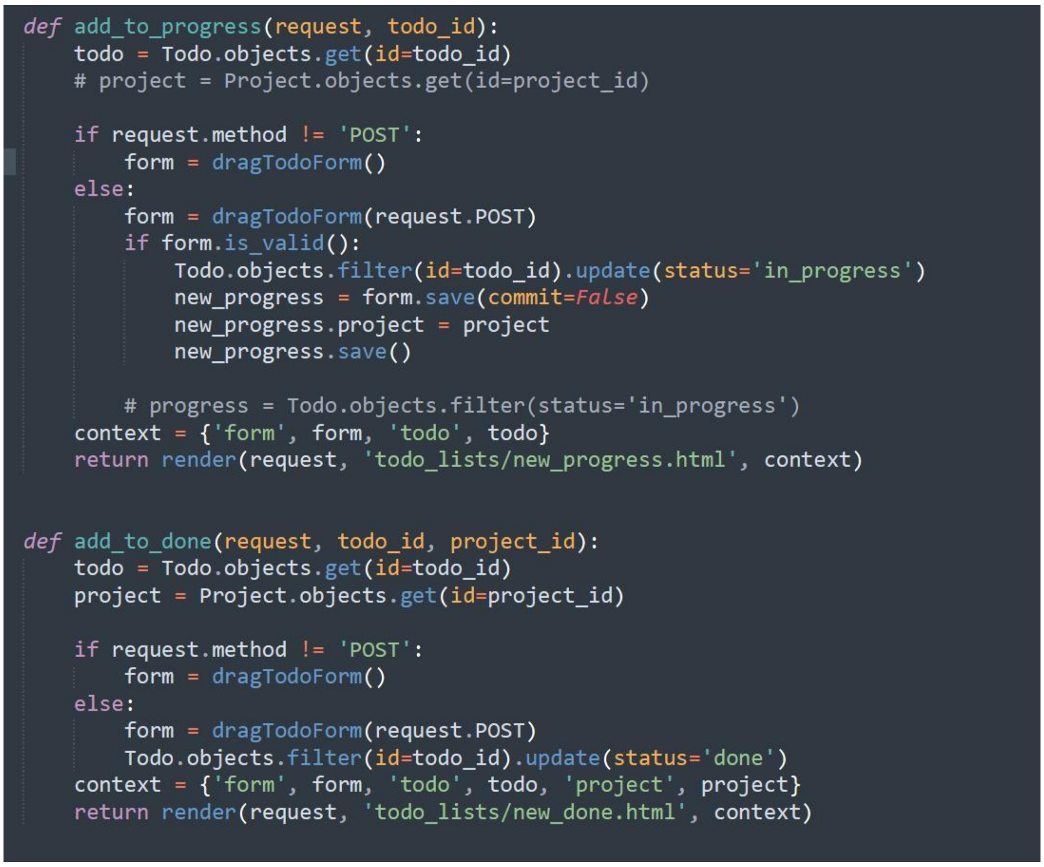
Also, sorry that didn't render as code on my comment. If you drop it in you idea and manipulate it into syntactical python, it will make sense. – ihavemadefire Oct 14 at 1:36

Fig. 16 I asked people their opinion in Stack Overflow



```
class Todo(models.Model):

    status_option = (
        ('to_do', 'to_do'),
        ('in_progress', 'in_progress'),
        ('done', 'done'),
    )
    status = models.CharField(max_length=20, choices=status_option, default='to_do')
    # todo_list's content
    team = models.ForeignKey('Team', on_delete=models.CASCADE)
    project = models.ForeignKey(Project, on_delete=models.CASCADE)
    name = models.CharField(max_length=20)
    create_date = models.DateTimeField(auto_now_add=True)
    start_date = models.DateTimeField(default=datetime.datetime.now)
    due_date = models.DateTimeField(default=datetime.datetime.now)
    priority_level = models.IntegerField(default=1)
    project_code = models.CharField(max_length=20)
    details = models.TextField()

    def __str__(self):
        return self.details[:20]+"..."
        # return self.team['team'].queryset
```

Fig. 17 I came up with model based on people's suggestion on Stack Overflow



```python
def add_to_progress(request, todo_id):
    todo = Todo.objects.get(id=todo_id)
    # project = Project.objects.get(id=project_id)

    if request.method != 'POST':
        form = dragTodoForm()
    else:
        form = dragTodoForm(request.POST)
        if form.is_valid():
            Todo.objects.filter(id=todo_id).update(status='in_progress')
            new_progress = form.save(commit=False)
            new_progress.project = project
            new_progress.save()

        # progress = Todo.objects.filter(status='in_progress')
    context = {'form', form, 'todo', todo}
    return render(request, 'todo_lists/new_progress.html', context)


def add_to_done(request, todo_id, project_id):
    todo = Todo.objects.get(id=todo_id)
    project = Project.objects.get(id=project_id)

    if request.method != 'POST':
        form = dragTodoForm()
    else:
        form = dragTodoForm(request.POST)
        Todo.objects.filter(id=todo_id).update(status='done')
    context = {'form', form, 'todo', todo, 'project', project}
    return render(request, 'todo_lists/new_done.html', context)
```

Fig.18 The algorithm that tried to change the status for task

5. All functions of visualization page

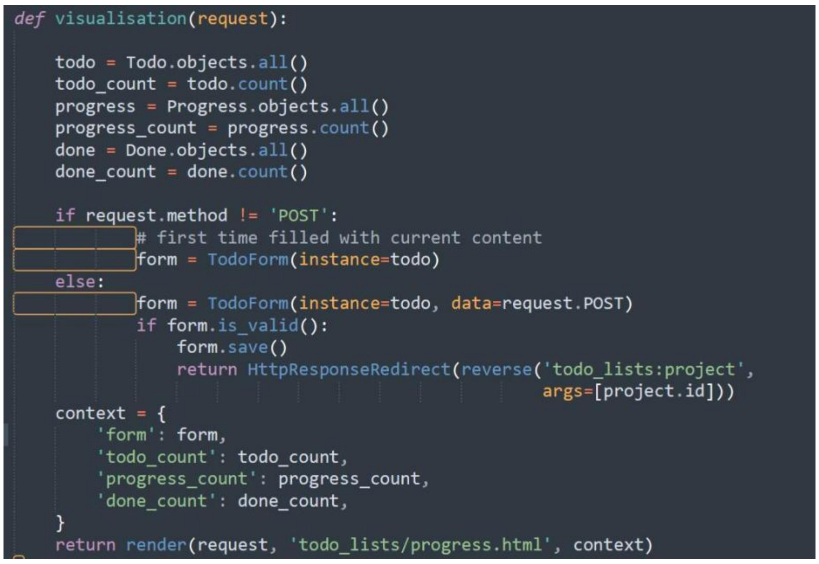
The visualization page has 4 parts in terms of contents: doughnut chart, team member display, Gantt chart and calendar. On the one hand, all the backend algorithms of the page have been done by myself, on the other hand, I have finished all the JavaScript parts except Gantt chart, and I added Bootstrap grid to the page. The doughnut chart is using Chart.js, Gantt chart and project Calendar are using Google chart, they are libraries for visualization. I have tried Plotly, Matplotlib and some other libraries, these libraries are Python-based libraries, they need to be defined chart in Django and integrated with my current '*def*' in views.py, I found them not particularly easier to understand than Chart.js and Google chart, considering the time restraint of the project, I chose Chart.js and Google chart.

For using a library, most of the code in JavaScript does not need to be changed, I just need to focus on where the data defines the chart, what format it requires and how to call it properly if it's a JavaScript '*var*' type.

The Most challenge part was the algorithm, for doughnut chart, it displays the total number of different type of tasks, Fig. 19 shows a possible method to retrieve total number of each type of task, but since I have abandoned the 'Progress' and 'Done' table and used 3 status in one table to replace them, this method will not work. The data input looks like Fig.20, there are 2 lists to be replaced by Django's template tag, what I did not understand was the context in views.py can only output a dictionary, how can I output a list? And how to put 3 aggregation into one list only based on their status? So I learned about Dajngo's method of calculating total numbers, how to extract a list from a dictionary, how to use Django shell to check output data, how to use template tags for filtering values, etc.

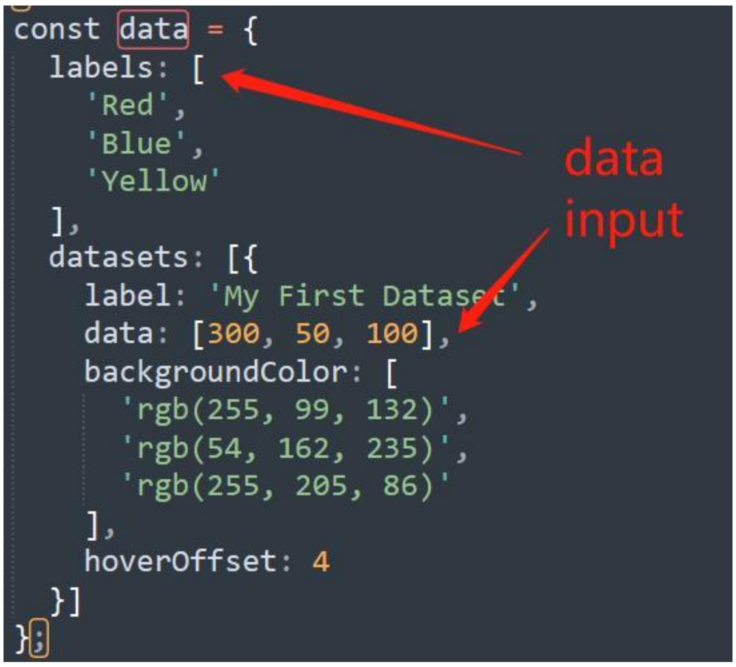
Then I came up with a solution like Fig.21, it is actually pretty close since I know that I must use Django's Aggregate function at this moment. Then as shown in Fig.22, Fig.23 and Fig.24, these 3 methods have the same result, which is that the doughnut chart wasn't displayed properly, only the title shown. It turned out that whether it's JSON, Queryset, list or dictionary, I have failed to identify their differences back then, that was the reason why the data did not show in doughnut chart properly, until people in Stack Overflow remind me to use Django Shell to check the output data format then I can see the format of data my codes output, it was quicker than trying over and over again in Chart.js. The final code can be seen as Fig.24.

The Gantt chart and Calendar from Google chart have also been through many obstacles like the doughnut chart, however I have successfully overcame the difficulty and learned how to deal with programming issue I don't know, I've learned how to check official documents, how to learned for new knowledge, how to adjust my mindset and how to ask for help in Stack Overflow.



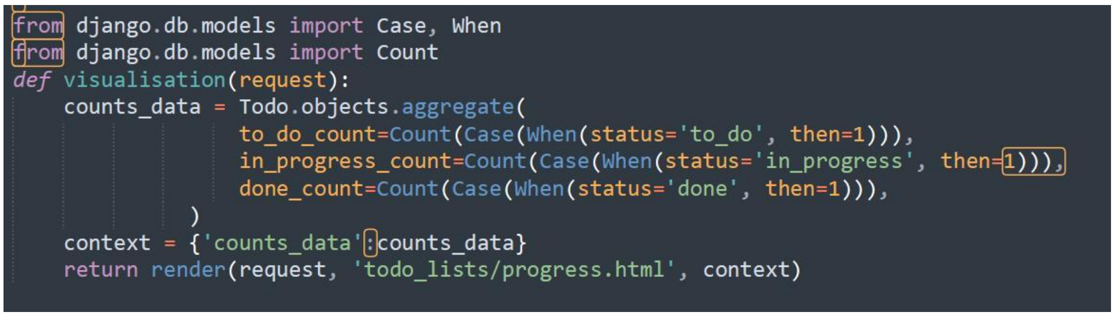
```python
def visualisation(request):

    todo = Todo.objects.all()
    todo_count = todo.count()
    progress = Progress.objects.all()
    progress_count = progress.count()
    done = Done.objects.all()
    done_count = done.count()

    if request.method != 'POST':
        # first time filled with current content
        form = TodoForm(instance=todo)
    else:
        form = TodoForm(instance=todo, data=request.POST)
        if form.is_valid():
            form.save()
            return HttpResponseRedirect(reverse('todo_lists:project',
                                                args=[project.id]))
    context = {
        'form': form,
        'todo_count': todo_count,
        'progress_count': progress_count,
        'done_count': done_count,
    }
    return render(request, 'todo_lists/progress.html', context)
```

Fig.19 One of the method I have tried to provide total number to doughnut chart



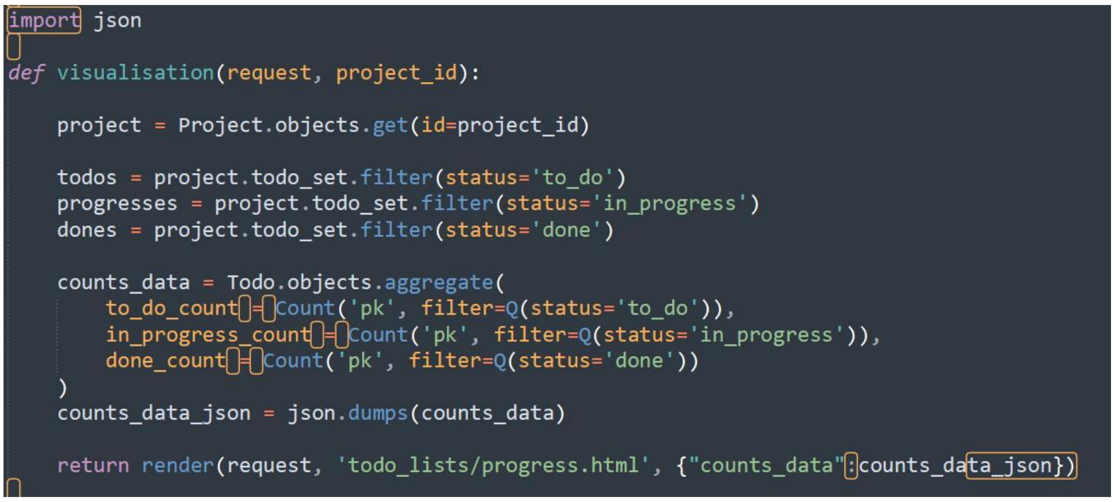
```
const data = {
  labels: [
    'Red',
    'Blue',
    'Yellow'
  ],
  datasets: [{
    label: 'My First Dataset',
    data: [300, 50, 100],
    backgroundColor: [
      'rgb(255, 99, 132)',
      'rgb(54, 162, 235)',
      'rgb(255, 205, 86)'
    ],
    hoverOffset: 4
  }]
};
```

Fig.20 Data input format



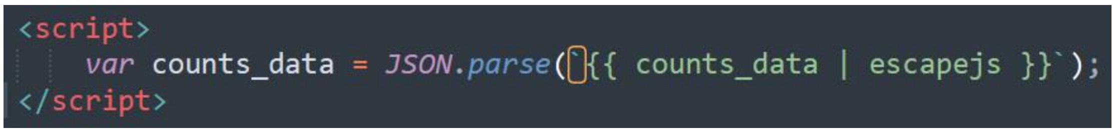
```
from django.db.models import Case, When
from django.db.models import Count
def visualisation(request):
    counts_data = Todo.objects.aggregate(
                    to_do_count=Count(Case(When(status='to_do', then=1))),
                    in_progress_count=Count(Case(When(status='in_progress', then=1))),
                    done_count=Count(Case(When(status='done', then=1))),
            )
    context = {'counts_data':counts_data}
    return render(request, 'todo_lists/progress.html', context)
```

Fig.21 One of the method that tried to calculate total number of different task



```
import json

def visualisation(request, project_id):

    project = Project.objects.get(id=project_id)

    todos = project.todo_set.filter(status='to_do')
    progresses = project.todo_set.filter(status='in_progress')
    dones = project.todo_set.filter(status='done')

    counts_data = Todo.objects.aggregate(
        to_do_count = Count('pk', filter=Q(status='to_do')),
        in_progress_count = Count('pk', filter=Q(status='in_progress')),
        done_count = Count('pk', filter=Q(status='done'))
    )
    counts_data_json = json.dumps(counts_data)

    return render(request, 'todo_lists/progress.html', {"counts_data":counts_data_json})
```

Fig.22 I tried JSON format for the output



```
<script>
    var counts_data = JSON.parse(`{{ counts_data | escapejs }}`);
</script>
```

Fig.23 parse JSON format in HTML page

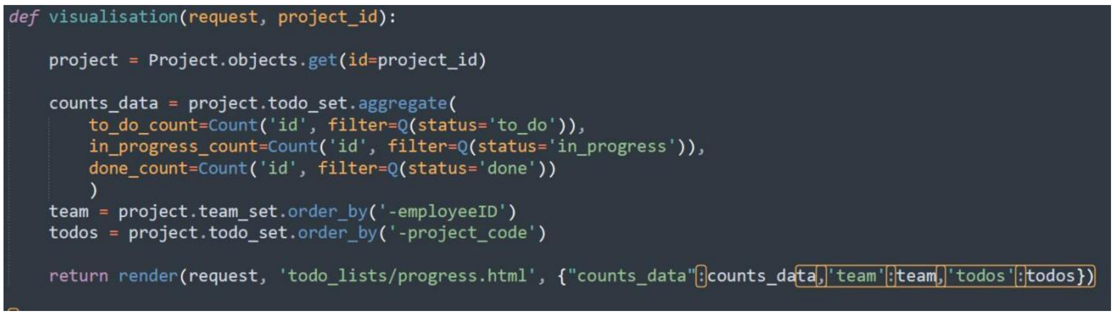

```python
def visualisation(request, project_id):

    project = Project.objects.get(id=project_id)

    counts_data = project.todo_set.aggregate(
        to_do_count=Count('id', filter=Q(status='to_do')),
        in_progress_count=Count('id', filter=Q(status='in_progress')),
        done_count=Count('id', filter=Q(status='done'))
        )
    team = project.team_set.order_by('-employeeID')
    todos = project.todo_set.order_by('-project_code')

    return render(request, 'todo_lists/progress.html', {"counts_data":counts_data,'team':team,'todos':todos})
```

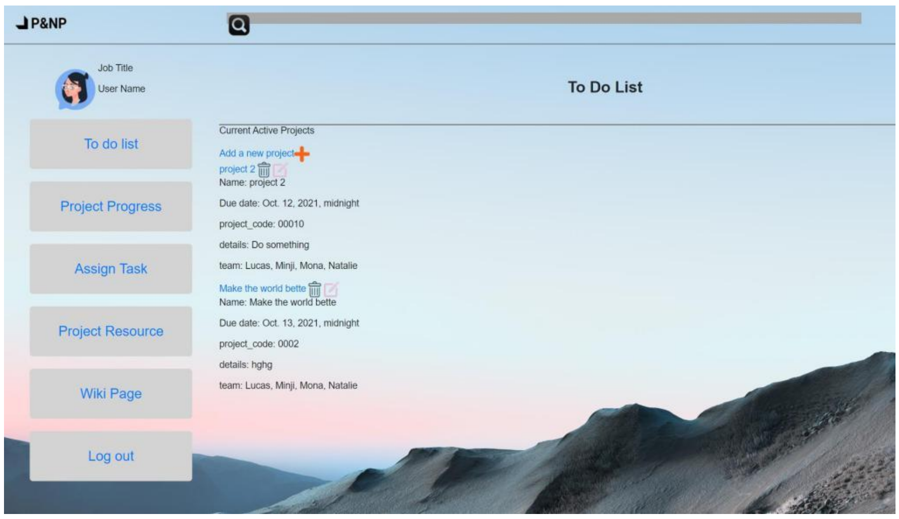
Fig. 25 Final method for providing data to doughnut chart

6. Create HTML pages and add HTTP request

This part as mentioned above was completed by back end team members, me and Mona, Fig.1 shown the HTML pages I have made. We created our own HTML pages in accordance with our own methods in views.py and urls.py. Fig.12 and Fig.13 showed some examples for using Django’s template tags in HTML. The template tags extract data from function’s ‘context’ defined in views.py, which is in format of dictionary, I have went into lots of error for knowing not much of how dictionary transit data.

7. Applied Bootstrap to all HTML pages in to-do list and visualization

As a back-end role in the team, my priority is to make sure all the functions run as planned. After all the testing and adjustments that has been made to Django’s codes, all the functions are working perfectly, at this moment, it was the last weeks of COS60010, there was no enough time for front-end team to learn about Django’s template tags in HTML page, thus I offered to apply Bootstrap to the pages in to-do lists and visualization. Fig. 26, 27 and 28 showed what our pages looked like on 27/10/2021, which was 2 days before the presentation. And Fig. 29 and 30 showed the example after applying Bootstrap style. I suggested Natalie use a template from Start Bootstrap to transform the project’s base.html (header, sidebar and footer), later she changed the main style using template structures.



To Do List

Current Active Projects

Add a new project

project 2

Name: project 2

Due date: Oct. 12, 2021, midnight

project_code: 00010

details: Do something

team: Lucas, Minji, Mona, Natalie

Make the world bette

Name: Make the world bette

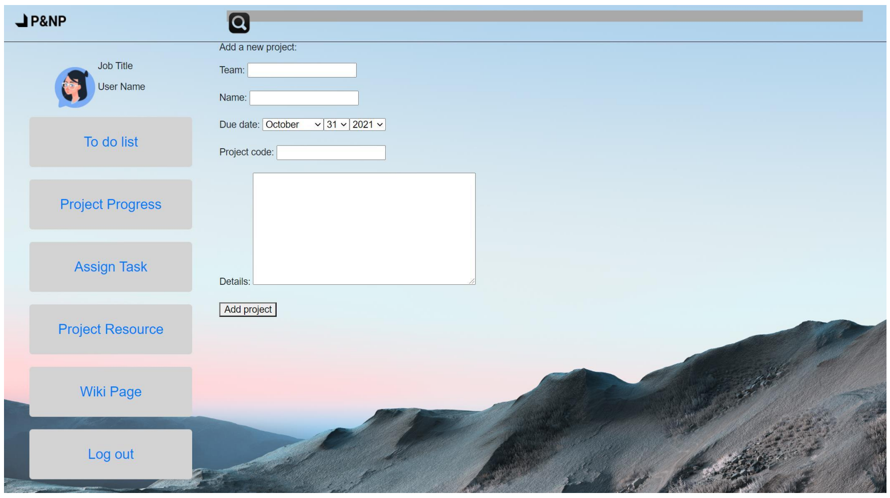
Due date: Oct. 13, 2021, midnight

project_code: 0002

details: hghg

team: Lucas, Minji, Mona, Natalie

Fig. 26 interface for displaying all current projects



Add a new project:

Team:

Name:

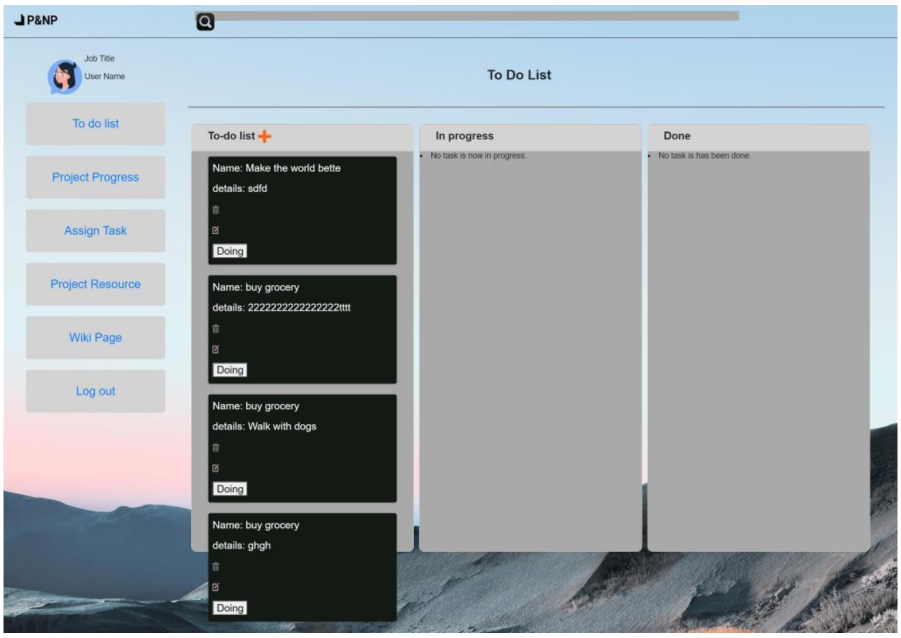
Due date: October 31 2021

Project code:

Details:

Add project

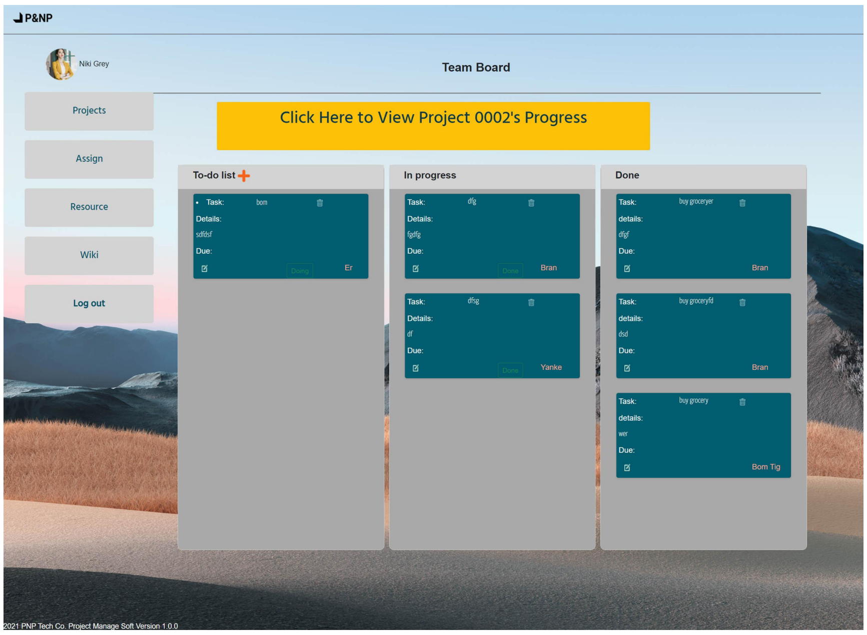
Fig. 27 Interface for creating a new project



To Do List

To-do list	In progress	Done
Name: Make the world bette details: sdfd Doing	• No task is now in progress.	• No task is has been done.
Name: buy grocery details: 2222222222222222tttt Doing		
Name: buy grocery details: Walk with dogs Doing		
Name: buy grocery details: ghgh Doing		

Fig. 28 Interface for tasks


P&NP
Niki Grey
Projects
Assign
Resource
Wiki
Log out
Team Board
Click Here to View Project 0002's Progress
To-do list
In progress
Done
2021 PNP Tech Co. Project Manage Soft Version 1.0.0

To-do list

- Task: bom
  Details: sdfdsf
  Due:
  Doing Er

In progress

Task: dfg
Details: fgdfg
Due:
Done Bran

Task: dfsg
Details: df
Due:
Done Yanke

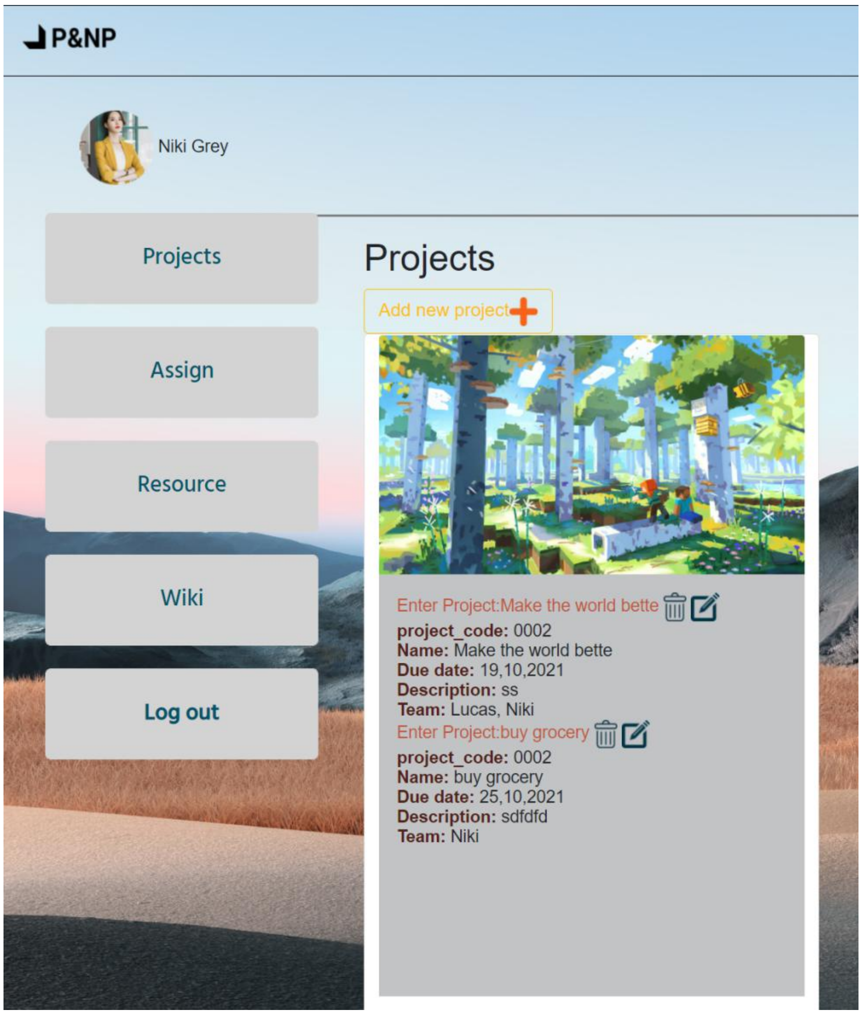
Done

Task: buy groceryer
details: dfgf
Due:
Bran

Task: buy groceryfd
details: dsd
Due:
Bran

Task: buy grocery
details: wer
Due:
Bom Tig

Fig.29 Interface for tasks after using Bootstrap



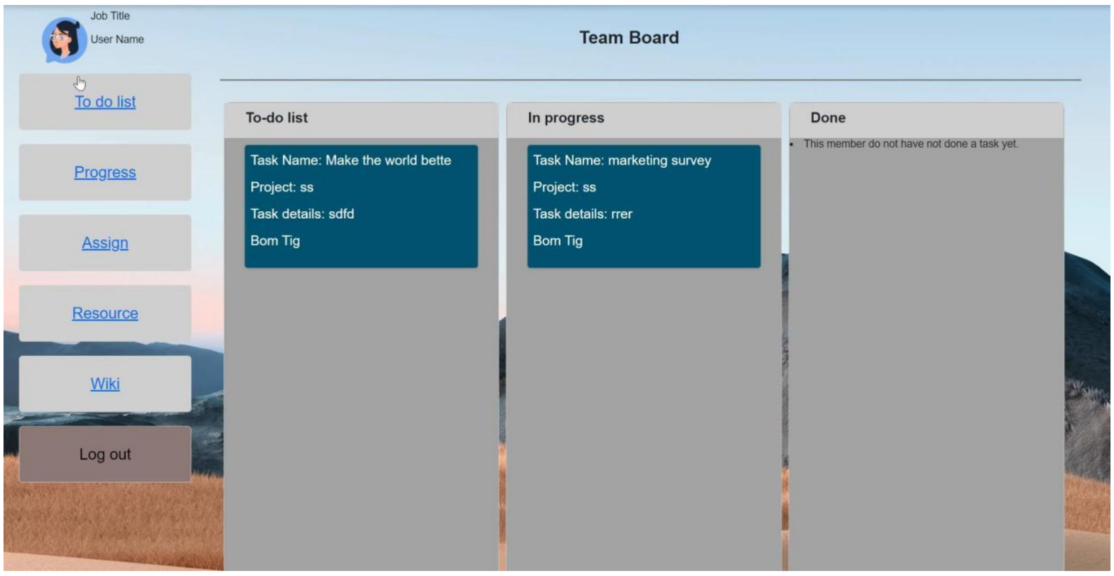
Enter Project:Make the world bette
project_code: 0002
Name: Make the world bette
Due date: 19,10,2021
Description: ss
Team: Lucas, Niki
Enter Project:buy grocery
project_code: 0002
Name: buy grocery
Due date: 25,10,2021
Description: sdfdfd
Team: Niki

Fig. 30 Interface for tasks after using Bootstrap

8. Helping other team members in programming

I have tried my best to cooperate with teammates in programming, mostly in the HTML pages that contain Django's template tag and back end algorithms. Take Natalie as an example, she was doing Assign page in Django, however, the assign function is one of the function I implemented in to-do list already, so I encouraged her to display team members' assign situation in other way, which there was a page that displaying all team members and click the member's name, all the tasks the member was doing can be seen, as shown in Fig.31, it's very similar to the my to do list function, however, to do list display all the task associated with teammate, and assign page display all teammate associated with task. In order to accomplish the function, she needed to know how my models and algorithm works in the to-do list, hence I discussed with her and helped her understand the logic I have used and provide assistance in building the assign page.

I also worked with Mona in the database, because we had one table called 'Team' that was shared, she used 'Team' to create a login and register function, and I used it to connect team members to tasks. The example as shown in Fig. 32 is my definition of the table. Mona's register function required the table to output a dictionary, but my function can accept the dictionary. We have a lot of cases like these that need to work together.

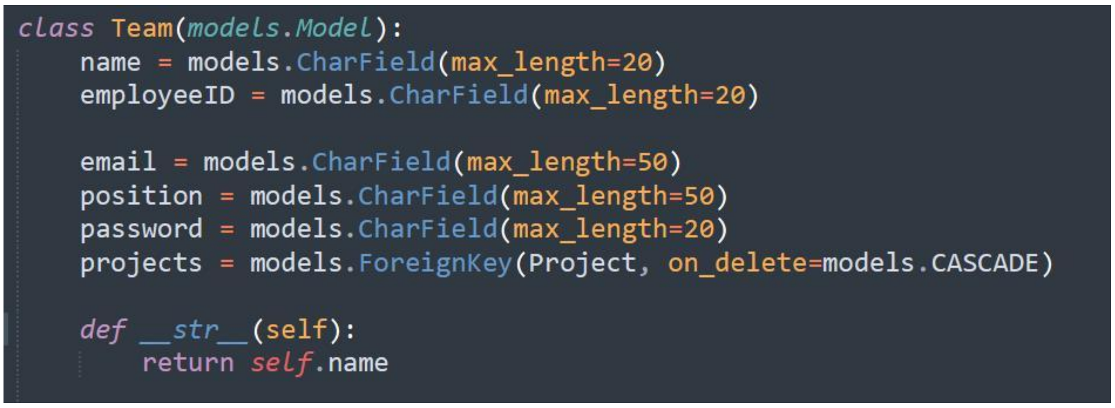


Task Name: Make the world bette
Project: ss
Task details: sdfd
Bom Tig

Task Name: marketing survey
Project: ss
Task details: rrer
Bom Tig

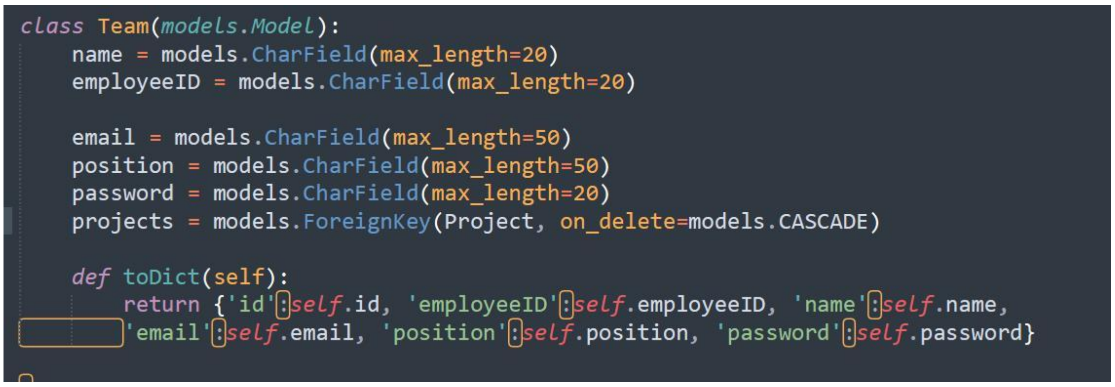
- This member do not have not done a task yet.

Fig. 31 The page that shows the task a member is doing



```python
class Team(models.Model):
    name = models.CharField(max_length=20)
    employeeID = models.CharField(max_length=20)

    email = models.CharField(max_length=50)
    position = models.CharField(max_length=50)
    password = models.CharField(max_length=20)
    projects = models.ForeignKey(Project, on_delete=models.CASCADE)

    def __str__(self):
        return self.name
```

Fig.32 My Team table

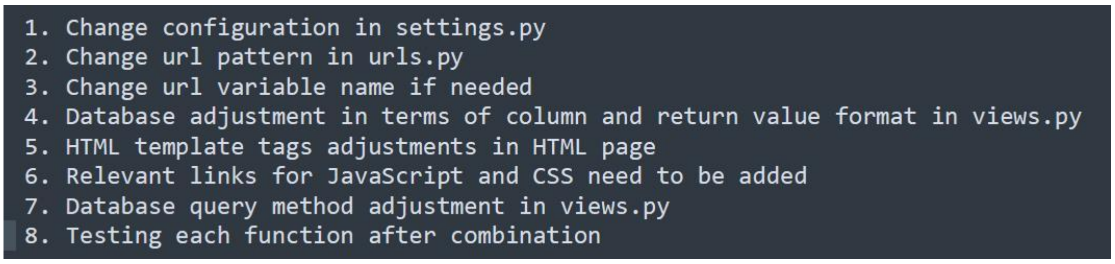


```python
class Team(models.Model):
    name = models.CharField(max_length=20)
    employeeID = models.CharField(max_length=20)

    email = models.CharField(max_length=50)
    position = models.CharField(max_length=50)
    password = models.CharField(max_length=20)
    projects = models.ForeignKey(Project, on_delete=models.CASCADE)

    def toDict(self):
        return {'id':self.id, 'employeeID':self.employeeID, 'name':self.name,
                'email':self.email, 'position':self.position, 'password':self.password}
```

Fig.33 Mona's Team table

9. Combine functions in testing phase

The testing part in this project was mainly code migrations and combinations, and has been done by me, Natalie and Mona. We have developed different parts of code in different computers by different teammates, in order to make all the functions work properly as a whole, it is necessary to combine codes and test all the functions as a whole system. Every time there was an adjustment there was a need for code migrations and testing. The process can be summarized as Fig. 34. The most challenging part for testing is dealing with all kinds of errors.



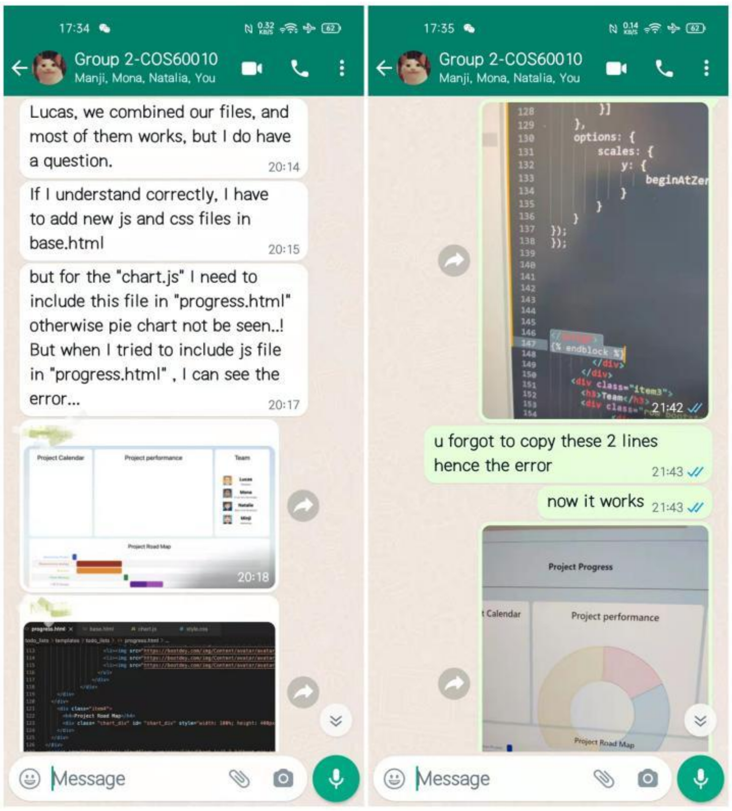
```
1. Change configuration in settings.py
2. Change url pattern in urls.py
3. Change url variable name if needed
4. Database adjustment in terms of column and return value format in views.py
5. HTML template tags adjustments in HTML page
6. Relevant links for JavaScript and CSS need to be added
7. Database query method adjustment in views.py
8. Testing each function after combination
```

Fig. 34 Things need to do when combine codes

3. Learning outcome reflection

I have learned many things from this course. In the technical part, firstly I have learned Python and methods of using Python to create web applications in Django. Secondly, I have obtained knowledge of using databases in real projects. Thirdly, I developed an understanding in software development and project management in an Agile environment. Fourthly, I found a way that suits me to study new programming knowledge, I would be more confident in the future for studying new knowledge.

In no-technical part, firstly I have learned how to work in a team for software development, the real project built by the team teach me the importance of team collaboration. Fig.35 is screenshot from group chat, it shows the example of team supports, for privacy purposes I have remove teammate’s name. Secondly, I am aware that making acknowledgment to other’s works is more important than emphasizing my own work. Listening to others' ideas rather than denying can make the team more united and proactive. Fig.36 shows an example of making acknowledgement to other’s work.



17:34

Group 2-COS60010
Manji, Mona, Natalia, You

Lucas, we combined our files, and most of them works, but I do have a question. 20:14

If I understand correctly, I have to add new js and css files in base.html 20:15

but for the "chart.js" I need to include this file in "progress.html" otherwise pie chart not be seen..! But when I tried to include js file in "progress.html" , I can see the error... 20:17

20:18

17:35

Group 2-COS60010
Manji, Mona, Natalia, You

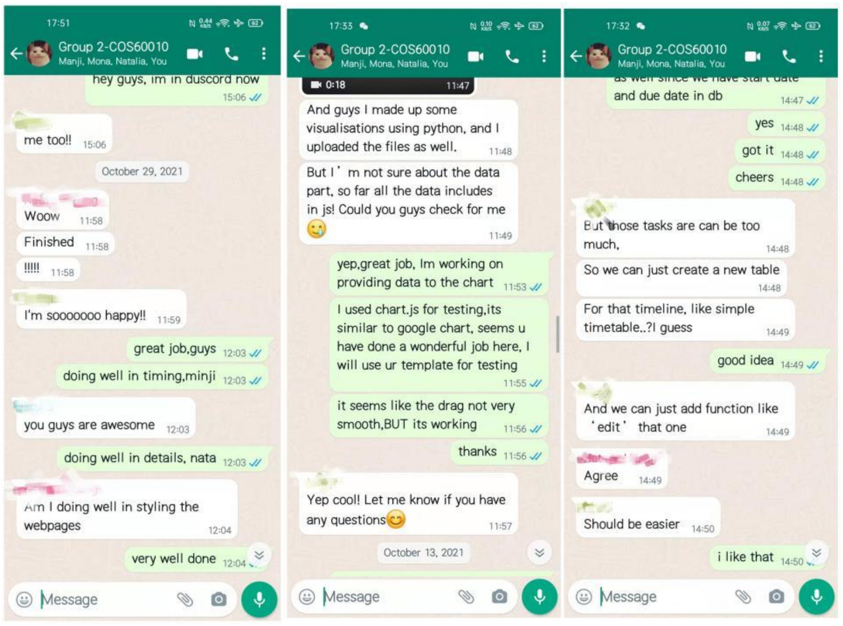
```
            }]
        },
        options: {
            scales: {
                y: {
                    beginAtZer
                }
            }
        }
    });
    });

{% endblock %}
        </div>
    </div>
    <div class="item3">
        <h3>Team</h3>
        <div class=
```

21:42

u forgot to copy these 2 lines hence the error 21:43

now it works 21:43

Fig. 35 An example of team support



17:51

Group 2-COS60010
Manji, Mona, Natalia, You

hey guys, im in duscord now 15:06

[redacted]
me too!! 15:06

October 29, 2021

[redacted]
Woow 11:58

Finished 11:58

!!!!! 11:58

[redacted]
I'm sooooooo happy!! 11:59

great job,guys 12:03

doing well in timing,minji 12:03

[redacted]
you guys are awesome 12:03

doing well in details, nata 12:03

[redacted]
Am I doing well in styling the webpages 12:04

very well done 12:04

Message

17:33

Group 2-COS60010
Manji, Mona, Natalia, You

0:18 11:47

And guys I made up some visualisations using python, and I uploaded the files as well. 11:48

But I' m not sure about the data part, so far all the data includes in js! Could you guys check for me 😢 11:49

yep,great job, Im working on providing data to the chart 11:53

I used chart.js for testing,its similar to google chart, seems u have done a wonderful job here, I will use ur template for testing 11:55

it seems like the drag not very smooth,BUT its working 11:56

thanks 11:56

[redacted]
Yep cool! Let me know if you have any questions😊 11:57

October 13, 2021

Message

17:32

Group 2-COS60010
Manji, Mona, Natalia, You

as well since we have start date and due date in db 14:47

yes 14:48

got it 14:48

cheers 14:48

[redacted]
But those tasks are can be too much, 14:48

So we can just create a new table 14:48

For that timeline, like simple timetable..?I guess 14:49

good idea 14:49

[redacted]
And we can just add function like 'edit' that one 14:49

[redacted]
Agree 14:49

[redacted]
Should be easier 14:50

i like that 14:50

Message

Fig. 36 An example of acknowledgement to other's work

4. Other contributions

I have never missed one group meeting, always actively participating in discussing group plans and design. As mentioned above, I have actively helped teammates that were doing front-end page understanding how Django works, also I worked closely with front-end teammates to make CSS style and JavaScript work in my HTML pages with Django's template tags. I believe through active cooperation, everyone can learn new knowledge and make progress. In the final presentation, our team made the decision to send 2 people to show the team's achievement due to a 10 minutes restriction. I was not one of them, but I helped with writing the slides and running the rehearsal.

5. Summary

In this article, I have reviewed the contribution to the team in technical and non-technical parts. In the technical part, I have completed main functions for the project, which is the to-do list part as well as project visualization part, both frontend and backend mostly. In addition I cooperated with other teammates in building the assign page. I also contributed to code combination, function testing and application of Bootstrap to HTML pages.

In the non-technical part, I tried my best to participate in every meeting and discuss ideas with other teammates. I provided support to their issues in programming and testing and I united the team by acknowledging and appreciating their work.

6. Reference

[1] Severance, C.Model View Controller (MVC). CharlesRussell Severance, 2021.