To-Do List

Mini Project Report

Open Source Technology Lab

Submitted by

Shrayash Alshi	18CE1055
Sarthak Chawande	18CE1078
Manish Bhoir	18CE1007
Sahil Raut	16CE1042

Guided by

Sumithra T.V

Department of Computer Engineering

Ramrao Adik Institute of Technology

Dr. D. Y. Patil Vidyanagar,Sector 7, Nerul, Navi Mumbai 400 706.

(Affiliated to University of Mumbai)

April – 2020

ABSTRACT

Python is a programming language that combines the features of C and Java. It offers an elegant style of developing programs like C. When the programmers want to go for object orientation, Python offers classes and objects like Java. The code in Python is easy to understand and develop. Hence, Python is gaining some popularity among the programming folks.

Python was first developed by Guido Van Rossum in the year 1991 at the centre for Mathematics and Computer Science managed by the Dutch government. Van Rossum was working on the project to develop system utilities in C where he had to interact with the Bourne shell available in UNIX. His urge to fill the gap between C and shell had led to the creation of Python. He released the first version of Python on February 20, 1991. The logo of the Python is intertwined snakes.

Python is the open source software and hence can be used easily for developing different projects.

Index

Introduction

For to-do list we made use of the Django framework of python. Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source.

Django's primary goal is to ease the creation of complex, database-driven websites. The framework emphasizes reusability and "pluggability" of components, less code, low coupling, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings files and data models. Django also provides an optional administrative create , read, update and delete interface that is generated dynamically through introspection and configured via admin models.

Some well-known sites that use Django include Instagram, Mozilla, The Washington Times, Disqus, Bitbucket and Next door.

Despite having its own nomenclature, such as naming the callable objects generating the HTTP responses "views",] the core Django framework can be seen as an MVC architecture. It consists of an object -relational mapper (ORM) that mediates between data models (defined as Python classes) and a relational database ("**M**odel"), a system for processing HTTP requests with a web templating system ("**V**iew"), and a regular-expression-based URL dispatcher ("**C**ontroller").

Also included in the core framework are:

- a lightweight and standalone web server for development and testing
- a form serialization and validation system that can translate between HTML forms and values suitable for storage in the database.
- a template system that utilizes the concept of inheritance borrowed from object-oriented programming
- a caching framework that can use any of several cache methods
- support for middleware classes that can intervene at various stages of request processing and carry out custom functions
- an internal dispatcher system that allows components of an application to communicate events to each other via pre-defined signals

- an internationalization system, including translations of Django's own components into a variety of languages.

- a serialization system that can produce and read XML and/or JSON representations of Django model instances

- a system for extending the capabilities of the template engine

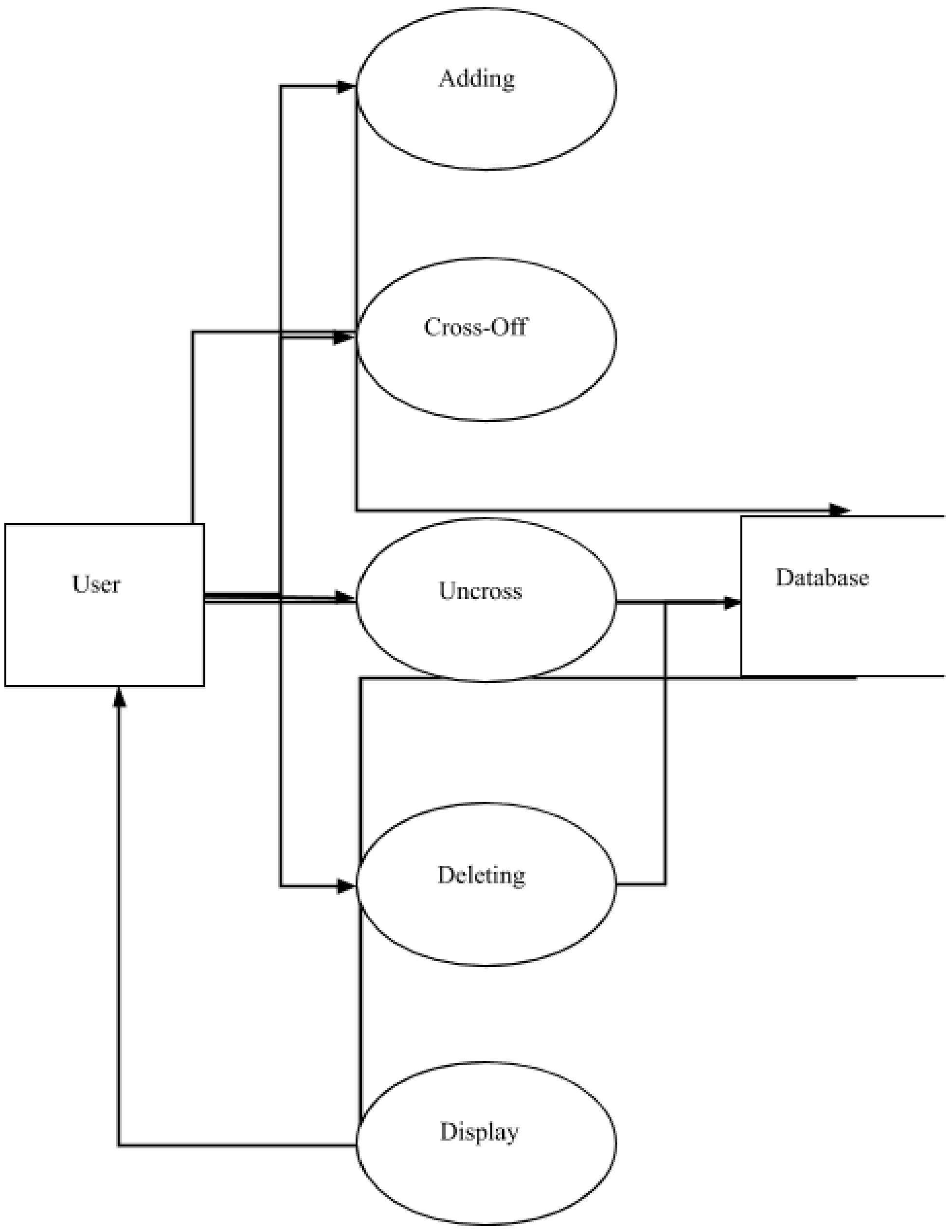
- an interface to Python's built-in unit test framework

- Django REST framework is a powerful and flexible toolkit for building Web APIs.

Design

In this project a user will request a process from the web app, the web-app will deal with or execute the function associated with that particular process. After completion of the process the web-app will make some changes in the database and display the information back to the user.

For e.g. suppose the user wishes to delete an item from the to-do list then this web app will execute the delete function of the views.py file, delete the item from the database and display the remaining items of the list to the user.



Implementation

Python offers django framework to create web applications. To create a web application using the django framework, first we need to install django in our system and then set up a feasible working environment for a django app to work properly.

Steps to install django and set up the working environment:

i. Install pip : open command prompt and enter command-python -m pip install -u pip

ii. Install virtual environment-
python install virtualenv

iii. Set virtual environment:
Setting up the environmental will allow to edit the dependency which generally system wouldn't allow-
   a. Create a virtual environmental-
   Virtualenv env_site-
   b. Change directory to env_site-
   Cd env_site
   c. Go to script directory and activate the virtual environmental-cd Scripts
   Activate

iv. Install Django-pip
Install Django

v. Return to the env_site directory
cd..

vi. Start project-Django-
Admin.py startproject to-do

vii. Change directory to
todo-cd todo

viii. Start the server-
python manage.py runsever

ix. Goto web browser and enter http://127.0.08000 to check whether the Django is installed or not.

In this project we have used SQLite database to store the information in the backend database. To access the contents of the database goto http://sqliteviewer.flowsoft7.com/and upload the local SQLite file. For the frontend framework we made use of Bootstrap, CSS, HTML.

Program Code

models.py:

```
from django.db import models

class List(models.Model):
   item=models.CharField(max_length=200)
   completed=models.BooleanField(default=False)


def __str__(self):

        return self.item + '|' + str(self.completed)
```

forms.py:

```
from django import forms
from .models import List
class ListForm(forms.ModelForm):
        class Meta:
                model=List
                fields=["item","completed"]
```

Urls.py:

```
from django.contrib import admin
from django.urls import path
from . import views
urlpatterns = [
  path('',views.home,name="home"),
  path('delete/<list_id>',views.delete,name="delete"),
  path('cross_off/<list_id>',views.cross_off,name="cross_off"),
  path('uncross/<list_id>',views.uncross,name="uncross"),
]
```

Views.py:

```
from django.shortcuts import render,redirect
from .models import List
from .forms import ListForm
from django.contrib import messages

def home(request):
        if request.method =='POST':
                form=ListForm(request.POST or None)
                if form.is_valid():
                        form.save()
                        all_items=List.objects.all
                        messages.success(request,('Item has been added to list!'))
                        return render(request,'home.html',{'all_items':all_items})
else:
        all_items=List.objects.all
        return render(request,'home.html',{'all_items':all_items})
```

```
def delete(request,list_id):
	item=List.objects.get(pk=list_id)
	item.delete()
	messages.success(request,('Item Has Been Deleted!'))
	return redirect('home')

def cross_off(request,list_id):
	item=List.objects.get(pk=list_id)
	item.completed=True
	item.save()
	return redirect('home')

def uncross(request , list_id):
	item=List.objects.get(pk=list_id)
	item.completed=False
	item.save()
	return redirect('home')
```

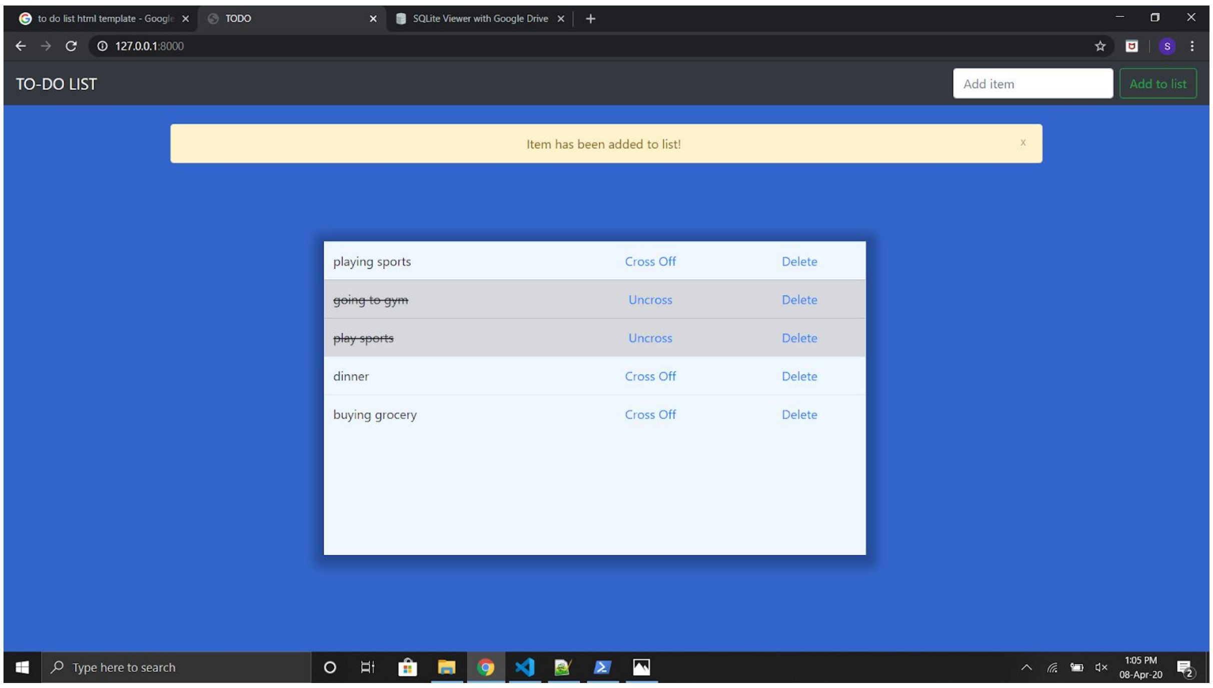
home.html:

```
{% extends 'base.html' %}
{% block content%}
   {% if messages %}
   {% for message in messages %}
<div class="alert alert-warning alert-dismissable"

role="alert"> <button class="close" data-dismiss="alert">
<small><sup>x</sup></small>
```

```
</button>
{{message}}
</div>
{% endfor %}
{% endif%}
{% if all_items %}
<table class="table table-bordered">
{% for things in all_items%}
{% if things.completed %}
<tr class="table-secondary">
                <td class="striker">{{things.item}}</td>
               <td><center><a href="{% url 'uncross' things.id
%}">Uncross</a></center></td>
           <td><center><a href="{% url 'delete'
things.id %}">Delete</a></center></td>
         </tr>
      {% else %}
          <tr>
          <td>{{things.item}}</td>
          <td><center><a href="{% url 'cross_off'
things.id %}">Cross Off</a></center></td>
          <td><center><a href="{% url 'delete'
things.id %}">Delete</a></center></td>


       </tr>
   {%endif%}
          {% endfor %}
</table>
{% endif%}
{% endblock%}
```

Output

1. After adding an item to- do list:

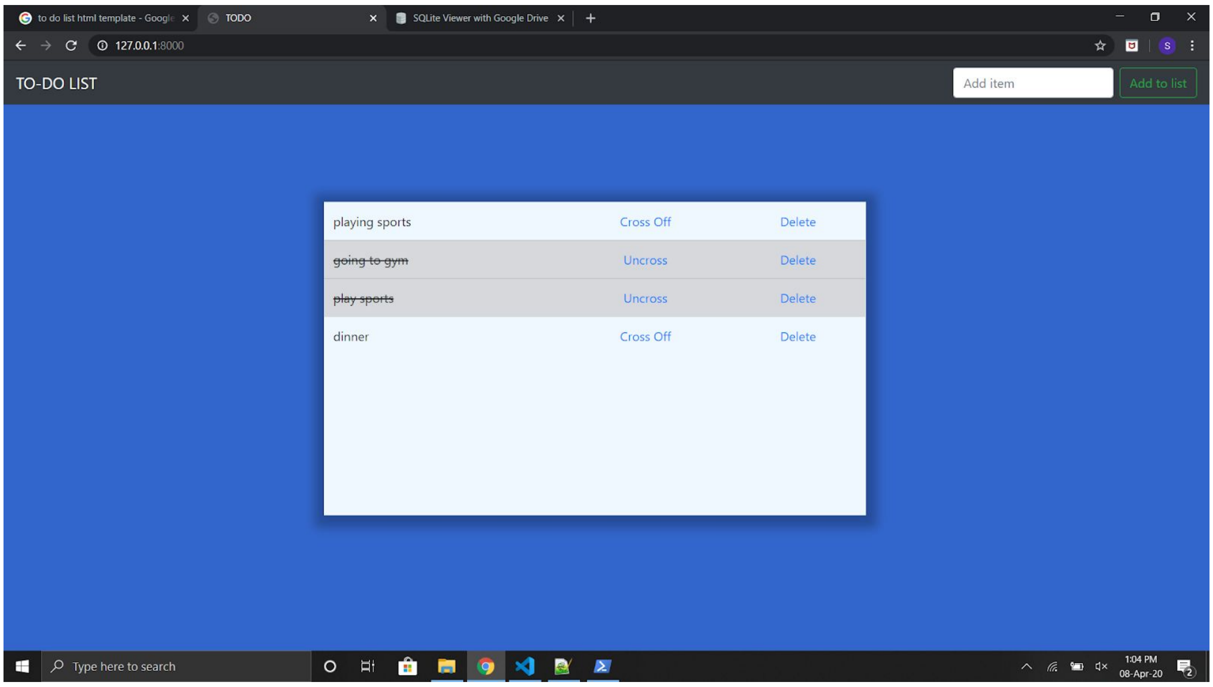


TO-DO LIST

Item has been added to list!

playing sports	Cross Off	Delete
~~going to gym~~	Uncross	Delete
~~play sports~~	Uncross	Delete
dinner	Cross Off	Delete
buying grocery	Cross Off	Delete

2. After crossing (on completion) an item:

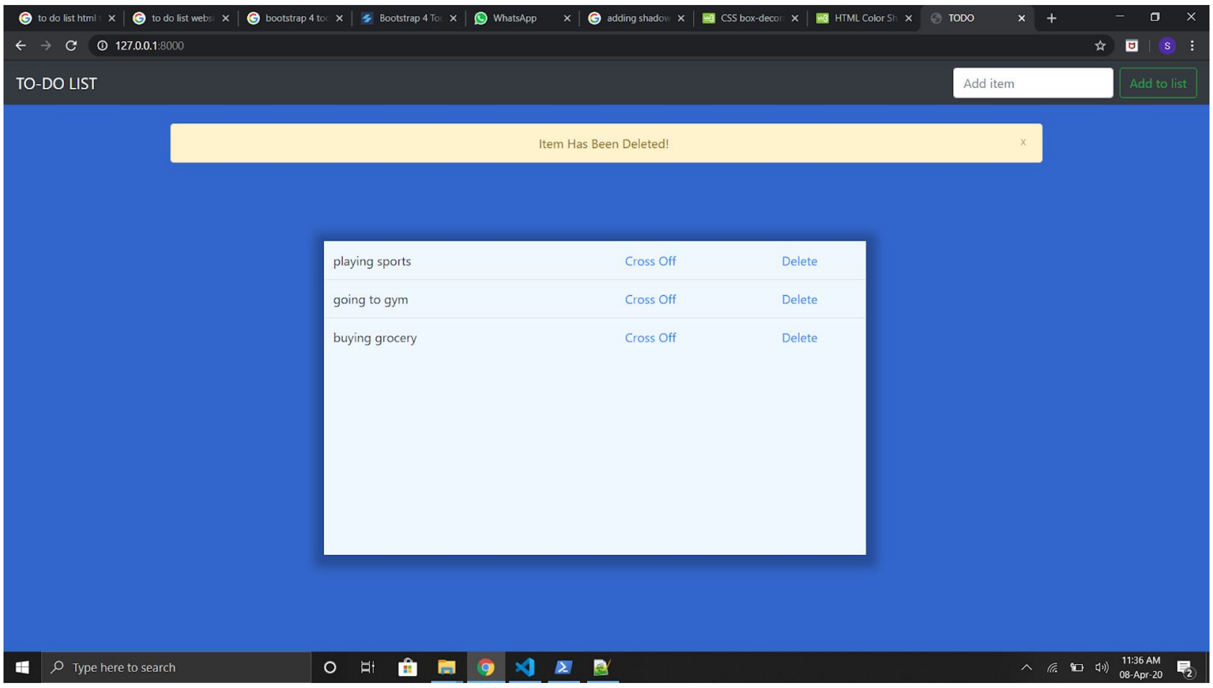


TO-DO LIST

Add item | Add to list

playing sports	Cross Off	Delete
~~going to gym~~	Uncross	Delete
~~play sports~~	Uncross	Delete
dinner	Cross Off	Delete

3. After deleting an item:



TO-DO LIST

Add item | Add to list

Item Has Been Deleted!

playing sports	Cross Off	Delete
going to gym	Cross Off	Delete
buying grocery	Cross Off	Delete

Conclusion

The project name is “To-Do-List”. In this project we can add various tasks which we have to perform. When a task is performed, we can cross off that task or if not perform we can uncross that task if we don’t want that task in the list. Once the task has been performed and has been crossed off we can delete that particular task. This webapp will be helpful to maintain a list of our day-to-day tasks.

References:

1. https://www.djangoproject.com/
2. https://www.django-cms.org/en/