Dharmsinh Desai University, Nadiad

Faculty of Technology

Department of Computer Engineering

B. Tech. CE Semester – IV

Subject : SOFTWARE PROJECT

A PROJECT REPORT ON

Project title : Multi User TO-DO App

By

CHAVDA DHARMI S. (CE-018) (19CEUBS127)

CHOKSI KANAL P. (CE-020) (19CEUOG002)

Guided by : Prof. Pinkal Chauhan &

Prof . Brijesh S. Bhatt &

Prof. Jigar Pandya

Contents

1.Abstract

Social media and other easily accessible online distractions make it hard for us to stay focused on our tasks and make it difficult for us to do our work efficiently.

Also, constantly switching between tasks may give us the false feeling that we are being productive when we are, in fact, not. It's more important for us to prioritize tasks and work on those that are most important, rather than focusing on deleting small items from our todo list just for the sake of appearances.

The goal of this app is to help us become more aware of how we spend time in the process of doing those tasks and how productive that time is. It can help set some constraints on social media to reduce distraction and track the time we spend working on the todo items. When we have a better sense of the estimated time we'll need to spend on our tasks, along with the validated time spent on the items for reference or personal/team reviews, we are able to manage our daily routines more efficiently.

Objectives:

- Maintenance of to-do's/tasks

Users Views:

- SuperUser(django superuser)
- End User

2.Introduction

1.1 Breif Introduction :

The objective of To-do app is to allow the users to manage and add their todos. It'll also facilitate keeping all the records of todos, such as their name,status,priority. So all the information about todos will be available in a few seconds as well as user can delete their todos after completion of it. Overall, it'll make todo Management an easier job for user and users can remember their daily tasks which he/she will be performed and also user can add priority of tasks so that he/she remember that which task he/she complete first. The main purpose of this document is to illustrate the requirements of the project Todoapp and is intended to help any user to maintain and manage its todos.

1.2 Tools / Technologies Used :

Technologies :

- Html
- Css
- Bootstrap
- Django
- Selenium (testing)

Tools :

- Visual Studio Code
- Git
- FireFox Browser
- WonderShare Edrawmax IDE (for design document)
- Chrome webdriver (testing)

3. Software Requirements Specifications

Multi User TO-DO APP

END-USER:

R .1 Signup

Description: user can signup here

Input : Input Credential

Output: Login Page

R .2 Login

Description: user can login here

Input : Input Credential

Output: Home page

R .2 Manage to-dos

Description: user can manage and show their to-dos

R .2.1 Add to-do

Description: user can add to-do here

Input: details of to-do

Output: “confirmation message” (to-do added successfully)

R.2.2 Remove to-do

Description: user can remove to-do here

Input: “delete option”.

Output: “confirmation message” (to-do deleted successfully)

R.2.3 Edit to-do status

Description: user can edit to-do status here

Input: select Pending/Completed button

Output: Save Changes

R.2.4 Veiw to-do

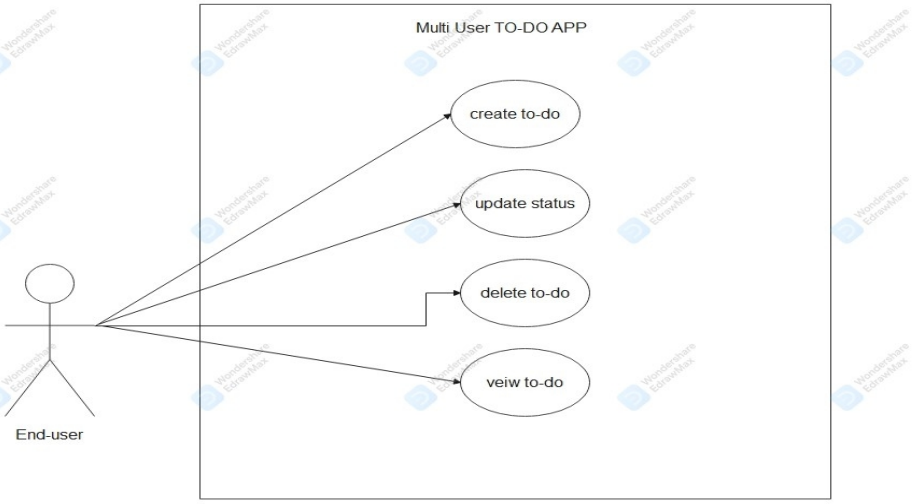
Description : user can show their to-do list here

Input : Logedin user’s credentials

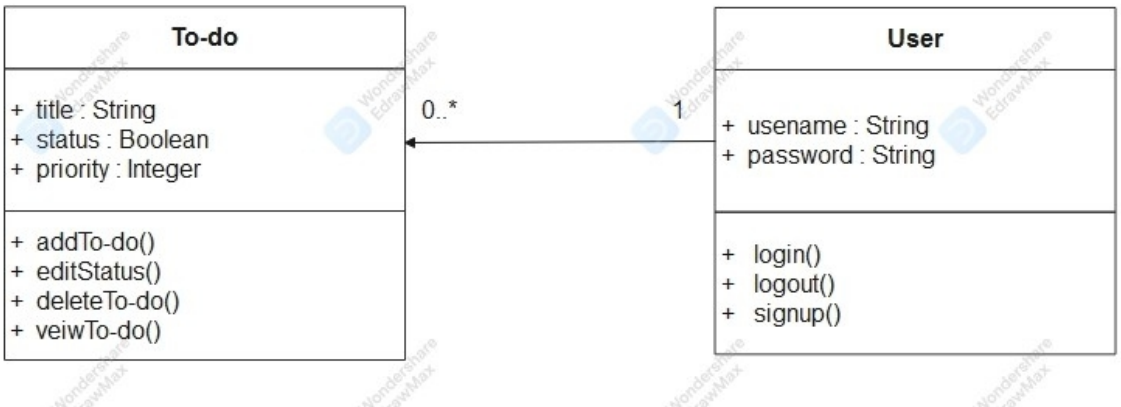
Output : Logedin User’s to-do list (on home page)

4.Design Documents

1. Use case Diagram :

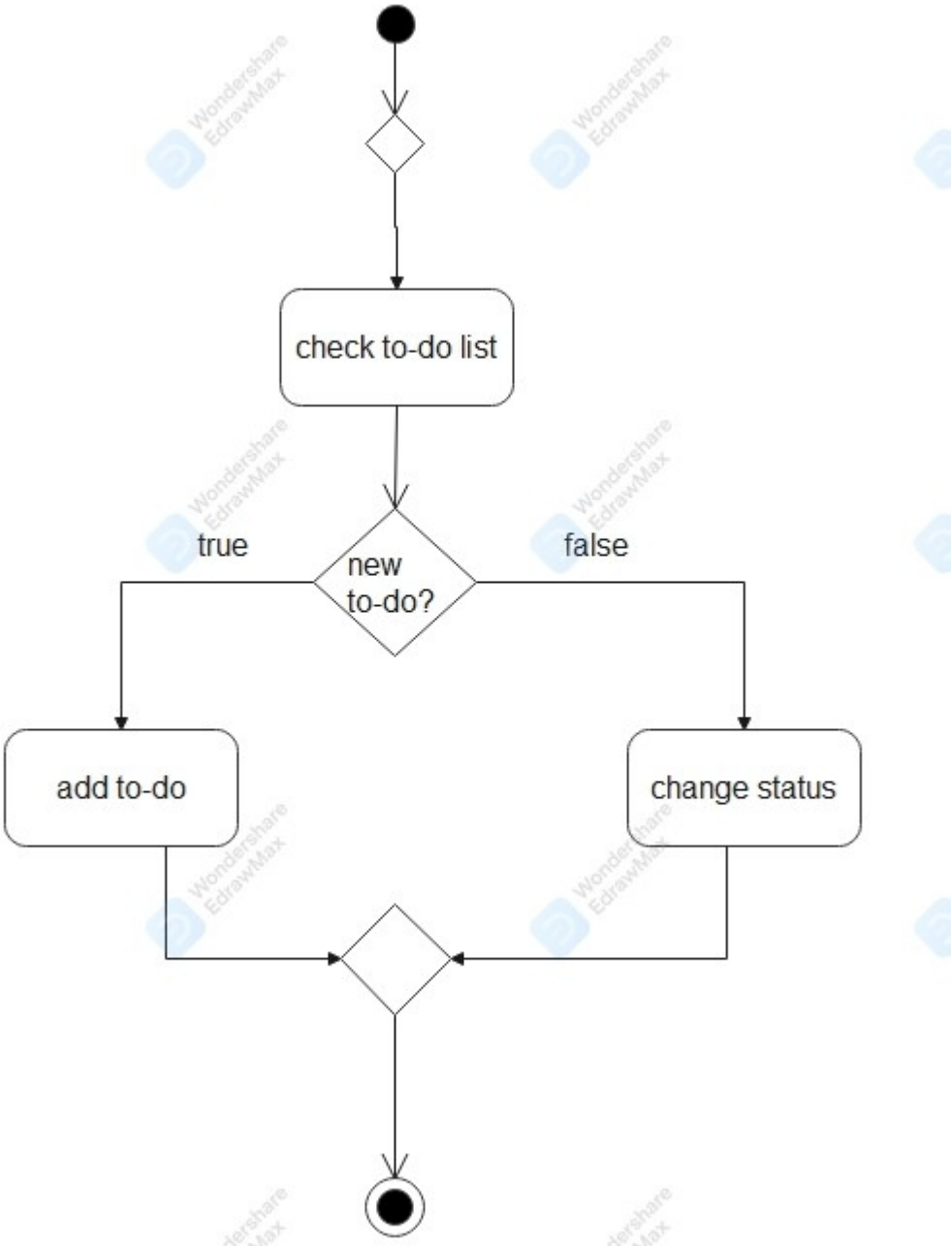


2. Class diagram :

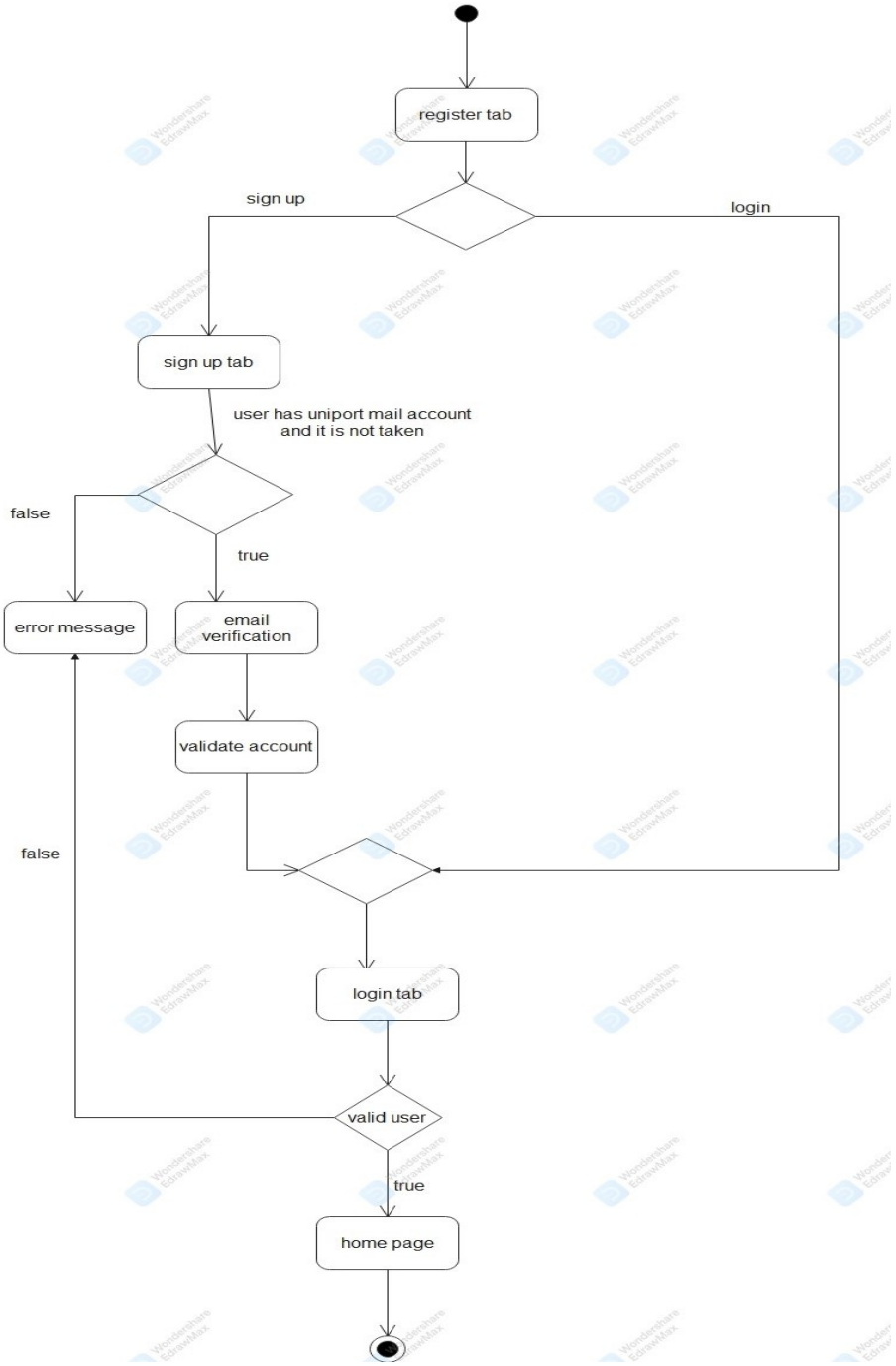


3. Activity diagram:

1)

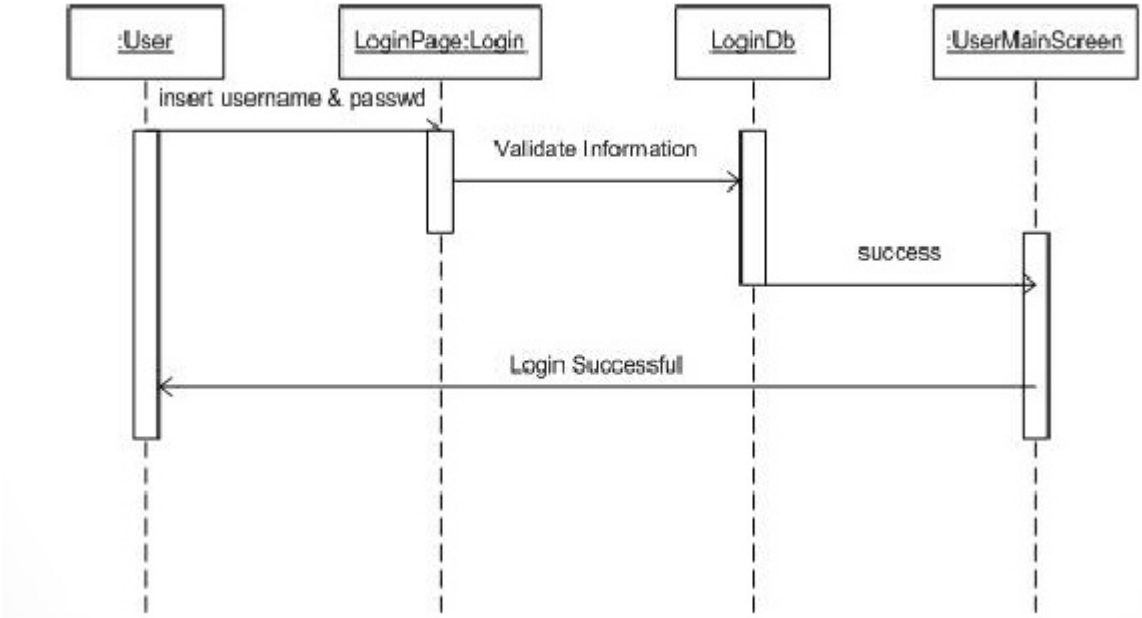


2)

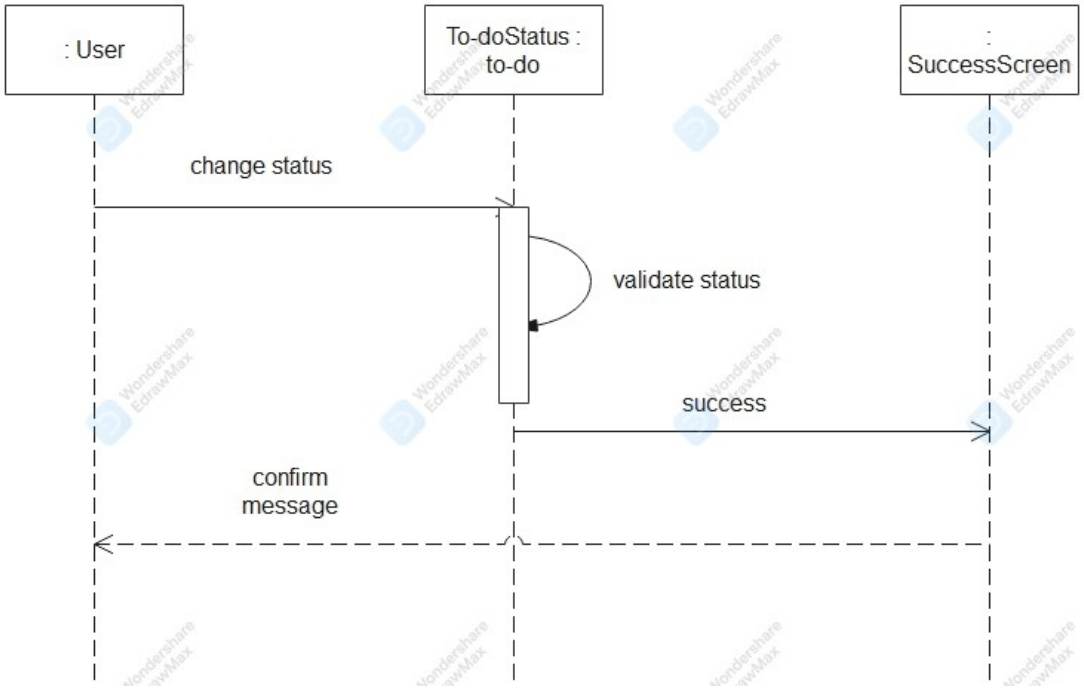

register tab
sign up
login
sign up tab
user has uniport mail account
and it is not taken
false
true
error message
email
verification
validate account
false
login tab
valid user
true
home page

4. Sequence diagram:

1)



2)



5. Data Dictionary :

USER						
Sr no.	Field name	Datatype	Width	Required	Unique	PK/FK
1	Username	varchar	256	Yes	Yes	Yes
2	Email id	varchar	256	Yes	Yes	No
3	Password	varchar	256	Yes	yes	No

TO-DO/Tasks						
Sr no.	Field name	Datatype	Width	Required	Unique	PK/FK
1	Username	varchar	256	Yes	Yes	Yes
2	Title	varchar	256	Yes	Yes	No
3	Status	varchar	256	Yes	Yes	No
4	Priority	varchar	256	Yes	No	No

5. Implementation Details

The system consists of 2 basic modules namely

- Authentication
- Manage to-do/tasks

1. Authentication Module

   Users are able to login themselves. System logs user in, then and only then user can use other functionalities of system.

2. Manage to-do/tasks Module

   User can perform crud operations (add ,update,delete)on to-do details also.here user can update task's status or delete task also can able to add new task

Functional Prototype

```python
@login_required(login_url='login')
def home(request):
    if request.user.is_authenticated:
        user = request.user
        form = TODOForm()
        todos = TODO.objects.filter(user = user).order_by('priority')
        return render(request , 'index.html' , context={'form' : form , 'todos' : todos})
```

home

```python
def login(request):
    if request.method == 'GET':
        form1 = AuthenticationForm()
        context = {
            "form" : form1
        }
        return render(request , 'login.html' , context=context )
    else:
        form = AuthenticationForm(data=request.POST)
        print(form.is_valid())
        if form.is_valid():
            username = form.cleaned_data.get('username')
            password = form.cleaned_data.get('password')
            user = authenticate(username = username , password = password)
            if user is not None:
                loginUser(request , user)
                return redirect('home')

        else:
            context = {
                "form" : form
            }
            return render(request , 'login.html' , context=context )
```

Login

```python
def signup(request):

    if request.method == 'GET':
        form = UserCreationForm()
        context = {
            "form" : form
        }
        return render(request , 'signup.html' , context=context)
    else:
        print(request.POST)
        form = UserCreationForm(request.POST)
        context = {
            "form" : form
        }
        if form.is_valid():
            user = form.save()
            print(user)
            if user is not None:
                return redirect('login')
        else:
            return render(request , 'signup.html' , context=context)
```

Signup

```python
@login_required(login_url='login')
def add_todo(request):
    if request.user.is_authenticated:
        user = request.user
        print(user)
        form = TODOForm(request.POST)
        if form.is_valid():
            print(form.cleaned_data)
            todo = form.save(commit=False)
            todo.user = user
            todo.save()
            print(todo)
            return redirect("home")
        else:
            return render(request , 'index.html' , context={'form' : form})
```

Add to-do

```python
def delete_todo(request , id ):
    print(id)
    TODO.objects.get(pk = id).delete()
    return redirect('home')

def change_todo(request , id  , status):
    todo = TODO.objects.get(pk = id)
    todo.status = status
    todo.save()
    return redirect('home')


def signout(request):
    logout(request)
    return redirect('login')
```

Delete to-do , Change status, Logout

6.Testing

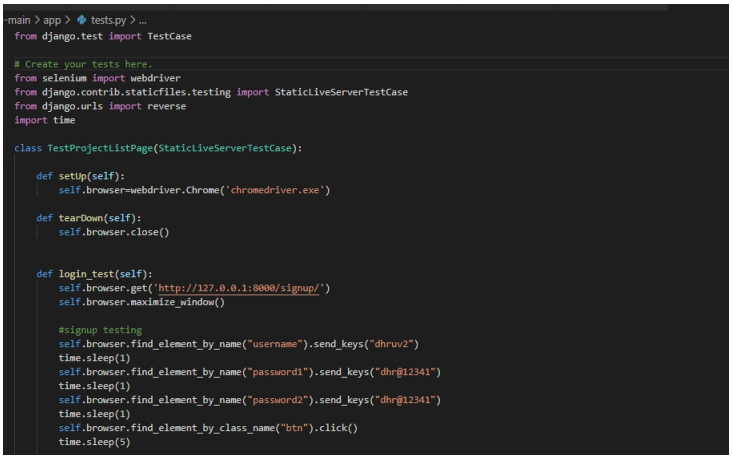
Testing method : manual testing

All test cases executed by us manually according to end user's prespective . our motto to test project is that it ensures matching with recuirment document or not.

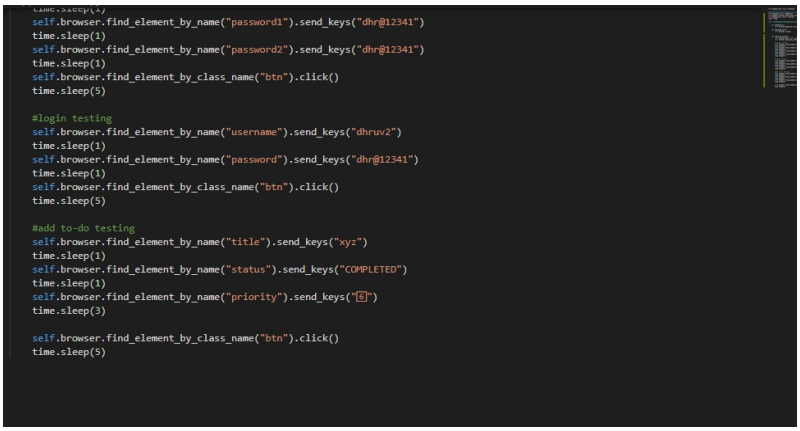
Sr. No.	Test Scenario	Expected Result	Actual Result	Status
TC_1	Login with incorrect credentials	User should not able to log in.	User is given error message. And redirected to login page.	Pass
TC_2	Login with correct credentials	User should be able to log in.	User is logged in and shown the dashboard according to input credentials.	Pass
TC_4	Delete To-do (if exists)	task matching record removed from db.	User can see confirmation message. And record are not in db.	Pass
TC_5	Add new to-do	To-do record Added into db.	User can see cofirmation message & record added into db	Pass
TC_6	Update status (if exists)	Task matching Record's status updated into db.	User can see cofirmation message & record updated into db	Pass

We also testing this app using **selenium automated testing tool.**

Screen shots of test.py file:

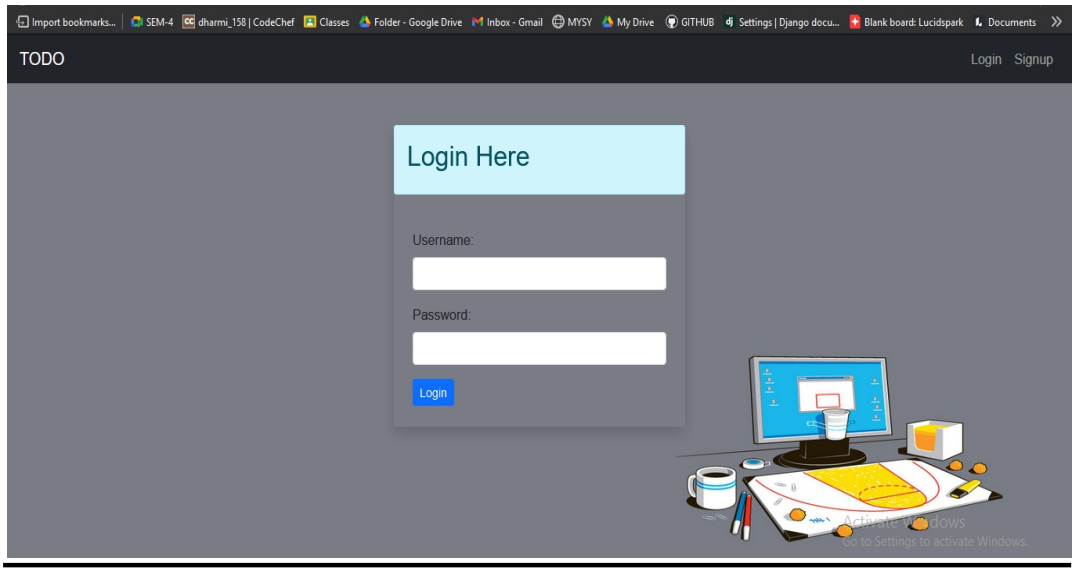


```
-main > app > tests.py > ...
from django.test import TestCase

# Create your tests here.
from selenium import webdriver
from django.contrib.staticfiles.testing import StaticLiveServerTestCase
from django.urls import reverse
import time

class TestProjectListPage(StaticLiveServerTestCase):

    def setUp(self):
        self.browser=webdriver.Chrome('chromedriver.exe')

    def tearDown(self):
        self.browser.close()


    def login_test(self):
        self.browser.get('http://127.0.0.1:8000/signup/')
        self.browser.maximize_window()

        #signup testing
        self.browser.find_element_by_name("username").send_keys("dhruv2")
        time.sleep(1)
        self.browser.find_element_by_name("password1").send_keys("dhr@12341")
        time.sleep(1)
        self.browser.find_element_by_name("password2").send_keys("dhr@12341")
        time.sleep(1)
        self.browser.find_element_by_class_name("btn").click()
        time.sleep(5)
```

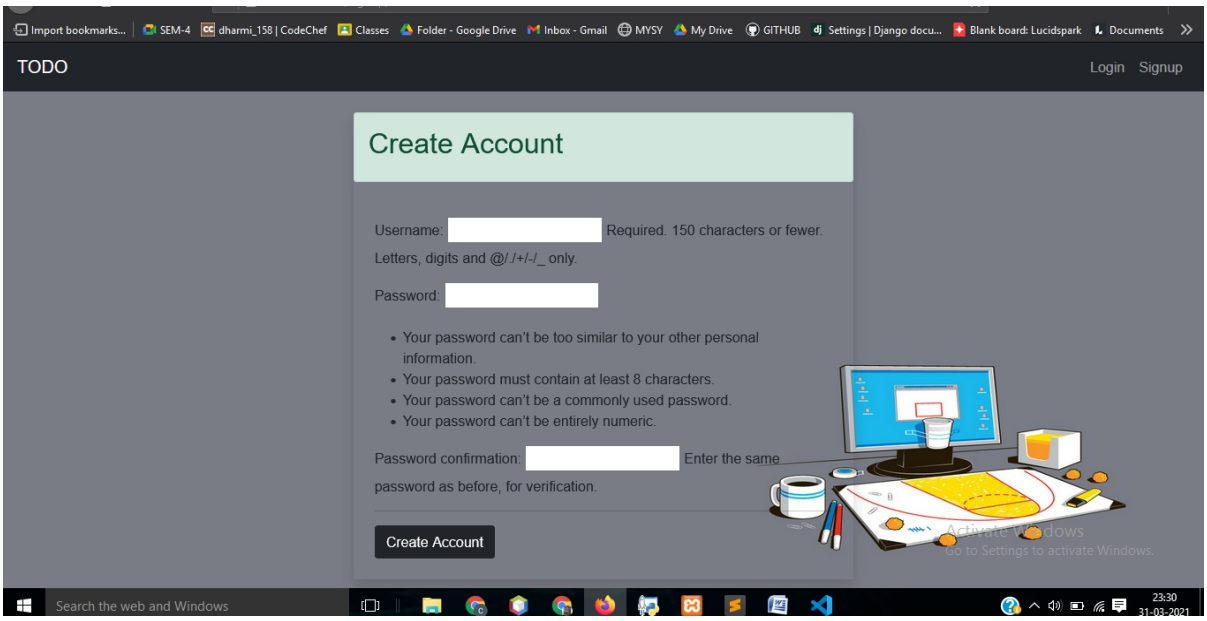


```
self.browser.find_element_by_name("password1").send_keys("dhr@12341")
time.sleep(1)
self.browser.find_element_by_name("password2").send_keys("dhr@12341")
time.sleep(1)
self.browser.find_element_by_class_name("btn").click()
time.sleep(5)

#login testing
self.browser.find_element_by_name("username").send_keys("dhruv2")
time.sleep(1)
self.browser.find_element_by_name("password").send_keys("dhr@12341")
time.sleep(1)
self.browser.find_element_by_class_name("btn").click()
time.sleep(5)

#add to-do testing
self.browser.find_element_by_name("title").send_keys("xyz")
time.sleep(1)
self.browser.find_element_by_name("status").send_keys("COMPLETED")
time.sleep(1)
self.browser.find_element_by_name("priority").send_keys("E")
time.sleep(3)

self.browser.find_element_by_class_name("btn").click()
time.sleep(5)
```

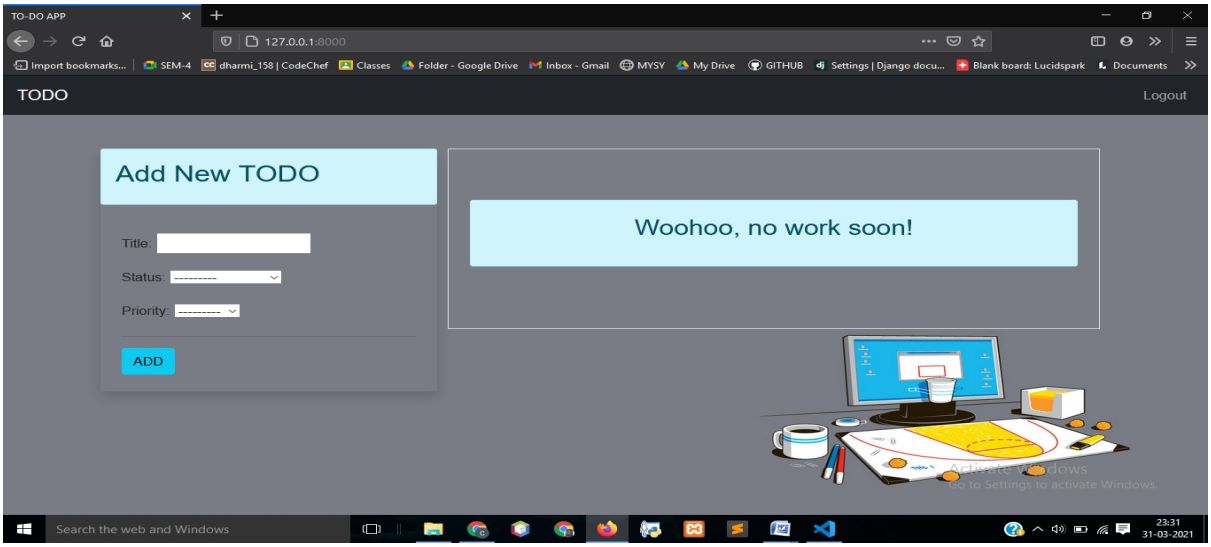
6. Workflow/Layouts



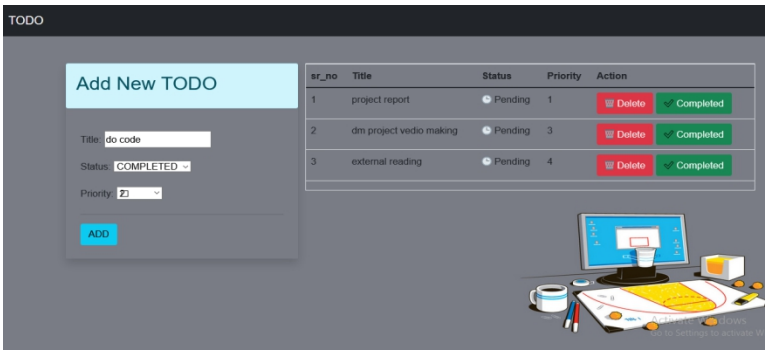
Home page/login page



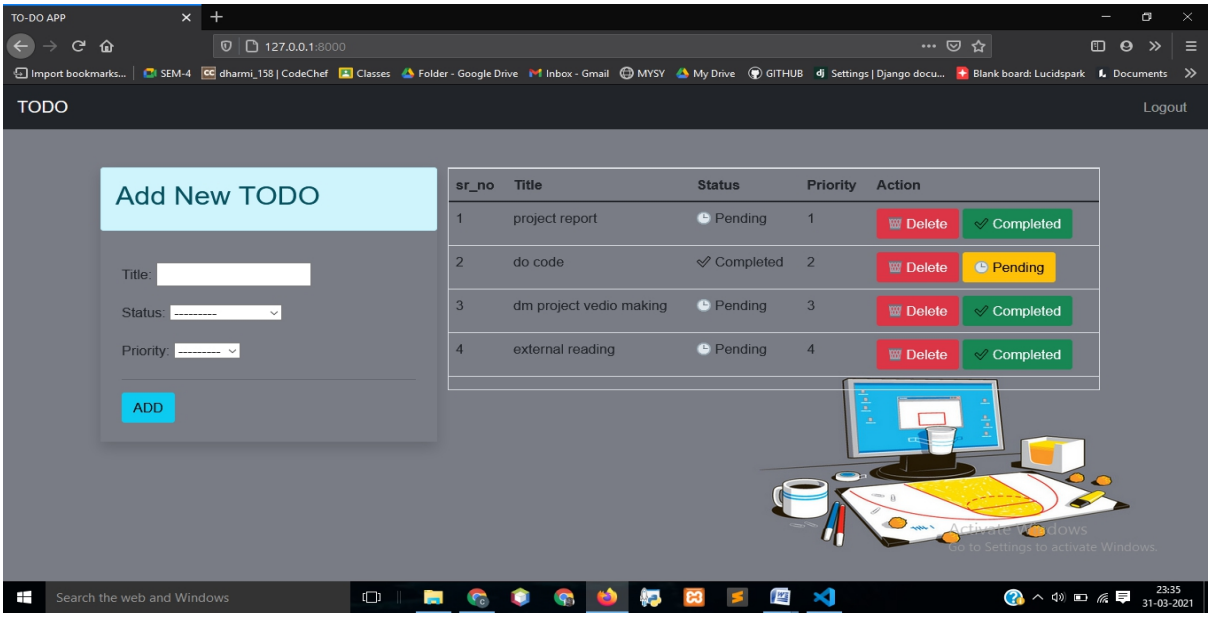
Sign up page



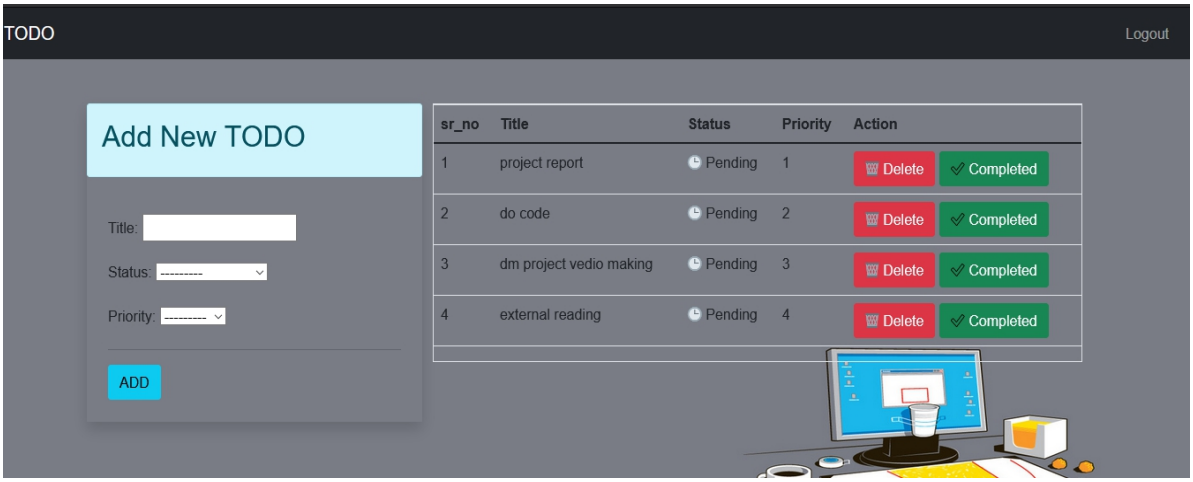
Index page (when no to-do are added)



Add to-do

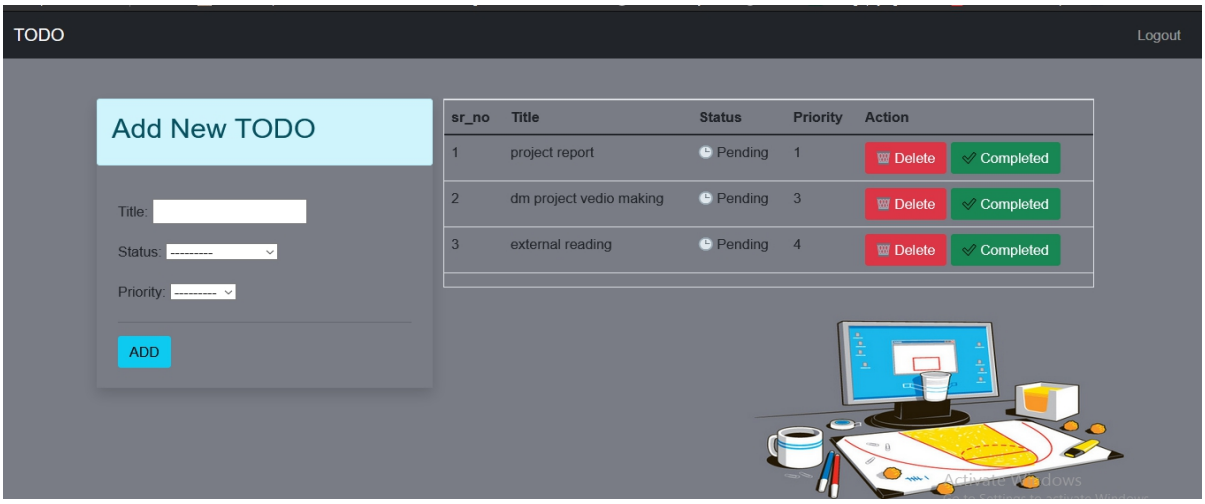


To-do added (title : do code)



Change status

Change =(title : do code) ~~completed~~ instead of pending



Delete to-do (title : do code ->deleted)

7. Conclusion

We have successfully implemented the multiuser to-do app. User will add the new to-do & edit to-do status. User can show all the to-dos and can also delete to-dos record .super user also can delete end user record (if needed).superuser can also edit status of any end user and they can also add new todo of any user and delete any to-do of any user . we also testing this app by using selenium automated testing tool . and its work perfectly fine .

9. Limitation and Future Extension

Limitations :

End user can't update task's title if already exists .but once end user delete task and then 2nd time he/she can add same to do in updated title name

Future extensions :

We can also add here timings of to-do like when it should be completed and based on this we can also send mail to similar username whose work is still pending . user can also search to-do title here . they can also update to-do title here.

10. Bibliography

References/resources used for developing project:

- https://getbootstrap.com/docs/4.0
- www.w3schools.com
- www.google.com
- www.stackoverflow.com
- www.wikipedia.com
- www.docs.djangoproject.com