

MINI PROJECT

On

REAL TIME-MESSENGER

Submitted by: -

Prashant Asthana
(181500487)

Tanishq Tripathi
(181500751)

Shriyanshi Baranwal
(181500691)

Department of Computer Engineering & Applications
Institute of Engineering & Technology

Under Supervision of:
Mr. Amir Khan



GLA University
Mathura- 281406, INDIA
2020

TOPIC

REAL TIME-MESSENGER

Contents

Abstract	4
Certificate	5
Synopsis	8
Acknowledgment	11
1. Introduction	12
1.1 Objective	12
2. Front end technologies used	14
2.1 What is Django.....	14
2.2 What are WebSockets.....	18
2.3 What is Python.....	21
2.4 Web Development.....	25
3. Languages Used	28
3.1 Python.....	28
3.2 JavaScript.....	29
3.3 HTML.....	30
3.4 CSS.....	31
4. Prerequisites Software	32
4.1 Django.....	32
4.2 Python.....	32
4.3 WebSocket.....	32
5. Screenshots	33
6. Running Project Screenshots	47
7. Reference	51

Abstract

Lately we see a constant switch towards real-time applications. With the wide support for WebSockets in recent browsers, more and more frameworks are giving us the ability to use them. Django, which was primarily a request/response-based framework, is doing the switch with Django Channels. Django channels is a lot more than just support for WebSockets, it's a complete architectural change for Django and - in my honest opinion - a great move towards the new era of frameworks.

We are creating a platform where a particular person can send a mail to different ids at a same time without having to check the database of their employees. This saves a lot of time. This project saves a lot of storage as there would be no use of creating any other database for sending mail to a large number of people. It creates a database of the details entered. This data can be updated any time, very easily and be securely saved.

TRAINING CERTIFICATION

Prashant Asthana



Tanishq Tripathi



Shriyanshi Baranwal



SYNOPSIS

Introduction

Nowadays, when all sorts of messenger have become extremely popular, when every second large company has launched or developed its own instant messenger, when an increase of smiles and change in the text size is considered as innovation, in the era of Gmail, yahoo, Orkut, Whatsapp etc. We will use Django-channels.

Python is fast becoming a popular coding language in the world, and there are many popular frameworks that build on Python. One of them is Django and it has many functionalities and supporting libraries. For this article, we like to explore one interesting method that builds on Django to handle not only HTTP but also long running connections such as WebSockets, MQTT, etc.

Technical Details

Following are the technologies used in this project:

- Python-Django
- WebSockets
- Web Development
- Python

It uses the Django auth system to provide user accounts; users are only be able to use the chat once logged in, and this provides their username details for the messenger.

This package allows our application to interact with a user not only using HTTP 1.1 (request-response), but also using HTTP/2 and WebSocket.

WebSocket is designed for exchanging messages between the client and the web server in real time. You should consider it as an open channel between the client and the server, with the ability to subscribe to the events sent to it.

Software Specification

- Technology Implemented : JavaScript, Django, python, WebSocket
- User Interface Design : Web based Application
- Web Browser : Chrome

Hardware Requirement

- Processor : Intel CORE i3
- Operating System : Windows 10
- RAM : 4 GB
- Hardware System : Computer System
- Hard Disk : 64 GB

ACKNOWLEDGEMENT

Primarily we would like to thank God for being able to complete this project. Then we would like to thank GLA University, our mentor Mr. Amir Khan, whose valuable guidance has been the ones that helped us patch this project and make it full proof success. Their suggestions and their instructions have served as the major contributions towards the completion of the project.

Then we would like to thank our parents and colleagues who helped us with their valuable suggestions and guidance which has been helpful in various phases of the completion of the project. Last but not the least we would like to thank our batchmates who helped us a lot.

INTRODUCTION

Objective

The main objective of creating this real-time messenger is:

Firstly, we are creating a platform where a particular person can send a mail to different ids at a same time without having to check the database of their employees. This saves a lot of time.

Secondly, this project saves a lot of storage as there would be no use of creating any other database for sending mail to a large number of people.

Thirdly, it creates a database of the details entered. This data can be updated any time, very easily and be securely saved.

Lately we see a constant switch towards real-time applications. With the wide support for WebSockets in recent browsers, more and more frameworks are giving us the ability to use them.

Django, which was primarily a request/response-based framework, is doing the switch with Django Channels. Django channels is a lot more than just support for WebSockets, it's a complete architectural change for Django and - in my honest opinion - a great move towards the new era of frameworks.

FRONT END TECHNOLOGY USED

WHAT IS DJANGO?



Django is a fully featured web framework for building web applications using the general-purpose Python language. Django creators call it the pragmatic framework for perfectionists with deadlines which is really true .If you have already used Django for building web apps for clients then you already know how Django makes it dead easy to build a web application from the

initial prototype to the final product .But Django was created ten years ago for satisfying the requirements for that era .Back then the web wasn't as modern and complex as today since the great portion of web apps were a bunch of static pages rendered on the server after fetching and formatting some data from a database .Developers used the popular MVC (Model View Controller) design pattern to structure their code .Now it is 2017 ,the web has dramatically changed the web is seeing or has already seen the rise of the real time apps with a lot of interactions .Users don't have to refresh the page each time to see the updates .Apps can send real time notifications and updates whenever they are available etc .This is all possible thanks to the new technologies that emerged such as

- WebRTC
- WebSockets
- HTTP 2

When Django was created, these technologies did not exist and real time wasn't a requirement so the framework was built around strict HTTP request and response cycles.

When you visit a web application ,the browser simply makes an HTTP request .On the server where your app lives Django calls the associated view ,which is simply a function that do some work such as processing ,getting data from database and formatting it then returns a

response which gets sent to the browser with the HTML content (Plus styles and JavaScript) as a payload ,the browser then renders the web pages .

In the traditional HTTP Request - HTTP Response cycle, the Django view function which gets called when a new request is made, stays only as long as the request lives and we can't hold a connection open or communicate with the browser (client) when the request dies so we can't take benefits from the new real time protocol or WebSockets.

But since Django is still a very popular framework that's still adopted by a huge base of web developers, the team behind this framework decided to support the real time web but without changing the way the framework is used so they have come up with Django channels.

Development Tools with Django support

For developing a Django project, no special tools are necessary, since the source code can be edited with any conventional text editor. Nevertheless, editors specialized on computer programming can help increase the productivity of development, e.g., with features such as syntax highlighting. Since Django is written in Python, text editors which are aware of Python syntax are beneficial in this regard.

Integrated development environments (IDE) add further functionality, such as debugging, refactoring, and unit testing. As with plain editors, IDEs with support for Python can be beneficial. Some IDEs that are specialized on Python additionally have integrated support for Django projects, so that using such an IDE when developing a Django project can help further increase productivity.

What are WebSockets?



WebSockets are an awesome bit of technology which enable us to do cool things such as perform real time communication between both a client and a server. They allow you to perform full-duplex communication over a single TCP connection and remove the need for clients to constantly poll API endpoints for updates or new content. Clients can create a single connection to a WebSocket server and simply listen for new events or messages from the server.

The main advantage this gives us is it reduces the amount of load on a network and can be more efficient for propagating information to huge numbers of clients.

Say for instance you have a real-time trading system that tracks stock market prices, you also have hundreds of

clients subscribed to this system. If we used the traditional method of constantly polling a REST API for new stock information every second then this would amount to thousands of network requests a minute from all of our clients. By using WebSockets we can maintain a single TCP connection for all of our clients and simply send any stock updates over said TCP connection whenever we want to update our clients.

Description of Web Socket Protocol

This protocol defines a full duplex communication from the ground up. Web sockets take a step forward in bringing desktop rich functionalities to the web browsers. It represents an evolution, which was waited for a long time in client/server web technology.

The main features of web sockets are as follows –

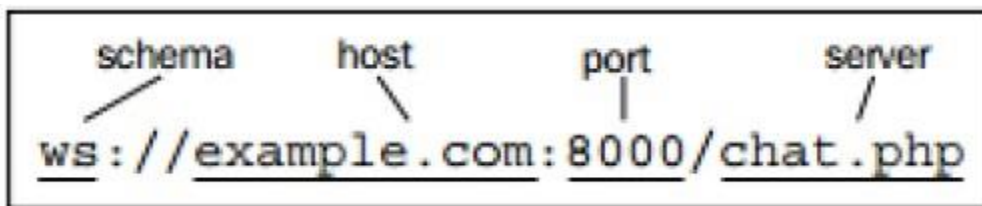
- Web socket protocol is being standardized, which means real time communication between web servers and clients is possible with the help of this protocol.
- Web sockets are transforming to cross platform standard for real time communication between a client and the server.
- This standard enables new kind of the applications. Businesses for real time web application can speed up with the help of this technology.

- The biggest advantage of Web Socket is it provides a two-way communication (full duplex) over a single TCP connection.

URL

HTTP has its own set of schemas such as http and https. Web socket protocol also has similar schema defined in its URL pattern.

The following image shows the Web Socket URL in tokens.



Browser Support

The latest specification of Web Socket protocol is defined as **RFC 6455** – a proposed standard.

RFC 6455 is supported by various browsers like Internet Explorer, Mozilla Firefox, Google Chrome, Safari, and Opera.

WHAT IS PYTHON?



In technical terms, Python is an object-oriented, high-level programming language with integrated dynamic semantics primarily for web and app development. It is extremely attractive in the field of Rapid Application Development because it offers dynamic typing and dynamic binding options.


Python is relatively simple, so it's easy to learn since it requires a unique syntax that focuses on readability. Developers can read and translate Python code much easier than other languages. In turn, this reduces the cost of program maintenance and development because it allows teams to work collaboratively without significant language and experience barriers.

Additionally, Python supports the use of modules and packages, which means that programs can be designed in a modular style and code can be reused across a variety of projects. Once you've developed a module or package you need, it can be scaled for use in other projects, and it's easy to import or export these modules.


One of the most promising benefits of Python is that both the standard library and the interpreter are available free of charge, in both binary and source form. There is no exclusivity either, as Python and all the necessary tools are available on all major platforms. Therefore, it is an enticing option for developers who don't want to worry about paying high development costs.

If this description of Python over your head, don't worry. You'll understand it soon enough. What you need to take away from this section is that Python is a programming language used to develop software on the web and in app form, including mobile. It's relatively easy to learn, and the necessary tools are available to all free of charge.

That makes Python accessible to almost anyone. If you have the time to learn, you can create some amazing things with the language.



Python Benefits:

- Back-end and front-end development
- cross-platform language
- open-source
- Strong community base
- Plethora of tools
- Fewer and simple lines of codes 

Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add

a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

WEB DEVELOPMENT



Web development is the work involved in developing a Web site for the Internet (World Wide Web) or an intranet (a private network).^[1] Web development can range from developing a simple single static page of plain text to complex Web-based Internet applications (Web apps), electronic businesses, and social network services. A more comprehensive list of tasks to which Web development commonly refers, may include Web engineering, Web design, Web content development, client liaison, client-side/server-side scripting, Web server and network security configuration, and e-commerce development.

Among Web professionals, "Web development" usually refers to the main non-design aspects of building Web sites: writing mark-up and coding.^[2] Web development

may use content management systems (CMS) to make content changes easier and available with basic technical skills.

For larger organizations and businesses, Web development teams can consist of hundreds of people (Web developers) and follow standard methods like Agile methodologies while developing Web sites. Smaller organizations may only require a single permanent or contracting developer, or secondary assignment to related job positions such as a graphic designer or information systems technician. Web development may be a collaborative effort between departments rather than the domain of a designated department. There are three kinds of Web developer specialization: front-end developer, back-end developer, and full-stack developer. Front-end developers are responsible for behaviour and visuals that run in the user browser, while back-end developers deal with the servers.

Most web devs use Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript to develop websites.

HTML defines the basic framework of a website – the foundation upon which everything else is built upon. It forms the blocks that define a page's layout, format, and critical components. Although it is theoretically possible to code a website on HTML only, it will be just a barebone site with no functions unless it's enriched with

CSS and JavaScript. Also, even simple style modifications such as changing the colour of a button require a lot of coding to be executed using HTML only.

CSS is used to style the content of a website using a small set of files that are kept across the entire site. This way, whenever a change must be applied to say, consistently change the colour of all the buttons found in every page of the website, a web dev needs to edit only a single file in CSS.

The JavaScript programming language is used to take care of the interactivity of many unique website elements. It can be used to create effects that alter the appearance of icons and drop-down menus, add animations, games, and other interactive elements.

Web developers are usually divided into front-end devs, back-end devs, and full-stack devs. A front-end dev takes care of all the visual aspects of the website (layout, navigation bar, etc.), its interactivity, and binds together all its elements.

Back-end devs take care of less visible tasks that ensure the website runs smoothly, such as managing the website's hosting services, database, and applications. Back-end devs might need to engineer solution to server issues by using additional server-side languages such as Python, Ruby, Java, and PHP.

Full-stack devs are developers able to do both front-end and back-end tasks at the same time.

LANGUAGES USED

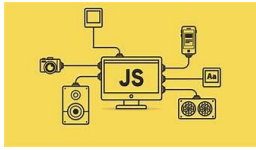
PYTHON: -



Python is an interpreted, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

JAVASCRIPT: -



JavaScript, often abbreviated as JS, is a programming language that conforms to the ECMAScript specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions. Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. JavaScript enables interactive web pages and is an essential part of web applications. The vast majority of websites use it for client-side page behaviour, and all major web browsers have a dedicated JavaScript engine to execute it.

HTML: -



Hypertext Markup Language (HTML) is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.

Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items.

CSS: -



Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML.^[1] CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.^[2]

CSS is designed to enable the separation of presentation and content, including layout, colours, and fonts.^[3] This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate .CSS file which reduces complexity and repetition in the structural content as well as enabling the .CSS file to be cached to improve the page load speed between the pages that share the file and its formatting.

PREREQUISITES

SOFTWARE

Django: -

To get the Django server running:

pip install Django

Run Django's development server (starts on localhost:8000):

python manage.py runserver

Python: -

Python3 should be installed in the system.

Libraries involved:

- django 2.0
- django-notifs
- uWSGI

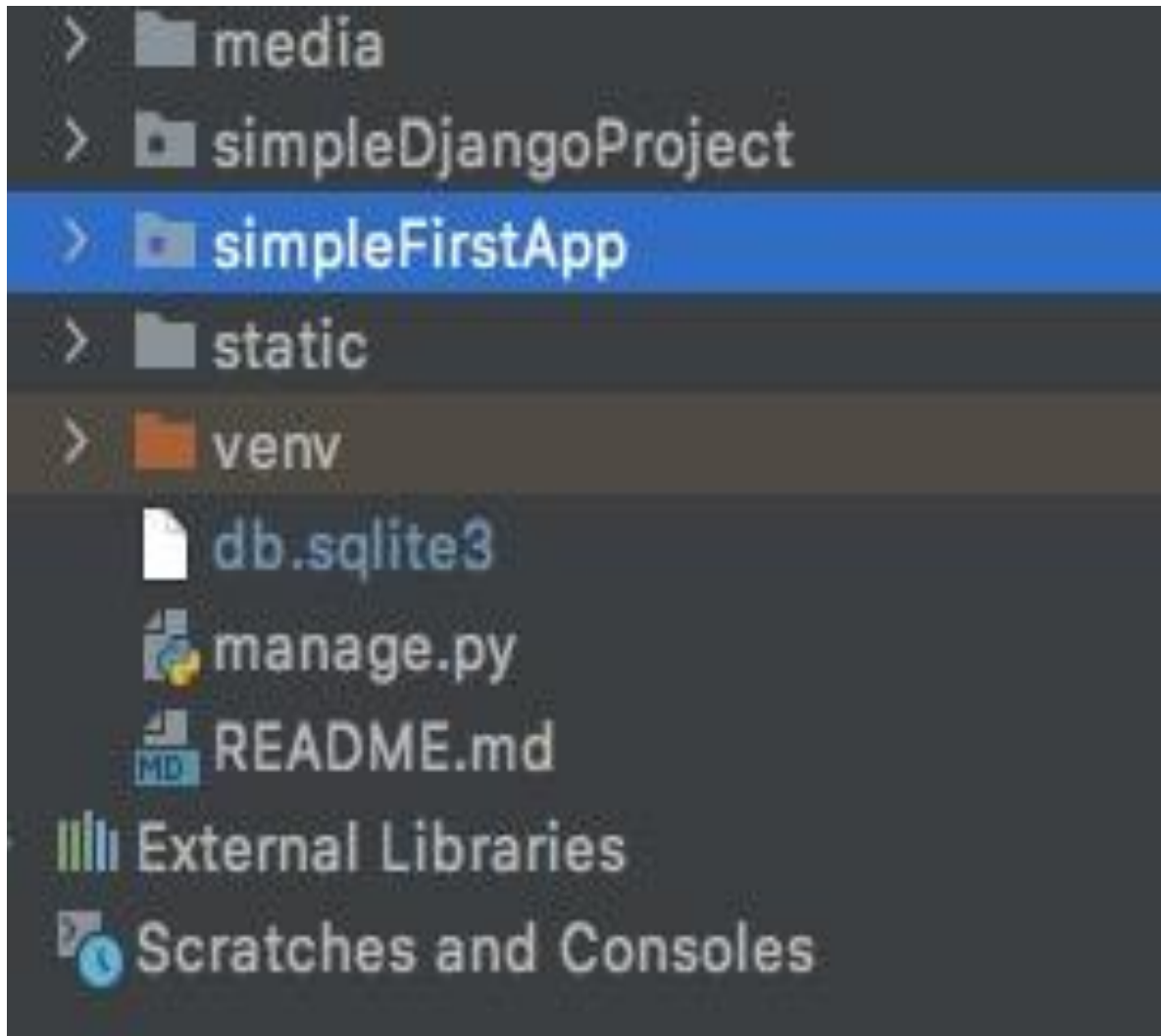
WebSocket: -

Python WebSockets to be installed:

- pywebsocket
- Tornado

. SCREENSHOTS

Files:



manage.py:

```
1  #!/usr/bin/env python
2  """Django's command-line utility for administrative tasks."""
3  import ...
4
5
6
7  def main():
8      os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'simpleDjangoProject.settings')
9      try:
10         from django.core.management import execute_from_command_line
11     except ImportError as exc:
12         raise ImportError(
13             "Couldn't import Django. Are you sure it's installed and "
14             "available on your PYTHONPATH environment variable? Did you "
15             "forget to activate a virtual environment?"
16         ) from exc
17     execute_from_command_line(sys.argv)
18
19
20 if __name__ == '__main__':
21     main()
22
```

login page:

1.

```
{% block bodydata %}
<div class="container">

    <div class="row block-area">
        <div class="col-lg-3">

        </div>
        <div class="col-lg-6 form-area">
            <form action="/do_loginn_user" method="post">
                {% csrf_token %}
                <div class="head-title"><h3 class="text-center" style="background-color: #f0f0f0;">Login User</h3></div>
                <div class="form-group">
                    <label class="control-label">Username : </label>
                    <input type="text" name="username" class="form-control">
                </div>
                <div class="form-group">
                    <label class="control-label">Password : </label>
                    <input type="password" name="password" class="form-control">
                </div>
                <div class="form-group">
                    <input type="submit" value="Submit" class="btn btn-block btn-success">
                </div>
            </form>
        </div>
        <div class="col-lg-3">

        </div>
    </div>
    <div class="row">
        <div class="col-lg-12">
            <div class="form-group">
                {% if messages %}
                    {% for message in messages %}
                        {% if message.tags == 'success' %}
                            <div class="alert alert-success">{{ message }}</div>
                        {% elif message.tags == 'error' %}
                            <div class="alert alert-danger">{{ message }}</div>
                        {% else %}
                            <div class="alert alert-info">{{ message }}</div>
                        {% endif %}
                    {% endfor %}
                {% endif %}
            </div>
        </div>
    </div>
</div>
{% endblock %}
```


2.

```
{% extends 'base_template.html' %}
```

```
{% block title %}
```

```
Add Data Page
```

```
{% endblock title %}
```

```
{% block custom_css %}
```

```
<style>
```

```
.bg_blue{
```

```
padding:10px;
```

```
border-radius:10px;
```

```
box-shadow:3px 3px 3px grey;
```

```
background:dodgerblue;
```

```
color:#fff;
```

```
}
```

```
.custom-block{
```

```
padding:15px;
```

```
border:1px solid dodgerblue;
```

```
}
```

```
.block-area{
```

```
margin-top:30px
```

```
}
```

```
.form-area{
```

```
border:1px solid dodgerblue;
```

```
padding:15px;
```

```
}
```

```
.head-title{
```

```
padding:5px;
```

```
background:dodgerblue;
```

```
color:#fff;
```

```
margin-bottom:10px;
```

```
}
```

```
</style>
```

```
{% endblock custom_css %}
```

```
{% block bodydata %}
```

3.

```
        </div>
    </form>
</div>
<div class="col-lg-3">

</div>
</div>
<div class="row">
    <div class="col-lg-12">
        <div class="form-group">
            {% if messages %}
                {% for message in messages %}
                    {% if message.tags == 'success' %}
                        <div class="alert alert-success">{{ message }}</div>
                    {% elif message.tags == 'error' %}
                        <div class="alert alert-danger">{{ message }}</div>
                    {% endif %}
                {% endfor %}
            {% endif %}
        </div>
    </div>
</div>
</div>
{% endblock bodydata %}

{% block custom_js %}
{% endblock custom_js %}
```

urls.py:

1.

```
urlpatterns = [

    # Simple Text Page
    path('', views.IndexPageController, name="index_page"),
    path('firstPage', views.FirstPageController, name="first_page"),

    #loading Html File Pages
    path('htmlPages', views.HtmlPageController, name='html_page'),

    #passing data to html templates
    path('htmlPagesWithData', views.HtmlPageControllerWithData, name='html_page_data'),

    #passing data from url to controller
    path('htmlWithDataPass/<str:url_data>', views.PassingDatatoController, name='html_data_pass'),

    path('addData', views.addData, name="add_data"),

    path('add_student', views.add_student, name="add_student"),

    path('add_teacher', views.add_teacher, name="add_teacher"),

    path('show_all_data', views.show_all_data, name="show_all_data"),

    path('update_student/<str:student_id>', views.update_student, name="update_student"),

    path('edit_student', views.edit_student, name="edit_student"),

    path('delete_student/<str:student_id>', views.delete_student, name="delete_student"),

    path('register_user/', views.RegisterUser, name="register_user"),

    path('login_user/', views.LoginUser, name="login_user"),

    path('save_user', views.SaveUser, name="save_user"),
```

2.

```
path('do_loginn_user', views.DoLoginUser, name="do_login_user"),

path('homePage/', views.HomePage, name="homepage"),

path('logout/', views.LogoutUser, name="logout"),

path('admin/', admin.site.urls),

path('teststudent', views.testStudent, name='test_student'),

path('getSubjects', apiViews.getSubjects, name='subjects'),

path('send_mail_plain', views.SendPlainEmail, name='plain_email'),

path('send_mail_plain_with_stored_file', views.send_mail_plain_with_stored_file, name='plain_email'),

path('send_mail_plain_with_file', views.send_mail_plain_with_file, name='plain_email'),
```

models.py

1.

```
is_active=models.BooleanField(default=True)
objects=models.Manager()

class StudentSubjects(models.Model):
    id=models.AutoField(primary_key=True)
    subject_id=models.ForeignKey(Subjects,on_delete=models.CASCADE)
    student_id=models.ForeignKey(Students,on_delete=models.CASCADE)
    created_at=models.DateTimeField(auto_now_add=True)
    objects=models.Manager()

class MultiStepFormModel(models.Model):
    id=models.AutoField(primary_key=True)
    fname=models.CharField(max_length=255)
    lname=models.CharField(max_length=255)
    phone=models.CharField(max_length=255)
    twitter=models.CharField(max_length=255)
    facebook=models.CharField(max_length=255)
    gplus=models.CharField(max_length=255)
    email=models.CharField(max_length=255)
    password=models.CharField(max_length=255)
    objects=models.Manager()

class Products(models.Model):
    id=models.AutoField(primary_key=True)
    name=models.CharField(max_length=255)
    desc=models.TextField()

class ProductImages(models.Model):
    id=models.AutoField(primary_key=True)
    product_id=models.ForeignKey(Products,on_delete=models.CASCADE)
    image=models.FileField(max_length=255)
```


2.

```
from django.db import models

# Create your models here.

class Teachers(models.Model):
    id=models.AutoField(primary_key=True)
    name=models.CharField(max_length=255)
    email=models.CharField(max_length=255)
    department=models.CharField(max_length=255)
    created_at=models.DateTimeField(auto_now_add=True)
    is_active=models.BooleanField(default=True)
    objects=models.Manager()

class Courses(models.Model):
    id=models.AutoField(primary_key=True)
    course_name=models.CharField(max_length=255)
    created_at=models.DateTimeField(auto_now_add=True)
    objects=models.Manager()

class Subjects(models.Model):
    id=models.AutoField(primary_key=True)
    course_id=models.ForeignKey(Courses,on_delete=models.CASCADE)
    subject_name=models.CharField(max_length=255)
    created_at=models.DateTimeField(auto_now_add=True)
    objects=models.Manager()

class Students(models.Model):
    id=models.AutoField(primary_key=True)
    name=models.CharField(max_length=255)
    email=models.CharField(max_length=255)
    standard=models.CharField(max_length=255)
    hobbies=models.CharField(max_length=255)
    roll_no=models.CharField(max_length=255)
```

students_edit.html

1.

```
<div class="form-group">
  <label class="control-label">Email : </label>
  <input type="text" name="email" class="form-control" value="{{ student.email }}">
</div>
<div class="form-group">
  <label class="control-label">Standard : </label>
  <input type="text" name="standard" class="form-control" value="{{ student.standard }}">
</div>
<div class="form-group">
  <label class="control-label">Hobbies : </label>
  <input type="text" name="hobbies" class="form-control" value="{{ student.hobbies }}">
</div>
<div class="form-group">
  <label class="control-label">Roll No : </label>
  <input type="text" name="roll_no" class="form-control" value="{{ student.roll_no }}">
</div>
<div class="form-group">
  <label class="control-label">Bio : </label>
  <input type="text" name="bio" class="form-control" value="{{ student.bio }}">
</div>
<div class="form-group">
  <label class="control-label">Profile : </label>
  
  <input type="file" name="profile" class="form-control" >
</div>
<div class="form-group">
  <input type="hidden" value="{{ student.id }}" name="id">
  <input type="submit" name="submit" class="btn btn-block btn-success" value="Edit Students">
</div>
</form>
</div>
</div>
<div class="row">
  <div class="col-lg-12">
```


2.

```
    </div>
</div>
</div>
<div class="row">
    <div class="col-lg-12">
        <div class="form-group">
            {% if messages %}
                {% for message in messages %}
                    {% if message.tags == 'success' %}
                        <div class="alert alert-success">{{ message }}</div>
                    {% elif message.tags == 'error' %}
                        <div class="alert alert-danger">{{ message }}</div>
                    {% endif %}
                {% endfor %}
            {% endif %}
        </div>
    </div>
</div>
</div>
{% endblock bodydata %}

{% block custom_js %}
{% endblock custom_js %}
```

3.

```
{% extends 'base_template.html' %}

{% block title %}
Add Data Page
{% endblock title %}

{% block custom_css %}
<style>
.bg_blue{
padding:10px;
border-radius:10px;
box-shadow:3px 3px 3px grey;
background:dodgerblue;
color:#fff;
}
.custom-block{
padding:15px;
border:1px solid dodgerblue;
}
</style>
{% endblock custom_css %}

{% block bodydata %}
<div class="container">
  <div class="row">
    <div class="col-lg-6 custom-block">
      <div class="row">
        <div class="col-lg-12">
          <h2 class="bg_blue">Edit Students</h2>
        </div>
        <div class="col-lg-12">
          <form action="/edit_student" method="post" enctype="multipart/form-data">
            {% csrf_token %}
            <div class="form-group">
              <label class="control-label">Name : </label>
              <input type="text" name="name" class="form-control" value="{{ student.name }}">
            </div>
          </form>
        </div>
      </div>
    </div>
  </div>
</div>
```

settings.py

1.

```
import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
from django.core.mail.backends import smtp

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'geby1jb%*pxhswi2qp3o1!h*#)@_wpzjd*b^p-wt=2i3p%e-'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'simpleFirstApp',
    'channels',
]
```

1

2.

```
LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

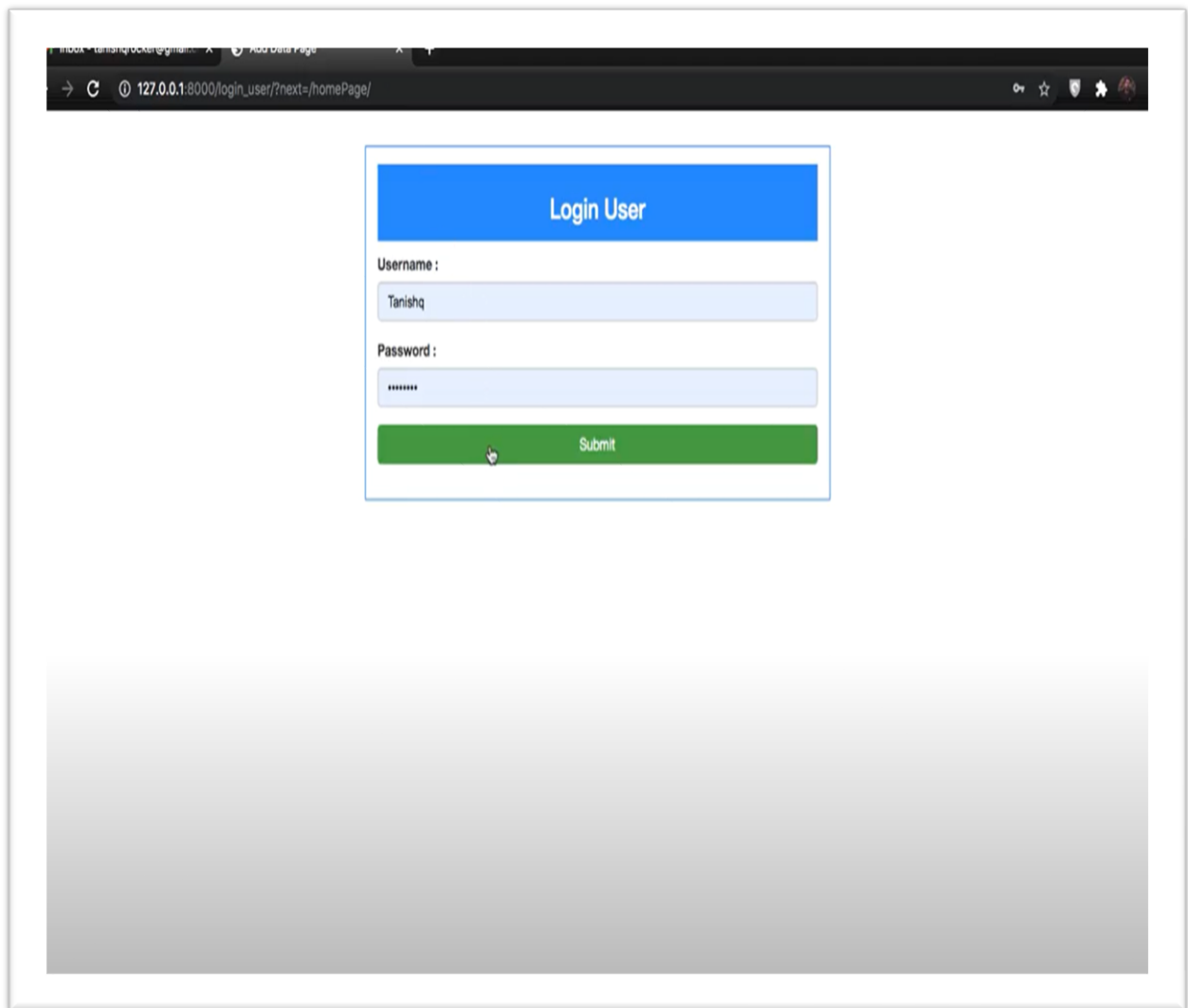

EMAIL_HOST="smtp.gmail.com"
EMAIL_PORT=587
EMAIL_HOST_USER="tanishqrocker@gmail.com"
EMAIL_HOST_PASSWORD='1242910030'
EMAIL_USE_TLS=True


ASGI_APPLICATION="simpleDjangoProject.routing.application"
CHANNEL_LAYERS={
    "default":{
        "BACKEND":"channels.layers.InMemoryChannelLayer"
    }
}
```

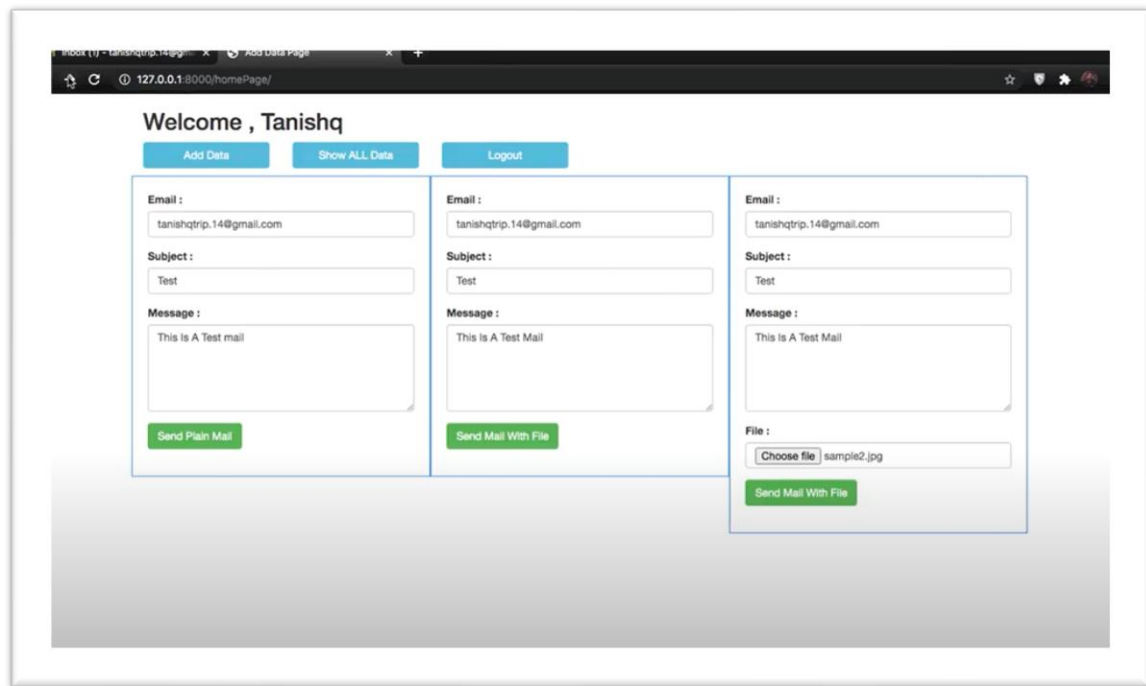
RUNNING PROJECT

SCREENSHOTS

Login page:



Sending mail:

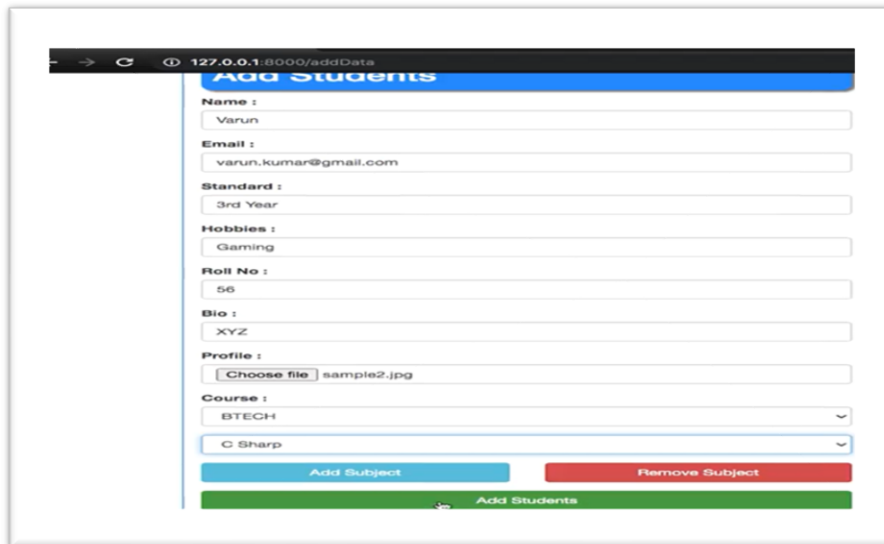


The screenshot shows a web browser window with the address bar displaying "127.0.0.1:8000/homePage/". The page title is "Welcome , Tanishq". Below the title, there are three buttons: "Add Data", "Show ALL Data", and "Logout". The main content area contains three identical email sending forms arranged horizontally. Each form has the following fields and buttons:

- Email :** A text input field containing "tanishqtrip.14@gmail.com".
- Subject :** A text input field containing "Test".
- Message :** A text area containing "This is A Test mail".
- Buttons:** A green button labeled "Send Plain Mail" (for the first form) and a green button labeled "Send Mail With File" (for the second and third forms).
- File Upload:** A "File :" label, a "Choose file" button, and a text input field containing "sample2.jpg".

Add data for student:

■

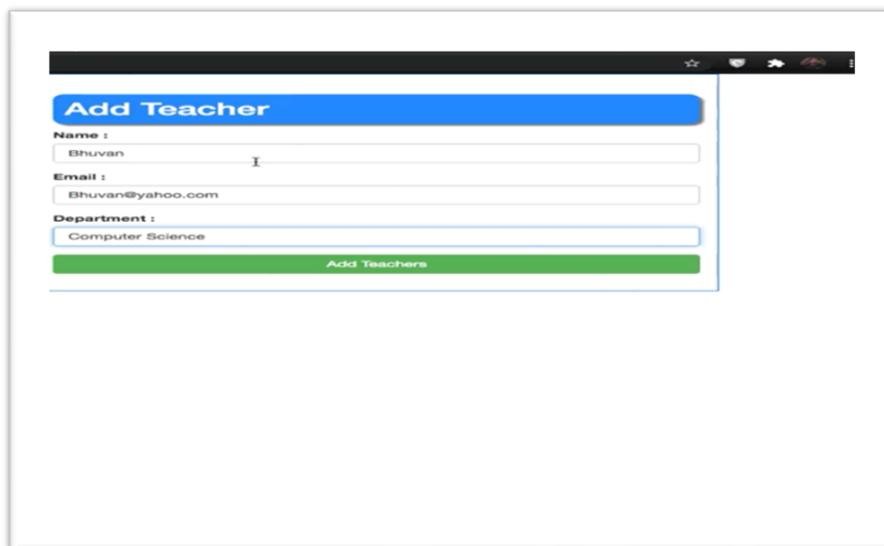


A screenshot of a web browser displaying a form titled "Add Students". The browser's address bar shows "127.0.0.1:8000/addData". The form contains the following fields and controls:

- Name :** Text input with "Varun" entered.
- Email :** Text input with "varun.kumar@gmail.com" entered.
- Standard :** Text input with "3rd Year" entered.
- Hobbies :** Text input with "Gaming" entered.
- Roll No :** Text input with "56" entered.
- Bio :** Text input with "XYZ" entered.
- Profile :** File upload section with a "Choose file" button and "sample2.jpg" listed.
- Course :** Dropdown menu with "BTECH" selected.
- Subject :** Dropdown menu with "C Sharp" selected.
- Buttons:** "Add Subject" (blue), "Remove Subject" (red), and "Add Students" (green) at the bottom.

Add data for teachers:

■

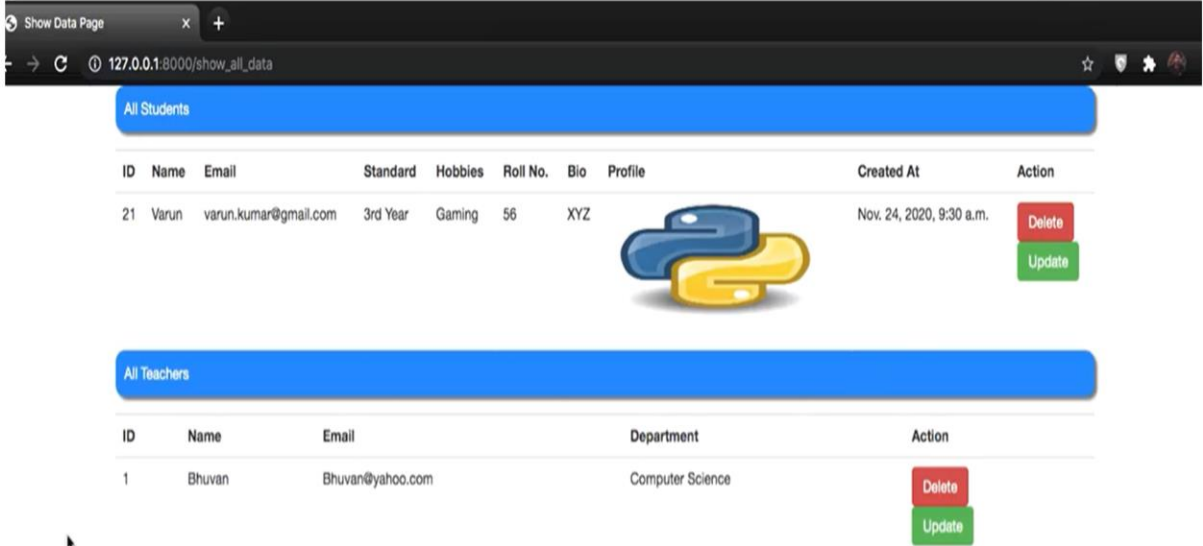


A screenshot of a web browser displaying a form titled "Add Teacher". The browser's address bar shows "127.0.0.1:8000/addTeacher". The form contains the following fields and controls:


- Name :** Text input with "Bhuvan" entered.
- Email :** Text input with "Bhuvan@yahoo.com" entered.
- Department :** Text input with "Computer Science" entered.
- Button:** "Add Teachers" (green) at the bottom.

Show data:

For student and teacher



The screenshot displays a web application interface with two main sections: 'All Students' and 'All Teachers'. The 'All Students' section features a table with columns: ID, Name, Email, Standard, Hobbies, Roll No., Bio, Profile, Created At, and Action. A single student record is shown with ID 21, Name Varun, Email varun.kumar@gmail.com, Standard 3rd Year, Hobbies Gaming, Roll No. 56, Bio XYZ, and a profile picture of a blue and yellow Python logo. The 'Created At' timestamp is Nov. 24, 2020, 9:30 a.m., and the Action column contains 'Delete' and 'Update' buttons. The 'All Teachers' section has a table with columns: ID, Name, Email, Department, and Action. One teacher record is listed with ID 1, Name Bhuvan, Email Bhuvan@yahoo.com, and Department Computer Science. It also includes 'Delete' and 'Update' buttons. The browser's address bar shows the URL 127.0.0.1:8000/show_all_data.

ID	Name	Email	Standard	Hobbies	Roll No.	Bio	Profile	Created At	Action
21	Varun	varun.kumar@gmail.com	3rd Year	Gaming	56	XYZ		Nov. 24, 2020, 9:30 a.m.	<button>Delete</button> <button>Update</button>

ID	Name	Email	Department	Action
1	Bhuvan	Bhuvan@yahoo.com	Computer Science	<button>Delete</button> <button>Update</button>

REFERENCES

1. <https://www.djangoproject.com/>
2. <https://www.wikipedia.org/>
3. <https://channels.readthedocs.io/en/stable/>
4. <https://medium.com/labcodes/introduction-to-django-channels-d1047e56f218>

End of Project



THANK YOU!