MANIPAL INSTITUTE OF TECHNOLOGY

MANIPAL

A Constituent Institution of Manipal University

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that Ms./Mr. ……...……

Reg. No. …..…… Section: …… Roll No: ……... has satisfactorily completed the lab exercises prescribed for INTERNET TECHNOLOGY LAB [CSE XXXX] of Third Year B. Tech. (Computer Science and Engineering-Artificial Intelligence and Machine Learning) Degree at MIT, Manipal, in the academic year 2023-2024.

Date: …….....

Signature
Faculty in Charge

CONTENTS

LAB NO.	TITLE	PAGE NO.	REMARKS
	Course Objectives and Outcomes	i	
	Evaluation plan	i	
	Instructions to the Students	ii	
1	HTML5	1 – 7	
2	CSS and Bootstrap	8 – 12	
3	Javascript	13 – 31	
4	JQuery	32 – 41	
5	Developing Web Application using Django – Part I	42 – 53	
6	Developing Web Application using Django – Part II	54 – 55	
7	Mini-project-Phase-1	73-73	
8	Form Processing using Django – Part I	74 – 89	
9	Form Processing using Django – Part II	90 – 90	
10	Databases – Part I	90 – 98	
11	Databases – Part II	99 –101	
12	Mini-project-Phase-1	102-102	
13	References	103	

Course Objectives

- Acquire in-depth understanding of web application architecture.
- Understand techniques to improve user experience in web applications.
- Gain knowledge about how to interact with database.

Course Outcomes

At the end of this course, students will be able to

- Develop a basic website using a modern web development tool.
- Design websites with better look and feel.
- Create real-world web applications that interacts with database.

Evaluation plan

- Internal Assessment Marks : 60%

  ➢ Continuous Evaluation : 42%

  Continuous evaluation component (for each evaluation):4 marks

  The assessment will depend on punctuality, program execution, maintaining the observation note and answering the questions in viva voce. Mid-term Evaluation: 10 marks

  ➢ Project Evaluation : 18%

- End semester assessment of two hour duration: 40 %
- Total (Internal assessment + End semester assessment): 100 marks

INSTRUCTIONS TO THE STUDENTS

Pre- Lab Session Instructions

1. Students should carry the Lab Manual Book and the required stationery to every lab session
2. Be in time and follow the institution dress code
3. Must Sign in the log register provided
4. Make sure to occupy the allotted seat and answer the attendance
5. Adhere to the rules and maintain the decorum
6. Students must come prepared for the lab in advance

In- Lab Session Instructions

- Follow the instructions on the allotted exercises
- Show the program and results to the instructors on completion of experiments
- On receiving approval from the instructor, copy the program and results in the Lab record
- Prescribed textbooks and class notes can be kept ready for reference if required

General Instructions for the exercises in Lab

- Implement the given exercise individually and not in a group.
- The programs should meet the following criteria:
  - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs. ○ Observation book should be complete with program, proper input output clearly showing the parallel execution in each process.
- Plagiarism (copying from others) is strictly prohibited and would invite severe penalty in evaluation.
- The exercises for each week are divided under three sets:
  - Solved example
  - Lab exercises - to be completed during lab hours
  - Additional Exercises - to be completed outside the lab or in the lab to enhance the skill
- In case a student misses a lab class, he/ she must ensure that the experiment is completed during the repetition lab with the permission of the faculty concerned but credit will be given only to one day's experiment(s).

- Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and / or combinations of the questions.
- A sample note preparation is given as a model for observation.

THE STUDENTS SHOULD NOT

- Bring mobile phones or any other electronic gadgets to the lab.
- Go out of the lab without permission.

Lab No.: 1 **Date:**

HTML5

Objectives:

In this lab, student will be able to

- Develop HTML5 web pages

DESCRIPTION

HTML5 – Hyper Text Markup Language Version 5

HTML5 is the 5th and newest version of HTML standard, providing new features like rich media support, interactive web applications etc.

The most interesting HTML5 elements are:

- Semantic elements like <header>, <footer>, <article> and <section>
- Attributes of form elements like number, date, time, calendar and range.
- Graphic elements like <svg> and <canvas>

Multimedia elements like <audio> and <video>

There are several Application Programming Interfaces too in HTML5 like HTML Geolocation, HTML Drag and Drop, HTML Web Workers etc.

Several elements of HTML4 have been removed in HTML5 like <big>, , <font>, <frame>, <frameset>, <strike> etc.

To indicate that your HTML content uses HTML5, simply add <!DOCTYPE html> on top of the html code.

Procedure to create and HTML document:
In notepad type the necessary code & save with the file name mentioned with .html or .htm extension.

Example:

```
<html>

<head>

<title> My First Page </title>
```

```
</head>
<body>
<h1> Hello </h1>
<h2> Welcome to Internet Technologies Lab </h2>
</body>
</html>
```

HTML5 Elements

HTML5 offers new elements for better document structure. The below given table gives a brief description on few HTML5 elements.

TAG	DESCRIPTION
<article>	Defines an article in a document
<dialog>	Defines a dialog box or window
<header>	Defines a header for a document or a section
<footer>	Defines a footer for a document or a section
<nav>	Defines navigation links
<time>	Defines a date/time
<output>	Defines the result of a calculation
<canvas>	Draw graphics, on the fly, via
	scripting(JavaScript)
<audio>	Defines sound content
<source>	Defines multiple media resources for media elements
<video>	Defines video or movie

Figure 1.1 Few HTML5 elements

New Input Types	New Input Attributes
• color • date	• autocomplete • autofocus

Figure 1.2 HTML5 input types and attributes

The above figure lists the new input types and attributes of HTML5. HTML5 Structural Tags

Tag	Description
<a>	Defines a hyperlink
 	Produces a single line break
	Specifies a division or a section in a document
<h1> to <h6>	Defines a HTML headings
<hr>	Produces a horizontal lines
	Defines a inline styleless section in a document
<nav>	Defines a navigation links

HTML5 Events

On visiting a website the user perform actions like clicking on links or image, hover over things etc. These are considered to be examples for Events.

Event handlers are developed to handle these events and this can be done using a scripting language like JavaScript, VBScript etc wherein event handlers are specified as a value of event tag attribute.

The following attributes (very few) can be used to trigger any **javascript** or **vbscript** code given as value, when there is any event occurs for any HTM5 element.

Attribute	Value	Description
Offline	script	Triggers when the document goes offline
Onchange	script	Triggers when an element changes
Onclick	script	Triggers on a mouse click
oncontextmenu	script	Triggers when a context menu is triggered
Ondrag	script	Triggers when an element is dragged
Onerror	script	Triggers when an error occur
Onfocus	script	Triggers when the window gets focus
onformchange	script	Triggers when a form changes
Onload	script	Triggers when the document loads
onmousedown	script	Triggers when a mouse button is pressed
Onpause	script	Triggers when a media data is paused
Onselect	script	Triggers when an element is selected
onsubmit	script	Triggers when a form is submitted

Figure 1.3 HTML5 event attributes

HTML5 Canvas

The HTML <canvas> element is used to draw graphics, on the fly, via JavaScript. The <canvas> element is only a container for graphics. The user must use JavaScript to actually draw the graphics. Canvas has several methods for drawing paths, boxes, circles, text, and adding images.

A canvas is a rectangular area on an HTML page. By default, a canvas has no border and no content. The markup looks like this:

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

Note: Always specify an Id attribute (to be referred to in a script), and a width and height attribute to define the size of the canvas. To add a border, use the style attribute.

Example: To draw a circle

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="100" style="border:1px solid
#d3d3d3;">
Your browser does not support the HTML5 canvas tag.</canvas> <script> var c =
document.getElementById("myCanvas");
var ctx = c.getContext("2d"); ctx.beginPath();
ctx.arc(95,50,40,0,2*Math.PI);
ctx.stroke();
</script>
 </body>
</html>
```

Output

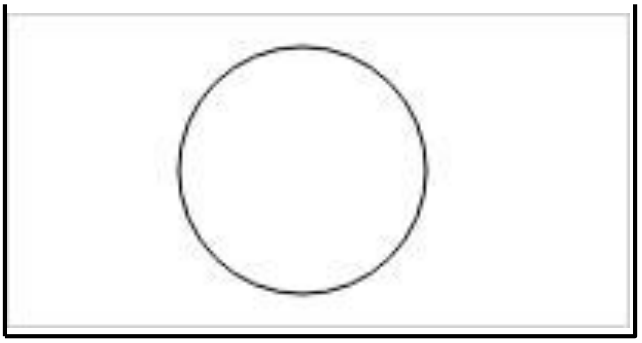


Figure 1.4

HTML5 Web Forms

The HTML <form> element defines a form that is used to collect user input. Form elements consist of various input elements like text field, check boxes, radio buttons, submit buttons etc.

<form>

</form>

Input Element

It is the most important form element and can be displayed in several ways, depending on the type attribute.

<input type = “text”> Defines a one line text input field

<input type = “radio”> Defines a radio button

<input type = “submit”> Defines a submit button

Action attribute

The action attribute defines the action to be performed when the form is submitted. Usually the form data is sent to a web page on the server when the user clicks on the submit button.

<form action = “/action_page.aspx”>
// action_page.aspx contains a server side script that handles the form data.

If the action attribute is omitted, the action is set to the current page.

Method attribute

The method attribute specifies the HTTP method (GET or POST) to be used when submitting the form data.

<form action = "/action_page.aspx" method="get">

The default method when submitting form data is GET. When GET is used the submitted form data will be visible in the page address field. Therefore it must not be used when sending sensitive information.

Use POST method if the form data contains sensitive or personal information. It does not display the submitted form data in the page address field. It has no size limitations and can be used to send large amounts of data.

II. LAB EXERCISE:

1. Write a simple poem and represent it as a web page. Give a title for the poem. Make rhyming words in the poem as bold.
2. Assume you have brought a new car. Write down the list of additional accessories you need for the car as an unordered list in HTML. Also, list the travel plans on the car as an ordered list.
3. Complete the following website name "Share your Winter Vacation Videos". The required video files are located inside the compressed folder "Week 1 Assignment Files/Media/".

Hint: Add a <video> element to a <figure> element that will either play paris.mp4, paris.webm or paris.ogv in the element. Test in different browsers

4. Create the following output in HTML

Country	Population (In Crores)	
INDIA	1998	85
	1999	90
	2000	100
USA	1998	30
	1999	35
	2000	40
UK	1998	25
	1999	30
	2000	35

Lab No:2 **Date:**

CSS and Bootstrap

Objectives:

In this lab, student will be able to

1. Develop web pages using design templates
2. Familiarize with Cascading Style Sheets
3. Learn how to use bootstrap elements ***What is Bootstrap?***

CSS – Cascading Style Sheet

CSS is a stylesheet language used for describing the presentation of a document written in a markup language ie it describes the style of a web document including the layout, design and display variations for various displays.

CSS can be applied to a web document in 3 ways.

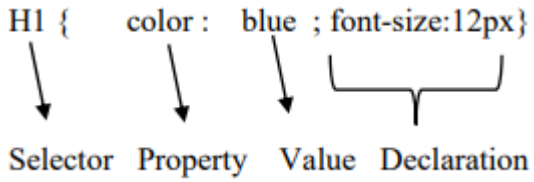
1) Inline style – Right next to the text it decorates, by using style attribute.
   <h1 style = "color : blue ;"> Hello </h1>
2) Internal style – At the top of the web page document, using
   <style> element in <head>
   <head>
   <style>
   h1 { color : blue ;}
   </style>
   </head>
3) External style – in a separate file

   <head>
   <link rel="stylesheet" href = "style.css">
   </head>
   style.css
   h1 { color : blue ;}

   The style definitions are usually saved in an external stylesheet since changing one single file can help in redesigning the entire web document with new look and feel.

CSS syntax

A CSS rule set consists of a selector and a declaration block. The selector points to the HTML element to be styled. The declaration block contains one or more declarations separated by semicolons



CSS Selectors are used to "find" or select HTML elements based on their element name, id, class, attribute etc. The element selector selects the elements based on the element name. The id selector uses the id attribute of an HTML element to select a specific element. The id of an element should be unique within a page. To select an element with a specific id, write a # character followed by the id of the element.

```
#para1{
text-align: center;
color:red; }
```

The class selector selects the elements with a specific class attribute.
To select elements with a specific class, write a period (.) character, followed by the name of the class.

```
.center {
text-align: center;
color:red;
}
```

Bootstrap

- Bootstrap is a free front-end framework for faster and easier web development
- Bootstrap includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, modals, image carousels and many other, as well as optional JavaScript plugins
- Bootstrap also gives you the ability to easily create responsive designs(automatically adjust themselves to look good on all devices)

- Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bootstrap Example</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
  <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></scr
ipt>
</head>
<body>
<div class="container">
  <h1>My First Bootstrap Page</h1>
  <p>This part is inside a .container class.</p>
  <p>The .container class provides a responsive fixed width container.</p>
  <p>Resize the browser window to see that its width (max-width) will change at
different breakpoints.</p>
</div>
</body>
</html>
```

Button Styles

Bootstrap 4 provides different styles of buttons:


```
button type="button" class="btn btn-outline-primary">Primary</button>
<button type="button" class="btn btn-outline-secondary">Secondary</button>
<button type="button" class="btn btn-outline-success">Success</button>
<button type="button" class="btn btn-outline-info">Info</button>
<button type="button" class="btn btn-outline-warning">Warning</button>
<button type="button" class="btn btn-outline-danger">Danger</button>
<button type="button" class="btn btn-outline-dark">Dark</button>
<button type="button" class="btn btn-outline-light text-dark">Light</button>
```

Lab Exercise:

1. Design the student bio-data form using button, label, textbox, radio button, table and checkbox using CSS.

2. Design a web page which shows the database oriented CRUD operation. Consider Employee data using CSS.

3. Create a web page using bootstrap as mentioned. Divide the page in to 2 parts top and bottom, then divide the bottom into 3 parts and design each top and bottom part using different input groups, input, badges, buttons and button groups. Make the design more attractive using bootstrap.

4. Design your class timetable using bootstrap table and carousel.

Additional Exercise:

1. Design an attractive ‘train ticket booking form using CSS.

2. Design an attractive ‘magazine cover page’ using different bootstrap elements.

Lab No : 3 **Date:**

JavaScript

Objectives:

In this lab, student will be able to

1. Develop responsive web pages using jquery
2. Familiarize with DOM manipulation and animations

JavaScript

JavaScript is a light weight and interpreted programming language. It is a scripting language that is commonly used for the client side web development. It makes the HTML pages more dynamic and interactive.

It can be used to put dynamic text in to HTML page (Eg: document.write("<h1>" + name + "</h1>"); // write variable text in to HTML page), react to events, validate data, create cookies etc.

Syntax

It can be implemented using JavaScript statements that are placed within the <script>….</script> HTML tags in a web page. The <script> tags, containing the JavaScript code can be placed anywhere within the web page, but normally recommended to place within the <head> tags.

The <script> tag alerts the browser program to start interpreting all the text between these tags as script. The script tag have mainly two attributes language and type, specifying the scripting language used.

To select an HTML element, JavaScript very often use the document.getElementById(id) method. The word document.write is a standard JavaScript command for writing output to a page.

<script language=”javascript” type=”text/javascript”>

document.getElementById("demo").innerHTML = "Hello JavaScript!";

</script>

II. SOLVED EXERCISE:

Develop an HTML5 program to validate the credentials with appropriate internal styling with the help of CSS and JavaScript

Program:

Login.html

```
<!DOCTYPE html>
<html>
<head>
<title> Login page </title>
<style>
body {
 background-color: lightblue;
}
h1 {
 color: white;
 text-align: center;
}
p {
 font-family: verdana;
 font-size: 20px;
}
</style>
<script language="javascript">
function check(form) /*function to check userid & password*/
{
```

```
 /*the following code checkes whether the entered userid and password are
matching*/
if(form.userid.value == "myuserid" && form.pswrd.value == "mypswrd")
 {
 window.open("https://www.google.com", "_blank");
/*opens the target page while Id & password matches*/
 }
else
{
 alert("Error Password or Username") /*displays error message*/
 }
}
</script>
</head>
<body>
<h1> Validate Credentials </h1>
<form name="login">
Username<input type="text" name="userid"/>
Password<input type="password" name="pswrd"/>
<input type="button" onclick="check(this.form)" value="Login"/>
<input type="reset" value="Cancel"/>
</form>
</body>
</html>
```

Output

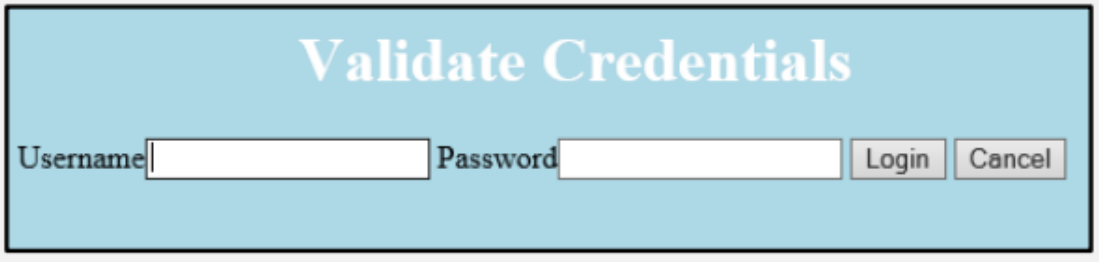


Figure 1.5

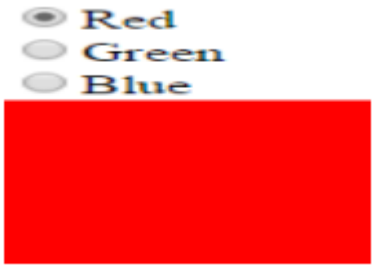
LAB EXERCISE:

1) Write a JavaScript program to perform an online quiz

2) Write a JavaScript program to Wish a user at different hours of a day. Use appropriate dialog boxes for wishing the user. Display the dynamic clock on the web page. Make use of CSS and HTML5 elements for creative and attractive design.

3) Write the java script program to display the grade [A, B,C,D] based on the marks entered by student(take the input into text boxes). Enter the marks of 4 subjects and calculate the average(using button). If the avg>90 then A, avg>80 then B, if avg>70 then C, if avg>60 then D.

4) Write the JavaScript program to show the below output.



ADDITIONAL QUESTIONS

1) Write a JavaScript code that displays text “FONT+++” with increasing font size in the interval of 200ms in RED COLOR, when the font size reaches 40pt it displays “FONT---” in BLUE color. Then the font size decreases to 5pt.
2) Write a JavaScript to design a simple calculator to perform the following operations: sum, product, difference and quotient.

Lab No : 4 **Date:**

JQuery

Objectives:

In this lab, student will be able to

1. Develop responsive web pages using jquery
2. Familiarize with DOM manipulation and animations

jQuery

jQuery is a fast and concise JavaScript library to develop web based application.

Here is the list of important core features supported by jQuery −

- *DOM manipulation* − The jQuery made it easy to select DOM elements, negotiate them and modifying their content by using cross-browser open source selector engine called Sizzle.
- *Event handling* − The jQuery offers an elegant way to capture a wide variety of events, such as a user clicking on a link, without the need to clutter the HTML code itself with event handlers.
- *AJAX Support* − The jQuery eases developing a responsive and feature rich site using AJAX technology.
- *Animations* − The jQuery comes with plenty of built-in animation effects which you can use in your websites.
- *Lightweight* − The jQuery is very lightweight library - about 19KB in size (Minified and gzipped).
- *Cross Browser Support* − The jQuery has cross-browser support, and works well in IE 6.0+, FF 2.0+, Safari 3.0+, Chrome and Opera 9.0+
- *Latest Technology* − The jQuery supports CSS3 selectors and basic XPath syntax.

You can download jQuery library from https://jquery.com/download/ on your local machine and include it in your HTML code.

Solved Example:

```
<html>
```

```
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript" src = "jquery-3.4.1.js">
    </script>
    <script type = "text/javascript" language = "javascript">
      $(document).ready(function() {
        $("div").click(function() {alert("Hello, world!");});
      });
    </script>
  </head>
  <body>
    <div id = "mydiv">
      Click on this to see a dialogue box.
    </div>
</body>
</html>
```

A good rule of thumb is to put your JavaScript programming (all your <script> tags) after any other content inside the <head> tag, but before the closing </head> tag. The *$(document).ready()* function is a built-in jQuery function that waits until the HTML for a page loads before it runs your script.

When a web browser loads an HTML file, it displays the contents of that file on the screen and also the web browser remembers the HTML tags, their attributes, and the order in which they appear in the file—this representation of the page is called the *Document Object Model*, or DOM for short.

Selector: jQuery offers a very powerful technique for selecting and working on a collection of elements—CSS selectors. The basic syntax is like this:

$('selector') use a CSS class selector like this:

$('.submenu')

```
    <script type = "text/javascript" language = "javascript">
      $(document).ready(function() {
        $("p").css("background-color", "yellow");
        $("#myid").css("background-color", "red");
      });
    </script>
  </head>
  <body>
```

```
    <div>
      <p class = "myclass">This is a paragraph.</p>
      <p id = "myid">This is second paragraph.</p>
      <p>This is third paragraph.</p>
    </div>
</body>
```

We can select tag available with the given class in the DOM. For example $('.someclass') selects all elements in the document that have a class name as some-class.

Get And Set Atrributes:

```
<script type = "text/javascript" language = "javascript">
      $(document).ready(function()      {
var title = $("p").attr("title");
         $("#divid").text(title);
         $("#myimg").attr("src", "/jquery/images/jquery.jpg");
      });
    </script>
  </head>
  <body>
    <div>
      <p title = "Bold and Brave">This is first paragraph.</p>
      <p id = "myid">This is second paragraph.</p>
      <div id = "divid"></div>
      <img id = "myimg" alt = "Sample image" />
    </div>
</body>
</html>
```

You can replace a complete DOM element with the specified HTML or DOM elements.

selector.replaceWith(content)

```
<script type = "text/javascript" language = "javascript">
      $(document).ready(function() {
        $("div").click(function () {
          $(this).replaceWith("<h1>JQuery is Great</h1>");
        });
      });
    </script>
```

Events

To make your web page interactive, you write programs that respond to events.
Mouse events: click, dblclick, mousedown, mouseup, mouseover, etc
Document/Window Events: load, resize, scroll, unload etc
Form Events: submit, reset, focus, and change

```
<script type = "text/javascript" language = "javascript">
        $(document).ready(function() {
          $('#button').click(function() {
            $(this).val("Stop that!");
              }); // end click
        });
      </script>
  </head>
  <body>
    <div id = "mydiv">
      Click on this to see a dialogue box.
        <input type="button" id="button">
    </div>
  </body>
```

- The hover(over, out) method simulates hovering (moving the mouse on, and off, an object).

```
<script type = "text/javascript" language = "javascript">
        $(document).ready(function() {
          $('div').hover(
            function () {
              $(this).css({"background-color":"red"});
            },
            function () {
              $(this).css({"background-color":"blue"});
            }
          );
        });     </script>
```

The bind() method is a more flexible way of dealing with events than jQuery's event specific functions like click() or mouseover(). It not only lets you specify an event and a

function to respond to the event, but also lets you pass additional data for the event-handling function to use.

```
$('#theElement').bind('click', function() {
// do something interesting
}); // end bind
```

- checked selector selects all checked check-boxes or radio buttons. Let us understand this with an example.

```
<html>
<head>
   <title></title>
   <script src="jquery-1.11.2.js"></script>
   <script type="text/javascript">
      $(document).ready(function () {
$('#btnSubmit').click(function () {
            var result = $('input[type="radio"]:checked');
if (result.length > 0) {
               $('#divResult').html(result.val() + " is checked");
            }
else {
               $('#divResult').html("No radio button checked");
            }
         });
      });
   </script>
</head>
<body style="font-family:Arial">
Gender :
   <input type="radio" name="gender" value="Male">Male
   <input type="radio" name="gender" value="Female">Female
    <input id="btnSubmit" type="submit" value="submit" />
      <div id="divResult">
   </div>
</body>
</html>
```

- The each() method in jQuery is used to execute a function for each matched element.

```
<html>
```

```
<head>
   <title></title>
   <script src="jquery-1.11.2.js"></script>
   <script type="text/javascript">
      $(document).ready(function () {
$('#btnSubmit').click(function () {
            var result = $('input[type="checkbox"]:checked');
if (result.length > 0) {
               var resultString = result.length + " checkboxe(s) checked<br/>";
result.each(function () {
                  resultString += $(this).val() + "<br/>";
               });
               $('#divResult').html(resultString);
            }
            else {
               $('#divResult').html("No checkbox checked");
            }
         });
      });
   </script>
</head>
<body style="font-family:Arial">
   Skills :
   <input type="checkbox" name="skills" value="JavaScript" />JavaScript
   <input type="checkbox" name="skills" value="jQuery" />jQuery
   <input type="checkbox" name="skills" value="C#" />C#
   <input type="checkbox" name="skills" value="VB" />VB
<br /><br />
   <input id="btnSubmit" type="submit" value="submit" />
   <br /><br />
   <div id="divResult">
   </div>
</body>
</html>
```


The animate() Method

The jQuery animate() method is used to create custom animations.

$(*selector*).animate({*params*},*speed*,*callback*);
The required params parameter defines the CSS properties to be animated.
The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.
The optional callback parameter is a function to be executed after the animation completes.

```
$("button").click(function(){
$("div").animate({left:'250px'});
});
```

Exercises:

1. Write a web page which contains table with 3 X 3 dimensions (fill some data) and one image. Style the rows with alternate color. Move the table and image together from right to left when button is clicked.
2. Design a calculator to perform basic arithmetic operations. Use textboxes and buttons to design web page.
3. Create a web page to design a birthday card shown below.

4. Design a webpage. The page contains:

- Dropdown list with HP, Nokia, Samsung, Motorola, Apple as items.
- Checkbox with Mobile and Laptop as items. Textbox where you enter quantity.
- There is a button with text as ‘Produce Bill’.

On Clicking Produce Bill button, alert should be displayed with total amount.

Additional Exercise:

1. Implement the bouncing ball using animate() function.
2. Write a web page which displays image and show the sliding text on the image.

Lab No:2

Lab No:5 **Date:**

Developing Web Application using Django-Part I

Objectives:

In this lab, student will be able to

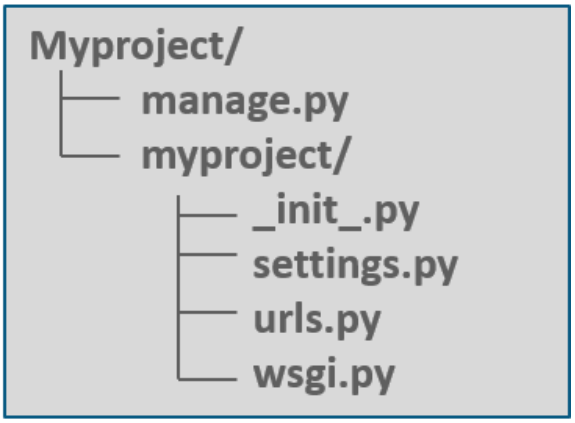
1. Understand the fundamentals of web forms creation
2. Learn to create views and templates
3. Design Django web applications using views and templates

1. Build Your First Web Application in Django

For creating a web application, first create a directory, say *PythonProject* where you would like to share your code, and then run the following command from the created directory using Windows Powershell:

```
django-admin startproject myproject
```

Myproject is the name of the project created. The following list of files are created inside the directory.



manage.py – It is a command-line utility that allows to interact with the Django project in various ways.

myproject/ – It is the actual Python package for the project. It is used to import anything, say – myproject.urls.

init.py – Init just tells the python that this is to be treated like a python package.

settings.py – This file manages all the settings of the project. **urls.py** – This is

the main controller which maps it to the website. **wsgi.py** – It serves as an entry

point for WSGI compatible web servers.

Now to create the application, type the below command in Powershell from the created project folder (i.e., *myproject).*

python manage.py startapp webapp

Now the *'webapp'* directory will have some extra things from the original *myproject*. It includes model, test which are related to the backend databases.

It is important to import your application manually inside the project settings. For that, open *myproject/settings.py* and add your app manually:

```
INSTALLED_APPS = (
                'webapp',
                'django.contrib.admin',
                'django.contrib.auth',
                'django.contrib.contenttypes',
                'django.contrib.sessions',
                'django.contrib.messages',
                'django.contrib.staticfiles',
)
```

Now to create a view, open *webapp/views.py* and put the below code in it:

```
from django.shortcuts import render from
django.http import HttpResponse def
index(request):
```

```
return HttpResponse("<H2>HEY! Welcome to Edureka! </H2>")
```

The above code creates a view which returns HttpResponse. Now this view is to be mapped to a URL. So create a new python file *"urls.py"* inside the *webapp* folder. In *webapp/urls.py* include the following code:

```
from django.conf.urls import url
```



```
from . import views
urlpatterns = [
                url(r'^$', views.index, name='index'),
]
```

In the above code, a view is referenced which will return index (defined in views.py file). The url pattern is in regular expression format where ^ stands for beginning of the string and $ stands for the end.

The next step is to point the root URLconf at the webapp.urls module. Open myproject/urls.py file and write the below code:

```
from django.conf.urls import include, url from
django.contrib import admin
urlpatterns = [
                url(r'^admin/', admin.site.urls),
                url(r'^webapp/', include('webapp.urls')),
]
```

In the above code, *webapp* and the *webapp.urls* are included. Now import *django.conf.urls.include* and insert an *include()* in the urlpatterns list. The *include()* function allows referencing other URLconfs.

Note that the regular expression doesn't have a '$' but rather a trailing slash, this means whenever Django encounters *include(),* it chops off whatever part of the URL matched up to that point and sends the remaining string to include URLconf for further processing.

To start the server, type the below command:

E:\MyFolder\myproject> python manage.py runserver

After running the server, go to **http://localhost:8000/webapp/** in your browser, and you should see the text "*HEY! Welcome to Edureka!*", which is defined in the index view(Fig 5.1).

Fig 5.1

2. Creating a View

Django's views are the information brokers of a Django application. A view sources data from your database (or an external data source or service) and delivers it to a template. The view makes decisions on what data gets delivered to the template—either by acting on input from the user, or in response to other business logic and internal processes. Each Django view performs a specific function and has an associated template.

Modify webapp/views.py and put the below code in it:

```
# \webapp\views.py


1  from django.shortcuts import render

2  from django.http import HttpResponse
```

```
3  from datetime import date
4  import calendar
5  from calendar import HTMLCalendar
6
```

```
7
8       def index(request,year,month):
9       year = int(year)
10      month = int(month)
11      if year < 1900 or year > 2099: year = date.today().year
12      month_name = calendar.month_name[month]
13      title = "MyClub Event Calendar - %s %s" % (month_name,year)
14      cal = HTMLCalendar().formatmonth(year, month)
15      return HttpResponse("<h1>%s</h1><p>%s</p>" % (title, cal))
```

Modify webapp/urls.py and put the below code in it

\webapp\urls.py

```
1 from django.urls import path, re_path
2 from . import views
3
4   urlpatterns = [
5   path(' ', views.index, name='index'),
6   re_path(r'^(?P<year>[0-9]{4})/(?P<month>0?[1-9]|1[0-2])/',      views.index,
    name='index'),
7   ]
```

Lab No:5

After running the server, go to http://localhost:8000/2019/03 in your browser, and the screen appears as shown in Fig 5.2.

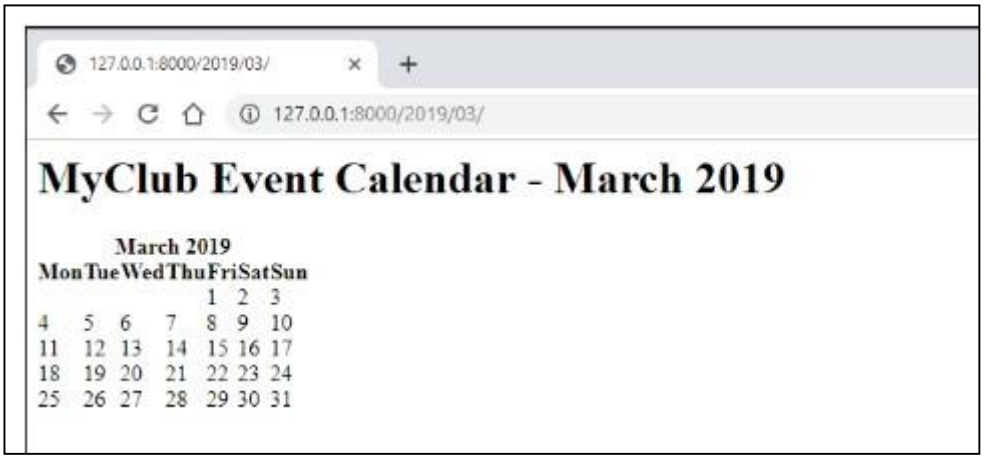


Fig 5.2

Creating a Site Template

All modern websites have a site template; a common look or branding that is duplicated across every page on the website.

The most common place for storing site template files in Django is in the website app that Django created automatically when *startproject* command is executed. Django didn't create the templates folder, so go ahead and create that folder. The folder structure should look like this:

\webapp
\templates
__init__.py
...

As the website app is not in INSTALLED_APPS, Django won't automatically look for templates in the \webapp\templates folder. So tell Django where to look by adding a path to the DIRS setting. Modify settings.py (changes in bold):

```
TEMPLATES = [{
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'webapp/templates')],

    'APP_DIRS': True,
    # ...
```

This looks complicated, but is easy to understand—os.path.join is a Python command to create a file path by joining strings together (concatenating). In this example, webapp/templates is joined to the project directory to create the full path to the templates directory, i.e., <project path>/myproject/webapp/templates.

Now that the template folder is created and the folder path is listed, Django can find the site template. Now to create a simple template, create a html file base.html:

\webapp\templates\base.html

```
1   <!doctype html>
2   <html>
3   <head>
4   <meta charset="utf-8">
5   <title>Basic Site Template</title>
6   </head>
7
8    <body>
9    <h1>{{ title }}</h1>
10   <p>{{ cal }}</p>
11   </body>
12   </html>
```

3. **Displaying a Template**

Now that the template is created, tell Django to use the new base template when displaying content on the site. This is done in views.py file. Make the following changes to the index view (changes in bold):

\webapp\views.py

```
1  from django.shortcuts import render
2  # from django.http import HttpResponse
3  from datetime import date
4  import calendar
5  from calendar import HTMLCalendar
6
7
8      def index(request, year=date.today().year, month=date.today().month):
```

```
9     year = int(year)
10    month = int(month)
11    if year < 1900 or year > 2099: year = date.today().year
12    month_name = calendar.month_name[month]
13    title = "MyClub Event Calendar - %s %s" % (month_name, year)
14    cal = HTMLCalendar().formatmonth(year, month)
15    # return HttpResponse("<h1>%s</h1><p>%s</p>" % (title, cal))
16    return render(request, 'base.html', {'title': title, 'cal': cal})
```

For the new view, replace the call to HttpResponse() with a call to render(). render() is a special Django helper function that creates a shortcut for communicating with a web browser. When Django receives a request from a browser, it finds the right view and the view returns a response to the browser.

When we wish to use a template, Django first must load the template, create a context—which is basically a dictionary of variables and associated data that is passed back to the browser—and then return a HttpResponse. Django's render() function provides a shortcut that provides all three steps in a single function.

When the original request, the template and a context is supplied directly to render(), it returns the appropriately formatted response without having to code the intermediate steps.

In the modified views.py, the original request object is returned from the browser, the name of the site template and a dictionary (the context) containing the title and cal variables from the view.

Once views.py file is modified, save it and fire up the development server. Navigate to http://127.0.0.1:8000/, to see your simple new site template.
The calendar will be rendered as plain text, not as HTML. To get Django to render the HTML correctly, turn off autoescape for the calendar code. As this is a common task, the Django developers created the autoescape tag to make life easier. Make the following changes to the base.html file (changes in bold):

\webapp\templates\base.html

```
1     <!doctype html>
2     <html>
3     <head>
4     <meta charset="utf-8">
```

```
5       <title>Basic Site Template</title>
6       </head>
7
8       <body>
9       <h1>{{ title }}</h1>
10      <p>{% autoescape off %}{{ cal }} {% endautoescape %}</p> 11     </body>
12 </html>
```

Now, when you refresh your browser, the site homepage should look like Fig 5.3.

Fig 5.3

Solved Excersise:

Develop a simple Django web application to accept two numbers from user and add them up.

A new project named MyForm is created which has the manage.py file. Inside MyForm a new app named formapp is created which contains all the application related files.

 # MyForm/settings.py

```
INSTALLED_APPS = [
   'formapp',
   'django.contrib.admin',
   'django.contrib.auth',
]
TEMPLATES = [
   {
      'BACKEND': 'django.template.backends.django.DjangoTemplates',
```

```
        'DIRS': [os.path.join(BASE_DIR,'formapp/templates')],
        'APP_DIRS': True,
},
]
```

 # MyForm/urls.py from
django.contrib import admin from
django.urls import path
from django.conf.urls import include, url

```
urlpatterns = [       path(r'^admin/',
admin.site.urls),
   url('', include('formapp.urls')),
]
```

 # formapp/urls.py from
django.conf.urls import url from
. import views

```
urlpatterns = [
url(' ', views.index, name='index'),
]
```

 # formapp/views.py from
django.shortcuts import render #
Create your views here.

```
def index(request):
          return render(request, 'basic.html')
```

 # formapp/templates/basic.html

```
<!doctype html>
 <html>
     <head>
        <meta charset="utf-8">
```

```
        <title>App to add two Nos</title>

    </head>
      <body>
       <script type="text/javascript">          function
myfunc(){          var n1=
document.getElementById("num1").value;          var n2=
document.getElementById("num2").value;
         var n3=parseInt(n1)+parseInt(n2);
         document.getElementById("para1").innerHTML="The sum of two numbers is
"+n3;
        }
        </script>
        Enter num1: <input type="text" id="num1"><br>
 Enter num2: <input type="text" id="num2"><br>
 <button onclick="myfunc()">Add</button><br\>
 <p id="para1"></p>
   </body>
</html>
```

After editing all the above files save them and fire up the development server as shown below

E:\newdir\MyForm> python manage.py runserver

Navigate to http://127.0.0.1:8000/, and find the output as shown in Fig 5.4.

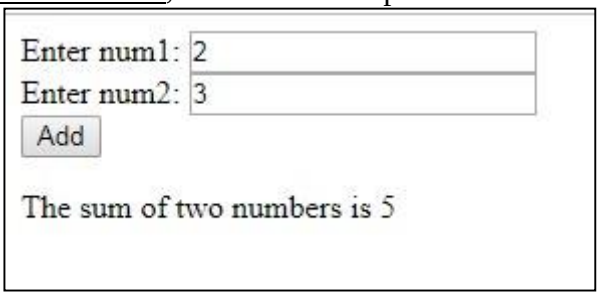


Fig 5.4

LAB EXERCISES:

1) Create a web form which allows the teacher to enter the student details like name, date of birth, address, contact number, email id and marks of English, Physics and Chemistry. After filling all the information and on clicking submit button, details should be added to a textarea displayed on the same page. Display the total percentage of marks obtained in a label.
2) Create a web form with employee ids in dropdown list. Use a textbox for accepting date of joining. Add a button named "Am I Eligible for Promotion". On clicking the button, if he has more than 5 years of experience, then display "YES" else "NO" in the label control.

ADDITIONAL EXERCISES:

1) Develop a simple web page to reproduce the given Captcha. Upon match, suitable message has to be displayed. If there is a mismatch for more than 3 times, TextBox has to be disabled.

Lab No:6 **Date:**

Developing Web Application using Django

Part II

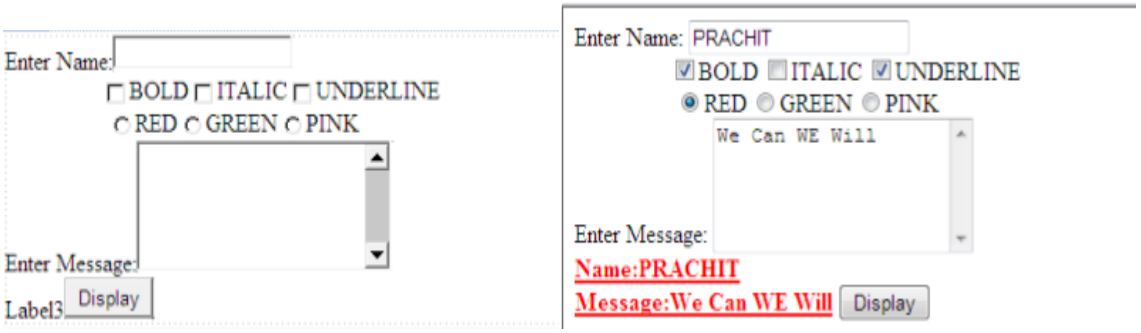
LAB QUESTIONS

1) Develop a simple web page to perform basic arithmetic operations. Take two integer inputs from the user, select the operation to be performed using a dropdown. Include a button "Calculate" to perform the selected operation, and then display the result in the same web page.
2) Develop a simple web form that generates the front cover for a magazine. The form should provide the options for selecting the image, background color, changing font size, color etc. Input messages must be taken from the user so as to display it on the front cover with legible font family and font size. The front cover developed should be proportionate to the web page size. Place the css files inside static folder.

3) Design a simple web application to provide information about a book. The home page of the application should display the cover page of the book along with three hyperlinks: Metadata, Reviews, Publisher info. Give provision to revert to home page from any other page.

ADDITIONAL EXERCISES:

1) Design a simple web application which will ask the user to input his name and a message, display the two items concatenated in a label, and change the format of the label using radio buttons and check boxes for selection, the user can make the label text bold, underlined or italic and change its color. include buttons to display the message in the label, clear the text boxes and label and exit



Lab No:7 **Date:**

Mini-project-phase-I

Objectives:

In this lab, student will be able to

1. Identify an idea to implement a mini project using ASP.NET concepts.
2. Formulate the synopsis for a mini project.
3. Perform the requirement gathering and design phases of the project.

INSTRUCTIONS TO STUDENTS TO CARRY OUT MINI PROJECT:

- Students are supposed to come up with an idea regarding a website.
- Students have to give the name of the project at the end of the 7th week of the regular lab session.
- Students can work in batch containing a maximum of two students.
- The project must cover most of the topics that are worked on during the previous and upcoming lab sessions.
- The project must be completed during the duration between Lab 7 to Lab 12.
- At the end of the last week of regular lab, the report has to be submitted, and the project must be demonstrated to the instructor.

Project Synopsis format

1. Synopsis should contain the following
    a. Project title.
    b. Abstract.
    c. Team members name, Section and roll number.

The mini project carries 18 marks.

Lab No:8 **Date:**

Form Processing using Django – Part I

Objectives:

In this lab, student will be able to

1. Develop web forms using Form class in Django
2. Learn to use Form Widgets to enhance the web forms

3. Design Django web applications using session management techniques

Django Forms:

When one creates a **Form** class, the most important part is defining the fields of the form. Each field has custom validation logic. Forms are basically used for taking input from the user in some manner and using that information for logical operations on databases. For example, registering a user by taking input as his name, email, password, etc.

Django maps the fields defined in Django forms into HTML input fields. Django handles three distinct parts of the work involved in forms:

- preparing and restructuring data to make it ready for rendering
- creating HTML forms for the data
- receiving and processing submitted forms and data from the client **Syntax:**

Django Fields have the following syntax:

field_name = forms.FieldType(**options)

Built in Django Form fields:

The **forms** library comes with a set of **Field** classes that represent common validation needs.

For each field, we describe the default widget used. We also specify the value returned when you provide an empty value.

BooleanField *class*

BooleanField(***kwargs***)**

- Default widget: **CheckboxInput**
- Empty value: **False**
- Normalizes to: A Python **True** or **False** value.

CharField *class*

CharField(***kwargs***)**

- Default widget: **TextInput**
- Empty value: Whatever you've given as **empty_value**.
- Normalizes to: A string.
- Uses arguments **max_length** or **min_length** (integer values),to ensure that the string is at most or at least the given length.

ChoiceField *class*

ChoiceField(***kwargs***)**

- Default widget: **Select**
- Empty value: **''** (an empty string) Normalizes to: A string.
- Validates that the given value exists in the list of choices.

DateField *class*

DateField(***kwargs***)**

- Default widget: **DateInput**
- Empty value: **None**
- Normalizes to: A Python **datetime.date** object.
- Validates that the given value is either a **datetime.date**, **datetime.datetime** or string formatted in a particular date format.

EmailField *class*

EmailField(***kwargs***)**

- Default widget: **EmailInput** Empty value: **''** (an empty string)

  Normalizes to: A string.
- Uses **EmailValidator** to validate that the given value is a valid email address, using a moderately complex regular expression.

FileField *class*

FileField(***kwargs***)**

- Default widget: **ClearableFileInput**
- Empty value: **None**
- Normalizes to: An **UploadedFile** object that wraps the file content and file name into a single object.

IntegerField *class*

IntegerField(***kwargs***)**

- Default widget: **NumberInput** when **Field.localize** is **False**, else **TextInput**.
- Empty value: **None**
- Normalizes to: A Python integer.

Takes two optional arguments for validation:

- **max_value**
- **min_value**

These control the range of values permitted in the field.

URLField *class*

URLField(***kwargs***)**

- Default widget: **URLInput**

FIELD OPTIONS	DESCRIPTION
required	By default, each Field class assumes the value is required, so to make it not required you need to set required=False
label	The label argument lets you specify the "human-friendly" label for this field. This is used when the Field is displayed in a Form.
label_suffix	The label_suffix argument lets you override the form's label_suffix on a per- field basis.
widget	The widget argument lets you specify a Widget class to use when rendering this Field. See Widgets for more information.
help_text	The help_text argument lets you specify descriptive text for this Field. If you provide help_text, it will be displayed next to the Field when the Field is rendered by one of the convenience Form methods.

error_messages	The error_messages argument lets you override the default messages that the field will raise. Pass in a dictionary with keys matching the error messages you want to override.

validators	The validators argument lets you provide a list of validation functions for this field.
localize	The localize argument enables the localization of form data input, as well as the rendered output.
disabled.	The disabled boolean argument, when set to True, disables a form field using the disabled HTML attribute so that it won't be editable by users.

- Empty value: '' (an empty string) Normalizes to: A string.

Takes the following optional arguments:

- **max_length**
- **min_length**

These are the same as **CharField.max_length** and **CharField.min_length**.

Core Field Arguments:

Core Field arguments are the arguments given to each field for applying some constraint or imparting a particular characteristic to a particular Field. For example, adding an argument required = False to CharField will enable it to be left blank by the user.

Creating a Django Form:

To use Django Forms, create a project and an app inside it. After you start an app, create a form in app/forms.py.

For creating a form in Django we have to specify what fields would exist in the form and of what type.

Let us create a form with CharField, IntegerField and BooleanField as follows:

 # app/forms.py from django

import forms class

RegForm(forms.Form): title =

forms.CharField() description =

forms.CharField() views =

forms.IntegerField() available =

forms.BooleanField()

Rendering Django Forms:

Django form fields have several built-in methods to ease the work of the developer but sometimes one needs to implement things manually for customizing User Interface(UI). A form comes with 3 in-built methods that can be used to render Django form fields.

- {{ form.as_table }} will render them as table cells wrapped in <tr> tags
- {{ form.as_p }} will render them wrapped in <p> tags
- {{ form.as_ul }} will render them wrapped in <li> tags

 # app/views.py from

django.shortcuts import render

from .forms import RegForm #

creating a home view def

home_view(request):

context = {} form =

RegForm(request.POST or None)

context['form'] = form return

render(request, "home.html", context)

#app/ templates/home.html

```
<html>
<body>
<form action="" method="POST">
    {{ form.as_p }}
    <input type="submit" value="Submit">
</form>
</body>
</html>
```

Use the command 'python manage.py runserver' to see the following output in the web page (Fig 6.1):

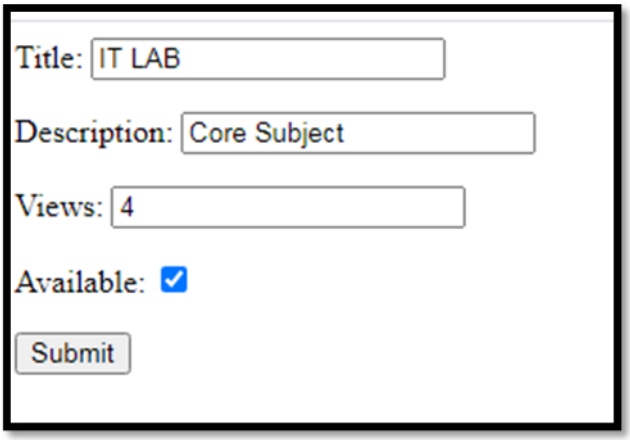


Fig 6.1

Widgets used in Django Forms:

A widget is Django's representation of an HTML input element. The widget handles the rendering of the HTML, and the extraction of data from a GET/POST dictionary that corresponds to the widget.

Whenever you specify a field on a form, Django will use a default widget that is appropriate to the type of data that is to be displayed.

However, if you want to use a different widget for a field, you can use the **widget** argument on the field definition. For example:

```
from django import forms class

CommentForm(forms.Form):

    name = forms.CharField()

url = forms.URLField()

    comment = forms.CharField(widget=forms.Textarea)
```

This would specify a form with a comment that uses a larger **Textarea** widget, rather than the default **TextInput** widget.

Widgets handling input of text

These widgets make use of the HTML elements **input** and **textarea**.

TextInput class

TextInput

- **input_type**: **'text'**
- **template_name**: **'django/forms/widgets/text.html'**
- Renders as: **<input type="text" ...>**

NumberInput class

NumberInput

- **input_type**: **'number'**
- **template_name**: **'django/forms/widgets/number.html'**
- Renders as: **<input type="number" ...>**

EmailInput class

EmailInput¶

- **input_type**: **'email'**
- **template_name**: **'django/forms/widgets/email.html'**

- Renders as: **<input type="email" ...>**

PasswordInput class

PasswordInput

- **input_type**: **'password'**
- **template_name**: **'django/forms/widgets/password.html'**
- Renders as: **<input type="password" ...>**

HiddenInput class

HiddenInput

- **input_type**: **'hidden'**
- **template_name**: **'django/forms/widgets/hidden.html'**
- Renders as: **<input type="hidden" ...>**

DateInput class

DateInput

- **input_type**: **'text'**
- **template_name**: **'django/forms/widgets/date.html'**
- Renders as: **<input type="text" ...>**

Textarea class

Textarea

- **template_name**: **'django/forms/widgets/textarea.html'**
- Renders as: **<textarea>...</textarea>**

CheckboxInput class

CheckboxInput

- **input_type**: **'checkbox'**
- **template_name**: **'django/forms/widgets/checkbox.html'**
- Renders as: **<input type="checkbox" ...>**

Select class

Select

- **template_name**: **'django/forms/widgets/select.html'**
- **option_template_name**: **'django/forms/widgets/select_option.html'**

Renders as: **<select><option ...>...</select>**

Sample code:

CHOICES= (('1','Choice1'), ('2','Choice2'), ('3','Choice3'),) select =

forms.ChoiceField(widget=forms.Select, choices=CHOICES)

RadioSelect class

RadioSelect

- **template_name**: **'django/forms/widgets/radio.html'**
- **option_template_name**: **'django/forms/widgets/radio_option.html'**

Similar to **Select**, but rendered as a list of radio buttons within **<li>** tags:

<**li**><**input** type="radio" name="..."></**li**>

Sample Code:

YES_SMARTPHONE = 'Yes'
NO_SMARTPHONE = 'No'
SMART_PHONE_OWNERSHIP = ((YES_SMARTPHONE, 'Yes'), (NO_SMARTPHONE, 'No'),)
smart_phone_ownership=forms.ChoiceField(widget=forms.RadioSelect(), choices=SMART_PHONE_OWNERSHIP, initial= "", label='Do you own a Smartphone?', required = False)

Custom Django Form field widgets:

We can override the default widget of each field for various purposes. To do so we need to explicitly define the widget we want to assign to a field.
Make following changes to app/forms.py

```
from django import forms

class GeeksForm(forms.Form):
    title = forms.CharField(widget = forms.Textarea)
    description = forms.CharField(widget = forms.CheckboxInput)
views = forms.IntegerField(widget = forms.TextInput)
available = forms.BooleanField(widget = forms.Textarea)
```

The output obtained will be as follows (Fig 6.2):

Fig 6.2

Solved Exercise:

A Sample program to demonstrate passage of multiple parameters from one page to another.

#loginapp/ forms.py

```
from django import forms class
LoginForm(forms.Form):
```

```
  username = forms.CharField(max_length = 100)
contact_num = forms.IntegerField()
```

#loginapp/views.py

```
from django.shortcuts import render
from loginapp.forms import LoginForm
def login(request):    username = "not
logged in"    cn="not found"    if
request.method == "POST":
     #Get the posted form
          MyLoginForm  =  LoginForm(request.POST)
if MyLoginForm.is_valid():
          username = MyLoginForm.cleaned_data['username']              cn=
MyLoginForm.cleaned_data['contact_num']

else:
          MyLoginForm = LoginForm()

  context = {'username': username,'contact_num':cn}

  return render(request, 'loggedin.html',context)
```

#loginapp/templates/login.html

```
<html>
  <body>

    <form name = "form" action = "{% url 'login' %}"
    method = "POST" >{% csrf_token %}

       <div style = "max-width:470px;">
         <center>
           <input  type  =  "text"  style  =  "margin-left:20%;"
placeholder = "Identifiant" name = "username" />
         </center>
       </div>
```

```
        <br>

        <div style = "max-width:470px;">
          <center>
```



```
            <input type = "number" style = "margin-left:20%;"
             placeholder = "contact_number" name = "contact_num" />            </center>
</div>

        <br>

        <div style = "max-width:470px;">
          <center>

            <button style = "border:0px; background-color:#4285F4; margin-top:8%;
height:35px; width:80%;margin-left:19%;" type = "submit"                value =
"Login" >                <strong>Login</strong>
            </button>

          </center>
        </div>

      </form>

    </body>
</html>
```

#loginapp/templates/loggedin.html

```
<html>
   <body>
     You are : <strong>{{username}}</strong>
     Your number is : <strong>{{contact_num}}</strong>
  </body>   </html>
```

Output (Fig 6.3):

Command to be used: E:\MyFolder\FormProject > python manage.py runserver
Lab No:6

Fig 6.3

Django Sessions:

Sessions are used to abstract the receiving and sending of cookies, data is saved on server side (like in database), and the client side cookie just has a session ID for identification. Sessions are also useful to avoid cases where the user browser is set to 'not accept' cookie

Setting Up Sessions

In Django, enabling session is done in your project **settings.py**, by adding some lines to the **MIDDLEWARE_CLASSES** and the **INSTALLED_APPS** options. This should be done while creating the project, so **MIDDLEWARE_CLASSES** should have −

'**django.contrib.sessions.middleware.SessionMiddleware**'

And **INSTALLED_APPS** should have −

'**django.contrib.sessions**'

By default, Django saves session information in database (django_session table or collection), but we can configure the engine to store information using other ways like: in **file** or in **cache**.

When session is enabled, every request (first argument of any view in Django) has a session (dict) attribute.

Solved Exercise: #sessapp/forms.py

```
from django import forms class LoginForm(forms.Form):
username = forms.CharField(max_length = 100)    password=
forms.CharField(widget= forms.PasswordInput())
```

#Sessapp/views.py

```
from django.shortcuts import render
from sessapp.forms import LoginForm
def login(request):    username = 'not
logged in'       if request.method ==
'POST':
          MyLoginForm   =   LoginForm(request.POST)
if MyLoginForm.is_valid():
          username = MyLoginForm.cleaned_data['username']
request.session['username'] = username       else:
        MyLoginForm = LoginForm()
  return render(request, 'loggedin.html', {"username" : username})

def formView(request):    if
request.session.has_key('username'):
username = request.session['username']
     return   render(request,   'loggedin.html',   {"username"   :   username})
else:
     return render(request, 'login.html', { })

def       logout(request):
try:
     del request.session['username']
except:       pass
  return HttpResponse("<strong>You are logged out.</strong>")
```

#sessapp/templates/login.html


```
<html>
  <body>

    <form name = "form" action = "{% url 'login' %}"
    method = "POST" >{% csrf_token %}

      <div style = "max-width:470px;">
        <center>
          <input type = "text" style = "margin-left:20%;"
placeholder = "Identifiant" name = "username" />
        </center>
      </div>

      <br>

      <div style = "max-width:470px;">
        <center>
          <input type = "password" style = "margin-left:20%;"
placeholder = "password" name = "password" />
        </center>
      </div>

      <br>

      <div style = "max-width:470px;">
        <center>

          <button style = "border:0px; background-color:#4285F4; margin-top:8%;
height:35px; width:80%;margin-left:19%;" type = "submit"          value =
"Login" >          <strong>Login</strong>
          </button>
```

```
        </center>
      </div>

    </form>

  </body>
</html>
```

#sessapp/templates/loggedin.html

```
<html>

  <body>
    You are : <strong>{{username}}</strong>
  </body>

</html>
```

#sessapp/urls.py

```
from django.conf.urls import url from
. import views

urlpatterns = [
  url(r'^connection/',views.formView, name = 'formView'),
  url(r'^login/', views.login, name = 'login'),
  url(r'^logout/', views.logout, name = 'logout'),
]
```

Output (Fig 6.4):

Commands to be used:

E:\MyFolder\SessProject> python manage.py migrate

E:\MyFolder\SessProject> python manage.py runserver

Fig 6.4

LAB EXERCISES:

1) Develop a web application using Django framework to demonstrate the transfer of multiple parameters between web pages. User should be presented with a dropdown list containing car manufacturers, a text box which takes model name of the manufacturer and a submit button. On submitting the web page, the user is forwarded to a new page. This new page should display the selected car manufacturer name and the model name.
2) Create a page firstPage.html with two TextBoxes [Name, Roll], DropDownList [Subjects], and a button. Create another page secondPage.html with a label and a button. When the user clicks the button in first Page, he should be sent to the second page and display the contents passed from first page in the label. The button in second page should navigate the user back to firstPage. Use Django sessions to transfer information.

ADDITIONAL EXERCISES:

1) Develop a Web Application for Grocery Checklist Generation as shown in the figure below. It must have **checkboxes** which must be populated on page load listing grocery items. On clicking the **Add Item** button the selected Items and their prices have to be displayed in a Table. Set the borderstyle and border width for the table and its cells.

Lab No:9 **Date:**

Form Processing using Django – Part II

LAB QUESTIONS

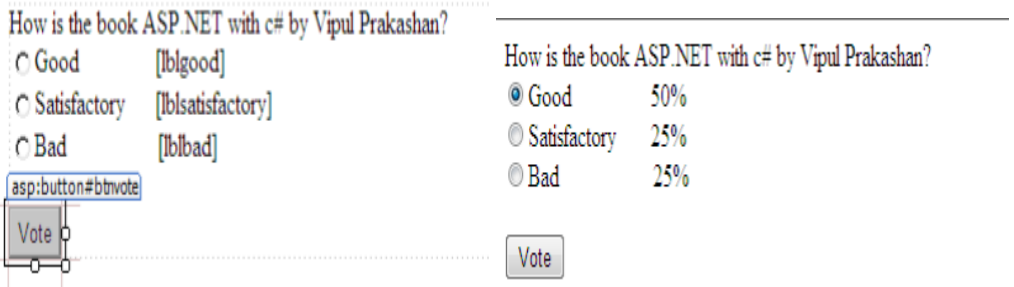
1) Create a Register page and Success page with the following requirements:

i. Register page should contain four input TextBoxes for UserName, Password, Email id and Contact Number and also a button to submit. Make the username as compulsory field and other fields as optional.

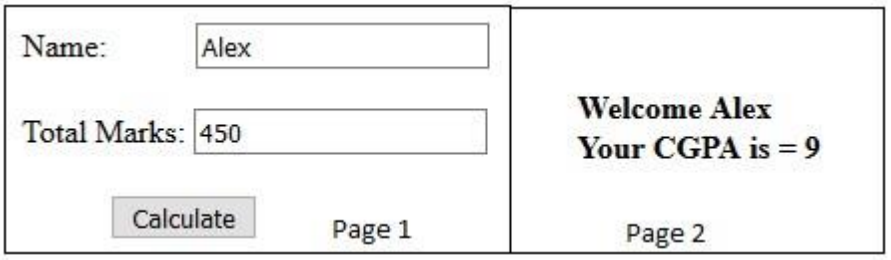
ii. On button click, Success page is displayed with message "Welcome {UserName}" and also his Email and Contact Number has to be displayed.

iii. Use secure technique to send details to the Success page (Hint: use csrftoken) 4) Design a website with two pages.

2) "How is the book ASP.NET with c# by Vipul Prakashan?" Give the user three choice : i) Good ii) Satisfactory iii) Bad. Provide a VOTE button. After user votes, present the result in percentage using labels next to the choices



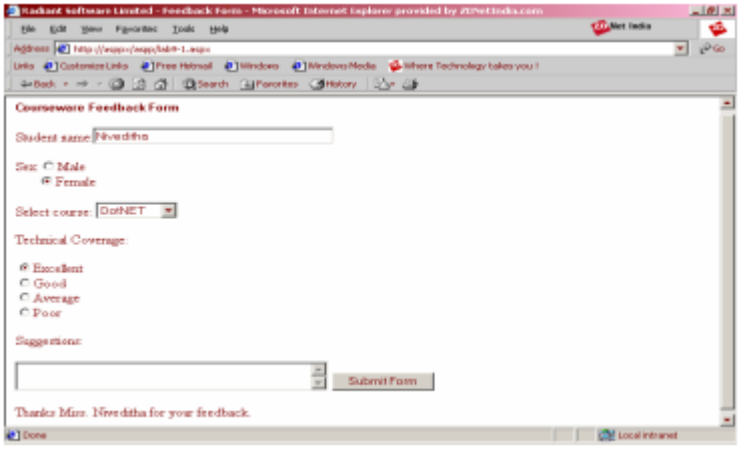
3) Create a website with two pages. Page 1 has two TextBoxes (name and total marks) and one 'Calculate' Button as shown in the figure. On clicking the 'Calculate' Button, CGPA (total marks/50) along with the name should be displayed in the Page 2. Use Django sessions to store the information.



ADDITIONAL QUESTIONS

1) Create a website with two pages.
   First page contains:
   RadioButton with HP, Nokia, Samsung, Motorola, Apple as options.
   CheckBox with Mobile and Laptop as items.
   TextBox to enter quantity.
   There is a button with text as "Produce Bill".
   On Clicking Produce Bill button, item should be displayed with total amount on another page.

2)Write a program to display the following feedback form. The different options for the list box must be ASP-XML, DotNET, JavaPro and Unix,C,C++. When the Submit Form button is clicked after entering the data, a message as seen in the last line of the above figure must be displayed.



Lab No:10 **Date:**

Databases – Part I

Objectives:

In this lab, student will be able to

1. Understand the MTV architecture
2. Create an App in Django and establish a connection with SQLite database
3. Set different privileges to different types of users.
4. Set the Django administrator account.

Django supports following databases

1. MySql
2. PostgreSql
3. Oracle
4. Sqllite

With the help of 3rd party backend Django supports following databases

1. SAP SQL Anywhere
2. IBM DB2
3. Microsoft SQL Server
4. Firebird
5. ODBC
6. ADSDB

Django abstracts the details of underlying database. One only need to specify the (models.py) python functions which will be converted into underlying database statements. Django supports CRUD operations. There are two way one can control the data on the website. First way is to use the admin interface second way is to use the forms.

Solved Exercise

Model is the name given to data abstraction part. To create the model you must first create an app. To create an app right click on the project→Add→DjangoApp
Let us name the app as “blog”

Step1: In settings.py add the app name (blog) under Installed_Apps as follows:

```
INSTALLED_APPS = [
    # Add your apps here to enable them
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'blog'
]
```

Under Templates provide the path of the template directory as follows

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR,'blog/templates/blog')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

If you are using sqllite leave the default setting for database which will look as follows

```
DATABASES = {
  'default': {
     'ENGINE': 'django.db.backends.sqlite3',
    'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
  }
}
```

If you are using MySQL in that case modify the database entry as follows

```
DATABASES = {
        'default': {
                        'ENGINE':'django.db.backends.mysql',
 'OPTIONS': {' read_default_file': '/path/to/my.cnf',},
}
}
```

My.cnf file is as follows:

```
# my.cnf
[client]
database = NAME

user = USER

password = PASSWORD
default-character-set = utf8
```

Step2: Modify the Projects urls.py as given below from

```
django.conf.urls import include, url

# Uncomment the next two lines to enable the admin:

from django.contrib import admin admin.autodiscover()

urlpatterns = [
    # Examples:
    # url(r'^$', MyBlog.views.archive, name='archive'),
    #url(r'^MyBlog/', include('MyBlog.MyBlog.urls')),
    # Uncomment the admin/doc line below to enable admin documentation:
    # url(r'^admin/doc/', include('django.contrib.admindocs.urls')),

    # Uncomment the next line to enable the admin:
  # (r'^$', 'django.views.generic.simple.redirect_to',
  # {'url': '/blog/'}),    url(r'^blog/',

include('blog.urls')),    url(r'^admin/',

include(admin.site.urls)),

]
```

Step3: Under blog app create a file named urls.py and type the following

```
from django.conf.urls import include,url  from blog.views import

archive,create_blogpost

urlpatterns = [       url(r'^$', archive, name='archive'),

url(r'^create/', create_blogpost, name='create_blogpost'),

    ]
```

Step4: Under models.py type the following from

django.db import models

Create your models here. from

django import forms class

BlogPost(models.Model):

title

models.CharField(max_length=

150) body =

models.TextField()

timestamp =

models.DateTimeField()

```
    class Meta:
        ordering = ('-timestamp',)
class BlogPostForm(forms.ModelForm):
    class Meta:
        model = BlogPost
```

exclude = ('timestamp',)

It contains the details of table and Model form uses the model already created to create the form. This approach avoids duplication of code and goes with python philosophy Do not Repeat Yourself.

Step5: Registering your app in the admin: To register your app type the following into

admin.py from django.contrib import admin import site from blog.models import

BlogPost

```
# Register your models here. from blog
import models class
BlogPostAdmin(admin.ModelAdmin):
    list_display = ('title', 'timestamp')
admin.site.register(models.BlogPost,BlogPostAdmin)
```

Step6: Type the following into views.py from

```
django.shortcuts import render

# Create your views here. from datetime import
datetime from django.http import
HttpResponseRedirect from django.shortcuts
import render from blog.models import BlogPost,
BlogPostForm
def archive(request):
    posts = BlogPost.objects.all()[:10]
return render(request, 'archive.html',
                        {'posts': posts, 'form': BlogPostForm()})
def create_blogpost(request):
if request.method == 'POST':
        form = BlogPostForm(request.POST)
        if form.is_valid():
```

```
        post = form.save(commit=False)
post.timestamp=datetime.now()
        post.save()
   return HttpResponseRedirect('/blog/')
```

It is displaying the 10 most recent blogs posted by users/admin.

Step7: Type the following lines into archive.html

```
<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
   <meta charset="utf-8" />
   <title></title>
</head>
<body>
<form action="/blog/create/" method=post>{% csrf_token %}
   <table>{{ form }}</table><br>
   <input type=submit>
</form>
<hr>
{% for post in posts %}
   <h2>{{ post.title }}</h2>
   <p>{{ post.timestamp }}</p>
   <p>{{ post.body }}</p>
```

<hr>

{% endfor %}

</body> </html>

This is the template which displays the blog posts that are separated by horizontal rule.

Once you have typed all of the above

i. Go to Projects→Django Check

If it succeeds then

ii. Goto Projects→Django Make Migrations

If it succeeds then

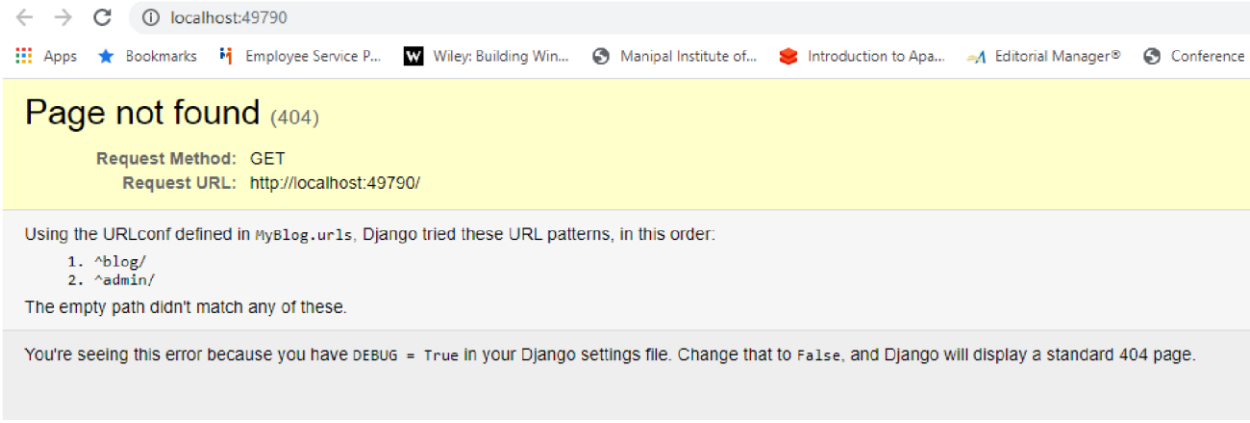
iii. Goto Projects→Django Migrate

If it Succeeds then

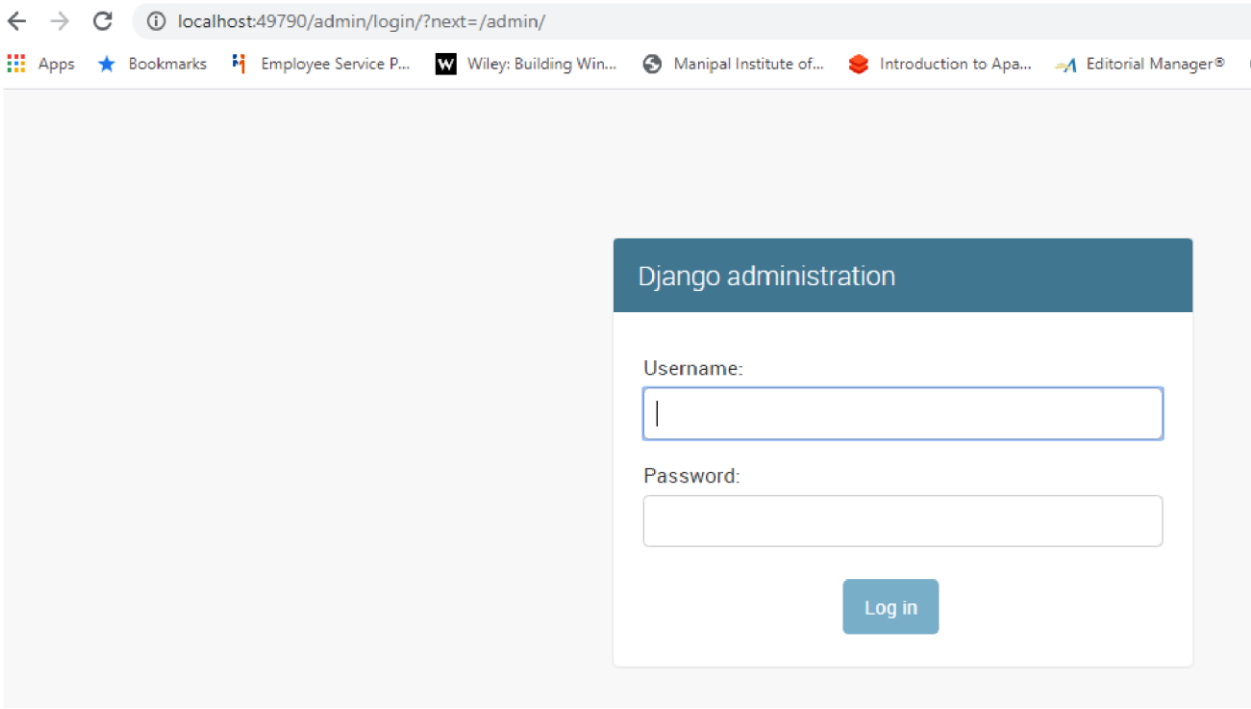
iv. Goto Projects→Django Create Superuser

You have to repeat the above four steps whenever you modify the model or use different database.

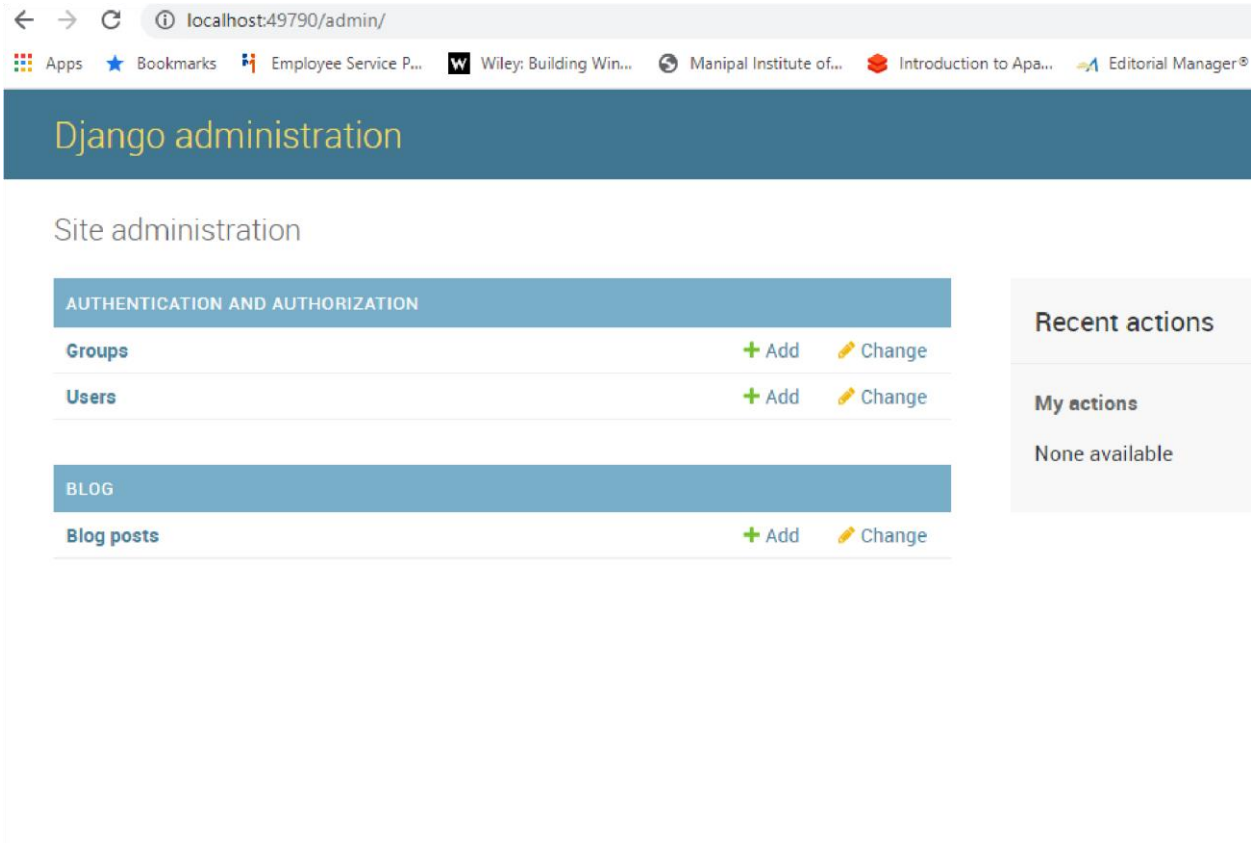
Once you have created the superuser you can open the website. It looks as follows



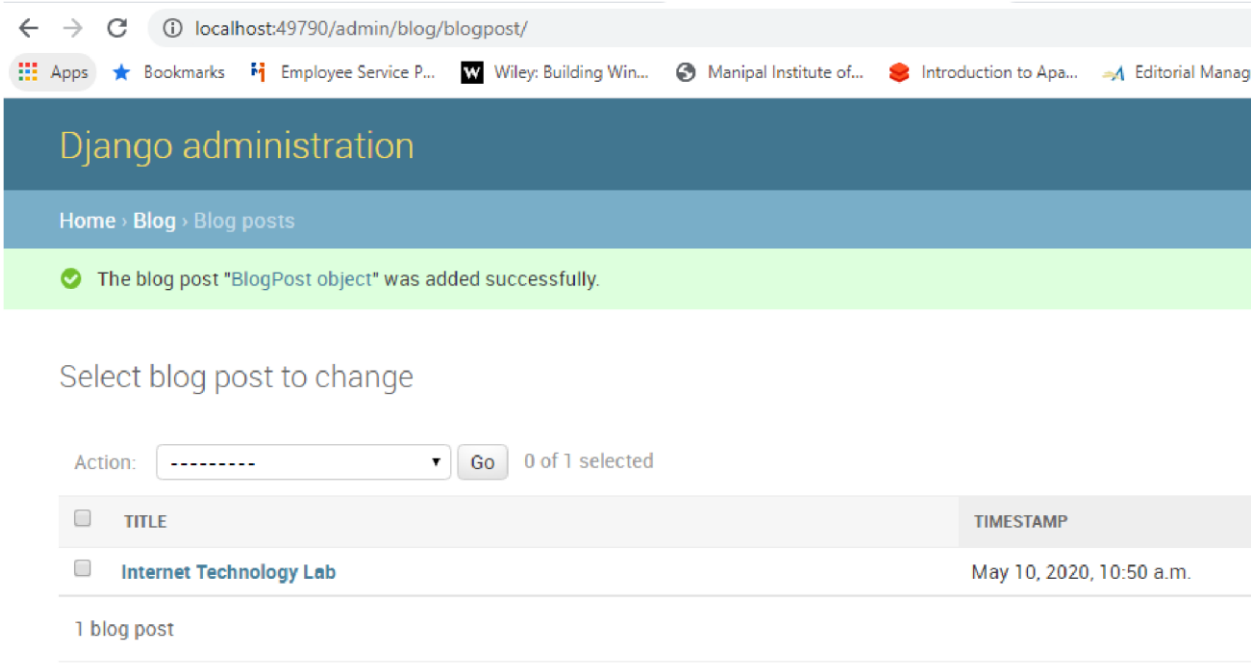
append /admin to the host name you will get the following screen



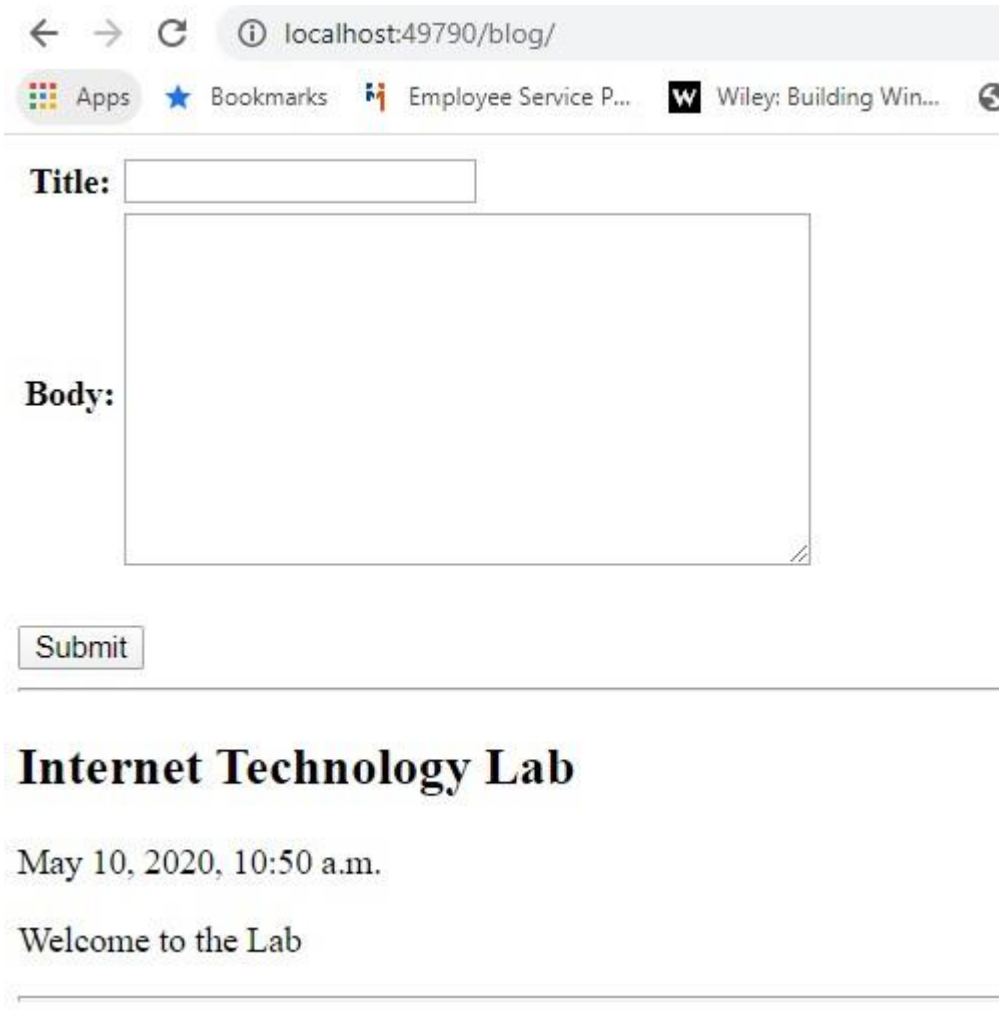
Type the superuser name and password you will be taken to following admin page. In the admin page you can see entry for Blog Posts as you have registered it.



Add a blog post You will be taken to following screen you can observe only title and timestamp are visible as per our code



Now you append the blog to the address you will get the following output.



As per our instructions users cannot edit the timestamp and current date and time will be taken for user entry. The blog entered by admin is also displayed. Once you enter the blog post details it will display it as under.


localhost:49790/blog/
Apps
Bookmarks
Employee Service P...
Wiley: Building Win...
Manipal Institute
Title:
Body:
Submit
Internet Technology Lab
May 10, 2020, 4:29 p.m.
Do the lab excercises
Internet Technology Lab
May 10, 2020, 10:50 a.m.
Welcome to the Lab

LAB EXCERCISES

1. Design a web site using Django, which is a website directory – A site containing links to other websites. A web page has different categories.
   - A category table has a name, number of visits, and number of likes.
   - A page table refers to a category, has a title, URL, and many views.

   Design a form that populates the above database and displays it.

2. Consider the following tables:
   WORKS(person-name,Company-name,Salary)
   LIVES(Person_name, Street, City)
   Assume Table data suitably. Design a Django webpage and include an option to insert data into WORKS table by accepting data from the user using TextBoxes. Also, include an option to retrieve the names of people who work for a particular company along with the cities they live in (particular company name must be accepted from the user).

ADDITIONAL EXERCISES

1. Assume a table “Institutes” with institute_id, name, and no_of_courses are the fields. Create a web page that retrieves all the data from “Institutes” table displays only Institute names in the list box.

Lab No:11 **Date:**

Databases-Part II

Lab Exercise

1. There are three tables in the database an author table has a first name, a last name and an email address. A publisher table has a name, a street address, a city, a state/ province, a country, and a Web site. A book table has a title and a publication date. It also has one or more authors (a many-to-many relationship with authors) and a single publisher (a one-to-many relationship - aka foreign key - to publishers). Design a form which populates and retrieves the information from the above database using Django.

2. Create a Django Page for entry of a Product information (title, price and description) and save it into the db. Create the index page where you would view the product entries in an unordered list.

3. Create a web page with DropDownList, Textboxes and Buttons. Assume the table 'Human' with First name, Last name, Phone, Address and City as fields. When the page is loaded, only first names will be displayed in the drop-down list. On selecting the name, other details will be displayed in the respective TextBoxes. On clicking the update button, the table will be updated with new entries made in the text box. On clicking the delete button, the selected record will be deleted from the table, and the DropDownList is refreshed.

ADDITIONAL QUESTIONS

1. Create a web page that receives the following information from a set of students: Student Id, Student Name, Course Name and Date of Birth. The application should also display the information of all the students once the data is Entered.

Lab No:12 **Date:**

Mini-project-Phase-II

Demonstrate the website developed as part of the mini project along with report and presentation.

Project Details

1. Student must do a mini project in Django.
2. Student must submit the synopsis in 7th lab.
3. Complete the Django mini project and demonstrate by 12th lab.
4. Student must submit the report in 12th lab.

Project Report format for research projects

1. Abstract
2. Motivation
3. Objectives
4. Introduction
5. Literature review
6. Methodology
7. Results
8. Limitations and Possible Improvements
9. Conclusion
10. References

Other type of projects can exclude literature review.

SlNo	Topic	Marks
1	Synopsis, Abstract, Problem Statement	3
2	Design:ER Diagram	3
3	Demo: database connectivity	5
4	UI Design	4
5	Report	3

References:

1. Mark Lutz, Learning Python, 5th Edition, O'Reilly, 2013

2. Nigel George, Mastering Django, Packt Publishing, 2016.

3. Leif Azzopardi and David Maxwell, Tango with Django 2, Apress, 2019