Summer Training Report

on

PYTHON & DJANGO FULL STACK WEB DEVELOPER BOOTCAMP

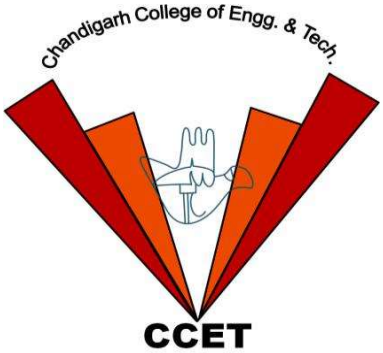
A Project Report/Synopsis submitted in partial fulfillment of the requirements for the award of

Bachelor of Engineering
IN
COMPUTER SCIENCE AND ENGINEERING

Submitted by

Ashlin K Siby (CO16307)
Keshav Garg (CO16326)
Vivsvaan Sharma (CO16362)

Under the supervision of
Dr. Ankit Gupta



CHANDIGARH COLLEGE OF ENGINEERING AND TECHNOLOGY (DEGREE WING)

Government Institute under Chandigarh (UT) Administration, Affiliated to Panjab University , Chandigarh

Sector-26, Chandigarh. PIN-160019

July-December, 2018

CHANDIGARH COLLEGE OF ENGINEERING AND TECHNOLOGY (DEGREE WING)
Government Institute under Chandigarh (UT) Administration | Affiliated to Panjab University , Chandigarh
Sector-26, Chandigarh. PIN-160019 | Tel. No. 0172-2750947, 2750943
Website: www.ccet.ac.in | Email: principal@ccet.ac.in | Fax. No. :0172-2750872

Department of Computer Sc. & Engineering

CANDIDATE'S DECLARATION

We hereby declare that the work presented in this report entitled "**Python and Django Full Stack Web Developer Bootcamp**", in fulfillment of the requirement for the award of the degree **Bachelor of Engineering in Computer Science & Engineering**, submitted in CSE Department, Chandigarh College of Engineering & Technology (Degree wing) affiliated to Punjab University, Chandigarh, is an authentic record of our own work carried out during my degree under the guidance of **udemy.com**. The work reported in this has not been submitted by us for award of any other degree or diploma.

Date: 4 December 2018

Place: Chandigarh

Ashlin K siby
CO16307

Keshav Garg
CO16326

Vivsvaan Sharma
CO16362


Certificate of Completion
This is to certify that Keshav Garg successfully completed 32 hours of Python and Django Full Stack Web Developer Bootcamp online course on June 27, 2018
Jose Portilla
Jose Portilla, Instructor
&
Udemy
Certificate no: UC-QL424F29
Certificate url: ude.my/UC-QL424F29
#BeAble


Certificate of Completion
This is to certify that Ashlin Siby successfully completed 32 hours of Python and Django Full Stack Web Developer Bootcamp online course on June 20, 2018
Jose Portilla
Jose Portilla, Instructor
&
Udemy
Certificate no: UC-LCMS86QH
Certificate url: ude.my/UC-LCMS86QH
#BeAble


Certificate of Completion
This is to certify that Vivsvaan Sharma successfully completed 32 hours of Python and Django Full Stack Web Developer Bootcamp online course on June 22, 2018
Jose Portilla
Jose Portilla, Instructor
&
Udemy
Certificate no: UC-9Y3KJLRA
Certificate url: ude.my/UC-9Y3KJLRA
#BeAble

ACKNOWLEDGEMENT

It is a great pleasure to present this report on our Summer Training, covering "Python and Django Full Stack Web Developer Bootcamp", undertaken by us during June, 2018. We are thankful to the Department of Computer Science and Engineering for encouraging us to undergo such training to enhance our knowledge of new fields and technologies in Computer Science and Engineering and thus, expand our horizons. We would also like to acknowledge the continuous engagement and help that we received from the Department during the preparation of this final report.

We would like to express our gratitude towards Udemy.com, the platform which provided us the resources and knowledge necessary in pursuing Python and Django Full Stack Web Developer Bootcamp. Their quick help regarding our doubts and feedback for our work was immensely helpful in allowing us to successfully complete the training and achieve the criteria of completion. Finally, we would like to thank our training guide, Dr. Ankit Gupta, for introducing and making us aware of the opportunities we had. He took personal interest in our endeavor's despite his busy schedule and numerous commitments, providing us with his time, encouragement, help, valuable guidance and technical expertise. Without his help, we might never have been able to successfully complete our training.

Ashlin K Siby
CO16307

Keshav Garg
CO16326

Vivsvaan Sharma
CO16362

ABSTRACT

This report is meant to describe the summer training which we, Ashlin K Siby, Keshav Garg and Vivsvaan Sharma undertook during the month of June 2018, which taught us about "Python and Django Full Stack Web Developemnt". The report intends to give the reader an overview of the content of the training as well as some of the project work that we took up in completing the same.

This report introduces our project "**AKADEMIA**" which is a data-driven website used by the admin, teachers and students. This website has two major components: Study Corner and Academic & Holiday Calendar. The major part of our project is study corner that consists of three components which are notes, question paper and study material. The implementation details will be introduced in the report. The implementation uses a tool called Django Framework which is an excellent open source web application frame work for complex data-driven website development. The major part of this report will introduce you how to create static website using HTML, CSS and JS; Introduction to Python and Django and lastly, our project "AKADEMIA".

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1: HTML

1.1 Introduction

HTML stands for HyperText Markup Language

It is the most basic building block of the web and every website will need it in some form or another. It is our first fundamental step in understanding how to build web applications.

Structure

All HTML documents must start with a document type declaration: <!DOCTYPE html>. The HTML document itself begins with <html> and ends with </html>. The visible part of the HTML document is between <body> and </body>.

HTML Headings: They are defined with the <h1> to <h6> tags. <h1> defines the most important heading. <h6> defines the least important heading:

HTML Paragraphs: They are defined with the <p> tag:

HTML Links: They are defined with the <a> tag:

Example:

```
<!DOCTYPE html>
     <html>
          <body>
               <h1>My First Heading</h1>
               <p>My first paragraph.</p>
               <a href="https://www.w3schools.com">This is a
          link</a>
          </body>
     </html>
```

1.2 Basics of HTML

Tags

The <html> tag tells the browser that this is an HTML document. The <html> tag represents the root of an HTML document. The <html> tag is the container for all other HTML elements (except for the <!DOCTYPE> tag).

Lists

Unordered HTML List

An unordered list starts with the <ul> tag. Each list item starts with the <li> tag. The list items will be marked with bullets (small black circles) by default:

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Ordered HTML List

An ordered list starts with the <ol> tag. Each list item starts with the <li> tag. The list items will be marked with numbers by default:

```
<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

1.3 Div and Span

Div

The tag defines a division or a section in an HTML document. The element is often used as a container for other HTML elements to style them with CSS or to perform certain tasks with JavaScript.

```
<div style="background-color:lightblue">
  <h3>This is a heading</h3>
  <p>This is a paragraph.</p>
</div>
```

Span

The tag is used to group inline-elements in a document. The tag provides no visual change by itself. The tag provides a way to add a hook to a part of a text or a part of a document.

1.4 Attributes

All HTML elements can have attributes. Attributes provide additional information about an element

Attributes are always specified in the start tag Attributes usually come in name/value pairs like: name="value"

Example: The link address in <a> tag is specified in the href attribute:

1.5 Tables & Forms in HTML

```
<table style="width:100%">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

Tables

An HTML table is defined with the <table> tag. Each table row is defined with the <tr> tag. A table header is defined with the <th> tag. By default, table headings are bold and centered. A table data/cell is defined with the <td> tag.

Forms

The HTML <form> element defines a form that is used to collect user input. An HTML form contains form elements. Form elements are different types of input elements, like text fields, checkboxes, radio buttons, submit buttons, and more. Different types of attributes are also present like target, method, Action etc.

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

When to Use GET?

The default method when submitting form data is GET. However, when GET is used, the submitted form data will be visible in the page address field:

When to Use POST?

Always use POST if the form data contains sensitive or personal information. The POST method does not display the submitted form data in the page address field.

```
<form action="/action_page.php">
  First name:<br>
  <input type="text" name="firstname" value="Mickey"><br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse"><br><br>
  <input type="submit" value="Submit">
</form>
```

Chapter 2: Introduction to CSS

2.1 Introduction

CSS stands for Cascading Style Sheets. CSS describes how HTML elements are to be displayed on screen, paper, or in other media. CSS saves a lot of work. It can control the layout of multiple web pages all at once. External stylesheets are stored in CSS files

CSS Syntax

A CSS rule-set consists of a selector and a declaration block:

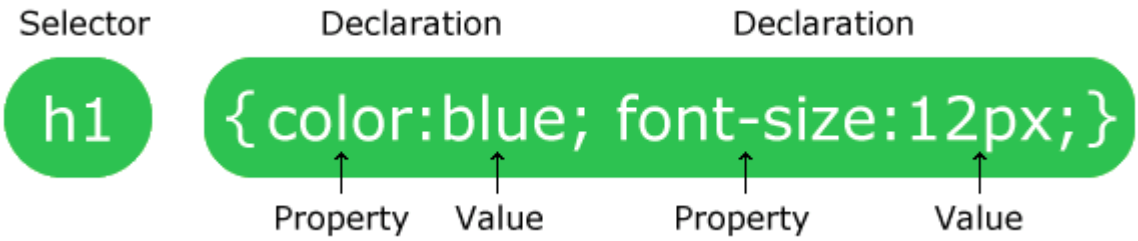


Fig: 2.1

Three Ways to Insert CSS:

External style sheet

Internal style sheet

Inline style

2.2 CSS Properties

Basic Properties

CSS Borders: The CSS border properties allow you to specify the style, width, and color of an element's border.

CSS Margins: The CSS margin properties are used to create space around elements, outside of any defined borders. With CSS, you have full control over the margins. There are properties for setting the margin for each side of an element (top, right, bottom, and left).

CSS Paddings: The CSS padding properties are used to generate space around an element's content, inside of any defined borders. With CSS, you have full control over the padding.

There are properties for setting the padding for each side of an element (top, right, bottom, and left).

CSS Height and Width: The height and width properties are used to set the height and width of an element. The height and width can be set to auto (this is default. Means that the browser calculates the height and width), or be specified in length values, like px, cm, etc., or in percent (%) of the containing block.

CSS [attribute] Selector

The [attribute] selector is used to select elements with a specified attribute.

The following example selects all <a> elements with a target attribute:

```
a[target] {
  background-color: yellow;
}
```

2.3 The Box Model

All HTML elements can be considered as boxes. In CSS, the term "box model" is used when talking about design and layout.

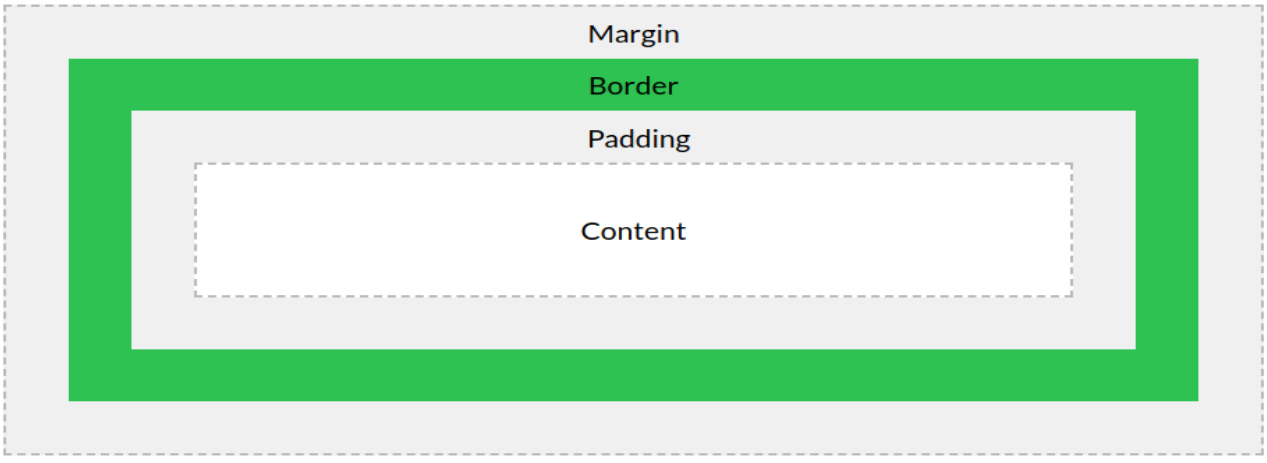


Fig: 2.3

The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model:

Explanation of the different parts:

Content - The content of the box, where text and images appear

Padding - Clears an area around the content. The padding is transparent

Border - A border that goes around the padding and content

Margin - Clears an area outside the border. The margin is transparent

2.4 Display Properties and Pseudo-Classes

The display property specifies if/how an element is displayed.

Every HTML element has a default display value depending on what type of element it is. The default display value for most elements is block or inline.

The position property specifies the type of positioning method used for an element (static, relative, fixed, absolute or sticky).

What are Pseudo-classes?

A pseudo-class is used to define a special state of an element. For example, it can be used to Style an element when a user mouses over it, Style visited and unvisited links differently, Style an element when it gets focus

2.5 Layout of the Website

A website is often divided into headers, menus, content and a footer:

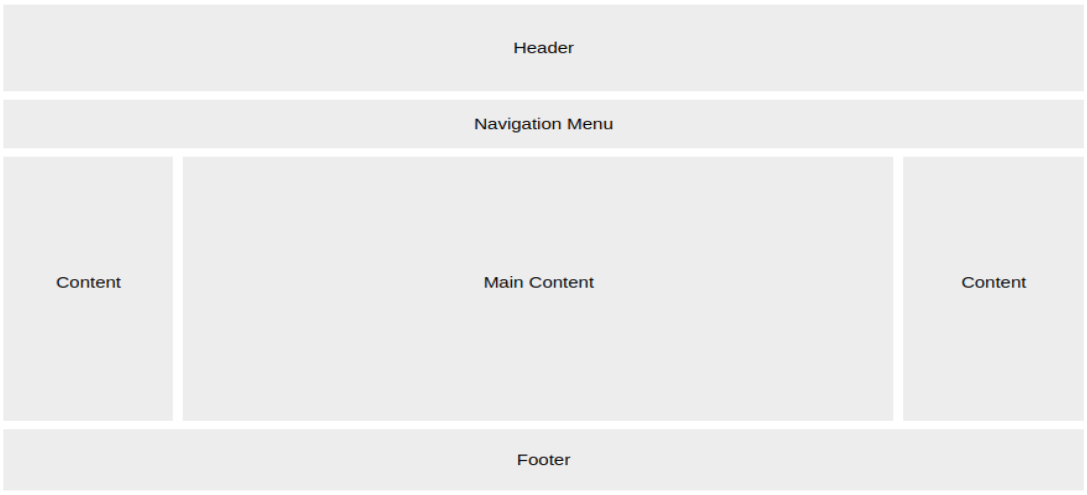


Fig: 2.5

What is Specificity?

If there are two or more conflicting CSS rules that point to the same element, the browser follows some rules to determine which one is most specific and therefore wins out.

Think of specificity as a score/rank that determines which style declarations are ultimately applied to an element.

The universal selector (*) has low specificity, while ID selectors are highly specific!

CSS Box Sizing

The CSS box-sizing property allows us to include the padding and border in an element's total width and height. Without the CSS box-sizing Property.

By default, the width and height of an element is calculated like this:

width + padding + border = actual width of an element
height + padding + border = actual height of an element

This means: When you set the width/height of an element, the element often appear bigger than you have set (because the element's border and padding are added to the element's specified width/height).

2.6 Flexbox

CSS Flexbox Layout Module

Before the Flexbox Layout module, there were four layout modes:

- Block, for sections in a webpage
- Inline, for text
- Table, for two-dimensional table data
- Positioned, for explicit position of an element

The Flexible Box Layout Module, makes it easier to design flexible responsive layout structure without using float or positioning.

We first need to define a flex container. The flex container becomes flexible by setting the display property to flex.

The flex container properties are:

- flex-direction
- flex-wrap
- flex-flow
- justify-content
- align-items
- align-content

2.7 Media Queries

CSS3 Introduced Media Queries

Media queries in CSS3 extended the CSS2 media types idea: Instead of looking for a type of device, they look at the capability of the device.

Media queries can be used to check many things, such as:

width and height of the viewport

width and height of the device

orientation (is the tablet/phone in landscape or portrait mode?)

resolution

Using media queries are a popular technique for delivering a tailored style sheet to desktops, laptops, tablets, and mobile phones (such as iPhone and Android phones).

A media query consists of a media type and can contain one or more expressions, which resolve to either true or false.

```
@media not|only mediatype and (expressions) {
    CSS-Code;
}
```

2.8 Bootstrap Overview

Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites. It is completely free to download and use.

Example

```
<div class="jumbotron text-center">
  <h1>My First Bootstrap Page</h1>
  <p>Resize this responsive page to see the effect!</p>
</div>

<div class="container">
  <div class="row">
    <div class="col-sm-4">
      <h3>Column 1</h3>
      <p>Lorem ipsum dolor..</p>
      <p>Ut enim ad..</p>
    </div>
    <div class="col-sm-4">
      <h3>Column 2</h3>
      <p>Lorem ipsum dolor..</p>
      <p>Ut enim ad..</p>
    </div>
    <div class="col-sm-4">
      <h3>Column 3</h3>
      <p>Lorem ipsum dolor..</p>
      <p>Ut enim ad..</p>
    </div>
  </div>
</div>
```

Bootstrap Buttons

Bootstrap provides different styles of buttons:

To achieve the button styles above, Bootstrap has the following classes:

Fig: 2.8.1

.btn

.btn-*default*

.btn-primary

.btn-success

.btn-info

.btn-warning

.btn-danger

.btn-link

Bootstrap provides four button sizes:

The classes that define the different sizes are:

.btn-lg

.btn-md

.btn-sm

.btn-xs

Navigation Bars

A navigation bar is a navigation header that is placed at the top of the page:

With Bootstrap, a navigation bar can extend or collapse, depending on the screen size.

A standard navigation bar is created with <nav class=”navbar navbar-default”>.

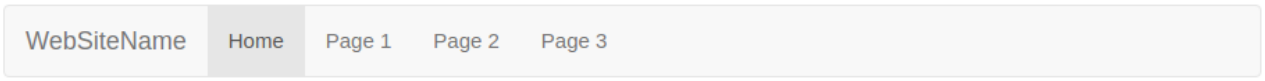


Fig: 2.8.2

The following example shows how to add a navigation bar to the top of the page:

```
<nav class=”navbar navbar-default”>
  <div class=”container-fluid”>
    <div class=”navbar-header”>
      <a class=”navbar-brand” href=”#”>WebSiteName</a>
    </div>
    <ul class=”nav navbar-nav”>
      <li class=”active”><a href=”#”>Home</a></li>
      <li><a href=”#”>Page 1</a></li>
```

```
    <li><a href="#">Page 2</a></li>
    <li><a href="#">Page 3</a></li>
   </ul>
  </div>
</nav>
```

2.9 Bootstrap Grid System

Bootstrap's grid system allows up to 12 columns across the page. If you do not want to use all 12 column individually, you can group the columns together to create wider columns:

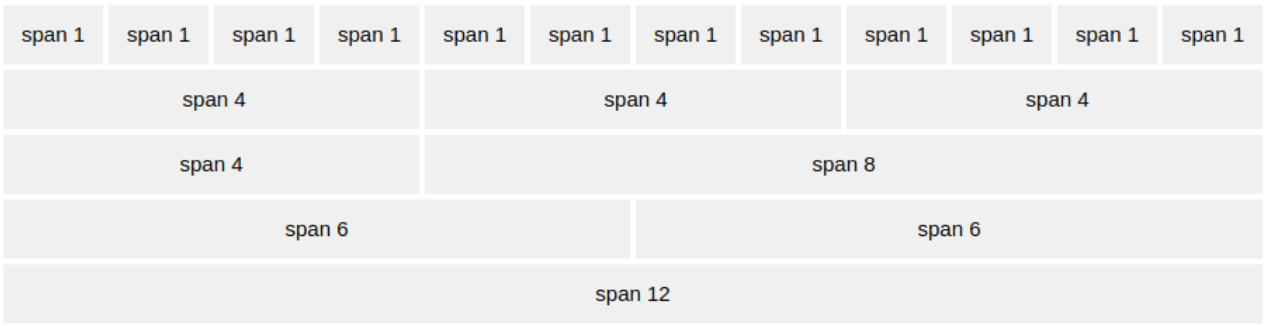


Fig: 2.9

Grid Classes

The Bootstrap grid system has four classes:

xs (for phones - screens less than 768px wide)

sm (for tablets - screens equal to or greater than 768px wide)

md (for small laptops - screens equal to or greater than 992px wide)

lg (for laptops and desktops - screens equal to or greater than 1200px wide)

The classes above can be combined to create more dynamic and flexible layouts.

Tip: Each class scales up, so if you wish to set the same widths for xs and sm, you only need to specify xs.

Grid System Rules

Some Bootstrap grid system rules:

Rows must be placed within a .container (fixed-width) or .container-fluid (full-width) for proper alignment and padding

Use rows to create horizontal groups of columns

Content should be placed within columns, and only columns may be immediate children of rows

Predefined classes like .row and .col-sm-4 are available for quickly making grid layouts

Columns create gutters (gaps between column content) via padding. That padding is offset in rows for the first and last column via negative margin on .rows

Grid columns are created by specifying the number of 12 available columns you wish to span. For example, three equal columns would use three .col-sm-4

Column widths are in percentage, so they are always fluid and sized relative to their parent element

The following is a basic structure of a Bootstrap grid:

```
<div class="container">
  <div class="row">
    <div class="col-*-*"></div>
    <div class="col-*-*"></div>
  </div>
  <div class="row">
    <div class="col-*-*"></div>
    <div class="col-*-*"></div>
    <div class="col-*-*"></div>
  </div>
  <div class="row">
    ...
  </div>
</div>
```

Chapter 3: JavaScript

3.1 Introduction

JavaScript is the programming language of HTML and the Web. It is one of the 3 languages all web developers must learn:

1. HTML to define the content of web pages
2. CSS to specify the layout of web pages
3. JavaScript to program the behavior of web pages

Web pages are not the only place where JavaScript is used. Many desktop and server programs use JavaScript. Node.js is the best known. Some databases, like MongoDB and CouchDB, also use JavaScript as their programming language.

The <script> Tag

In HTML, JavaScript code must be inserted between <script> and </script> tags.

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

Writing into an HTML element, using innerHTML.

Writing into the HTML output using document.write().

Writing into an alert box, using window.alert().

Writing into the browser console, using console.log().

JavaScript Can Change HTML Content

One of many JavaScript HTML methods is getElementById().

This example uses the method to "find" an HTML element (with id="demo") and changes the element content (innerHTML) to "Hello JavaScript":

3.2 Programs

A computer program is a list of "instructions" to be "executed" by a computer.

In a programming language, these programming instructions are called statements.

A JavaScript program is a list of programming statements.

The JavaScript syntax defines two types of values: Fixed values and variable values.

Fixed values are called literals. Variable values are called variables.

Variables in javaScript are defined as:

var x = 5;

3.3 Functions

A JavaScript function is a block of code designed to perform a particular task. A JavaScript function is executed when "something" invokes it (calls it).

A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:
(parameter1, parameter2, ...) The code to be executed, by the function, is placed inside curly brackets: {}

Objects

Objects are variables too. But objects can contain many values.You define (and create) a JavaScript object with an object literal:
Example
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

HTML events are "things" that happen to HTML elements.

3.4 HTML Events

What are HTML Events

An HTML event can be something the browser does, or something a user does. Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Often, when events happen, you may want to do something. JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

With single quotes:

<element event='some JavaScript'>

With double quotes:

<element event="some JavaScript">

In the following example, an onclick attribute (with code), is added to a button element:

```
<button onclick="document.getElementById('demo').innerHTML = Date()">The time is?</button>
```

Some HTML Events

Event	Description
onchange	An HTML element has been changed
onclick	The user click an HTML Element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

3.5 What can JavaScript Do?

Event handlers can be used to handle, and verify, user input, user actions, and browser actions:

- Things that should be done every time a page loads
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data
- And more....

Many different methods can be used to let JavaScript work with events:

- HTML event attributes can execute JavaScript code directly.
- HTML event attributes can call JavaScript functions.
- You can assign your own event handler functions to HTML elements

You can prevent events from being sent or being handled.

Chapter 4: Python

4.1 Introduction

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). This tutorial gives enough understanding on Python programming language.

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- Python is a Beginner's Language − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Python Features

Python's features include −

- Easy-to-learn − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- Easy-to-read − Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain − Python's source code is fairly easy-to-maintain.
- A broad standard library − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

- Interactive Mode − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- Portable − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases − Python provides interfaces to all major commercial databases.
- GUI Programming − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- Scalable − Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below −

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

4.2 Variable Type

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.

For example −

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string
```

Multiple Assignment

Python allows you to assign a single value to several variables simultaneously. For example −

```
a = b = c = 1
```

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables. For example −

```
a,b,c = 1,2,"john"
```

Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

Standard Data Types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types −

- Numbers

- String
- List
- Tuple
- Dictionary

Python Numbers

Number data types store numeric values. Number objects are created when you assign a value to them. For example −

```
var1 = 1
var2 = 10
```

You can also delete the reference to a number object by using the del statement. The syntax of the del statement is −

```
del var1[,var2[,var3[....,varN]]]
```

You can delete a single object or multiple objects by using the del statement. For example −

```
del var
del var_a, var_b
```

Python supports four different numerical types −

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

Python Strings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator. For example −

```
str = 'Hello World!'
```

Python Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator. For example −

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']
```

Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as read-only lists. For example −

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
```

Python Dictionary

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]). For example −

```
dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"
```

Data Type Conversion

Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function.

There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

4.3 Operators

Operators are the constructs which can manipulate the value of operands.

Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator.

Types of Operator

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

4.4 Decision Making

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.

Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.

Following is the general form of a typical decision making structure found in most of the programming languages −

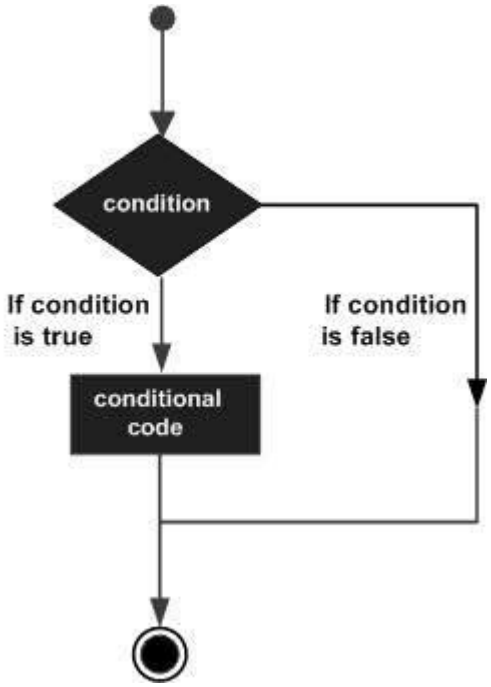


Fig: 4.4

Python programming language assumes any non-zero and non-null values as TRUE, and if it is either zero or null, then it is assumed as FALSE value.

Python programming language provides following types of decision making statements. Click the following links to check their detail.

Sr.No.	Statement & Description
1	if statements An if statement consists of a boolean expression followed by one or more statements.
2	if...else statements An if statement can be followed by an optional else statement, which executes when the boolean expression is FALSE.
3	nested if statements You can use one if or else if statement inside another if or else ifstatement(s).

4.5 Loops

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement −

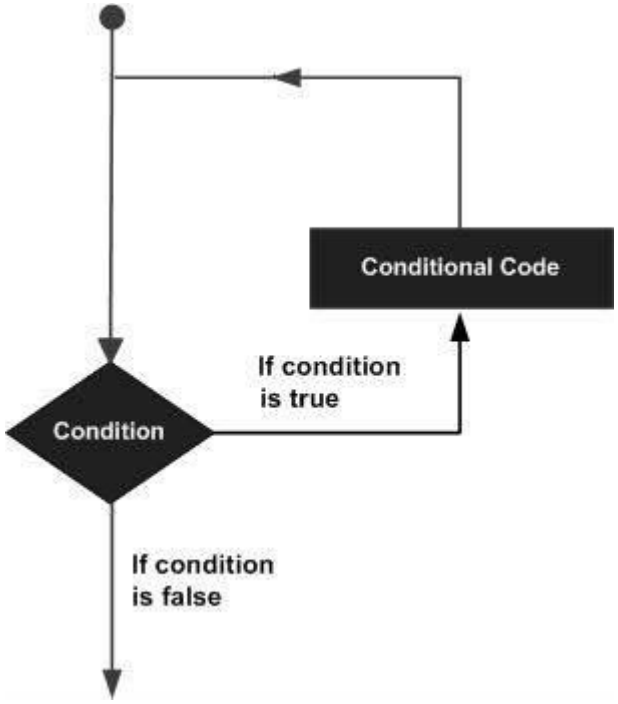


Fig: 4.5

Python programming language provides following types of loops to handle looping requirements.

Sr.No.	Loop Type & Description
1	while loop Repeats a statement or group of statements while a given condition is TRUE. It

	tests the condition before executing the loop body.
2	for loop Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	nested loops You can use one or more loop inside any another while, for or do..while loop.

4.6 Functions

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called *user-defined functions.*

Defining a Function

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword def followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Syntax

```
def functionname( parameters ):
```

```
   "function_docstring"
   function_suite
   return [expression]
```

By default, parameters have a positional behavior and you need to inform them in the same order that they were defined.

Example

The following function takes a string as input parameter and prints it on standard screen.

```
def printme( str ):
   "This prints a passed string into this function"
   print str
   return
```

4.7 Classes and Objects

Python has been an object-oriented language since it existed. Because of this, creating and using classes and objects are downright easy. This chapter helps you become an expert in using Python's object-oriented programming support.

If you do not have any previous experience with object-oriented (OO) programming, you may want to consult an introductory course on it or at least a tutorial of some sort so that you have a grasp of the basic concepts.

Creating Classes

The *class* statement creates a new class definition. The name of the class immediately follows the keyword *class* followed by a colon as follows −

```
class ClassName:
   'Optional class documentation string'
   class_suite
```

- The class has a documentation string, which can be accessed via *ClassName.__doc__*.
- The *class_suite* consists of all the component statements defining class members, data attributes and functions.

Creating Instance Objects

To create instances of a class, you call the class using class name and pass in whatever arguments its *__init__* method accepts.

```
"This would create first object of Employee class"
```

```
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
```

Accessing Attributes

You access the object's attributes using the dot operator with object. Class variable would be accessed using class name as follows −

```
emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" % Employee.empCount
```

4.8 Regular Expressions

A *regular expression* is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. Regular expressions are widely used in UNIX world.

The module re provides full support for Perl-like regular expressions in Python. The re module raises the exception re.error if an error occurs while compiling or using a regular expression.

We would cover two important functions, which would be used to handle regular expressions. But a small thing first: There are various characters, which would have special meaning when they are used in regular expression. To avoid any confusion while dealing with regular expressions, we would use Raw Strings as r'expression'.

The *match* Function

This function attempts to match RE *pattern* to *string* with optional *flags*.

Here is the syntax for this function −

```
re.match(pattern, string, flags=0)
```

Here is the description of the parameters −

Sr.No.	Parameter & Description
1	pattern This is the regular expression to be matched.

2	string This is the string, which would be searched to match the pattern at the beginning of string.
3	flags You can specify different flags using bitwise OR (|). These are modifiers, which are listed in the table below.

The *re.match* function returns a match object on success, None on failure. We use*group(num)* or *groups()* function of match object to get matched expression.

Sr.No.	Match Object Method & Description
1	group(num=0) This method returns entire match (or specific subgroup num)
2	groups() This method returns all matching subgroups in a tuple (empty if there weren't any)

Chapter 5: Django

5.1 Introduction

Django is a web development framework that assists in building and maintaining quality web applications. Django helps eliminate repetitive tasks making the development process an easy and time saving experience. This tutorial gives a complete understanding of Django.

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Django makes it easier to build better web apps quickly and with less code.

Note − Django is a registered trademark of the Django Software Foundation, and is licensed under BSD License.

Advantages of Django

Here are few advantages of using Django which can be listed out here −

- Object-Relational Mapping (ORM) Support − Django provides a bridge between the data model and the database engine, and supports a large set of database systems including MySQL, Oracle, Postgres, etc. Django also supports NoSQL database through Django-nonrel fork. For now, the only NoSQL databases supported are MongoDB and google app engine.
- Multilingual Support − Django supports multilingual websites through its built-in internationalization system. So you can develop your website, which would support multiple languages.
- Framework Support − Django has built-in support for Ajax, RSS, Caching and various other frameworks.
- Administration GUI − Django provides a nice ready-to-use user interface for administrative activities.
- Development Environment − Django comes with a lightweight web server to facilitate end-to-end application development and testing.

5.2 Overview

As you already know, Django is a Python web framework. And like most modern framework, Django supports the MVC pattern. First let's see what is the Model-View-Controller (MVC) pattern, and then we will look at Django's specificity for the Model-View-Template (MVT) pattern.

MVC Pattern

When talking about applications that provides UI (web or desktop), we usually talk about MVC architecture. And as the name suggests, MVC pattern is based on three components: Model, View, and Controller. Check our MVC tutorial here to know more.

DJANGO MVC - MVT Pattern

The Model-View-Template (MVT) is slightly different from MVC. In fact the main difference between the two patterns is that Django itself takes care of the Controller part (Software Code that controls the interactions between the Model and View), leaving us with the template. The template is a HTML file mixed with Django Template Language (DTL).

The following diagram illustrates how each of the components of the MVT pattern interacts with each other to serve a user request −

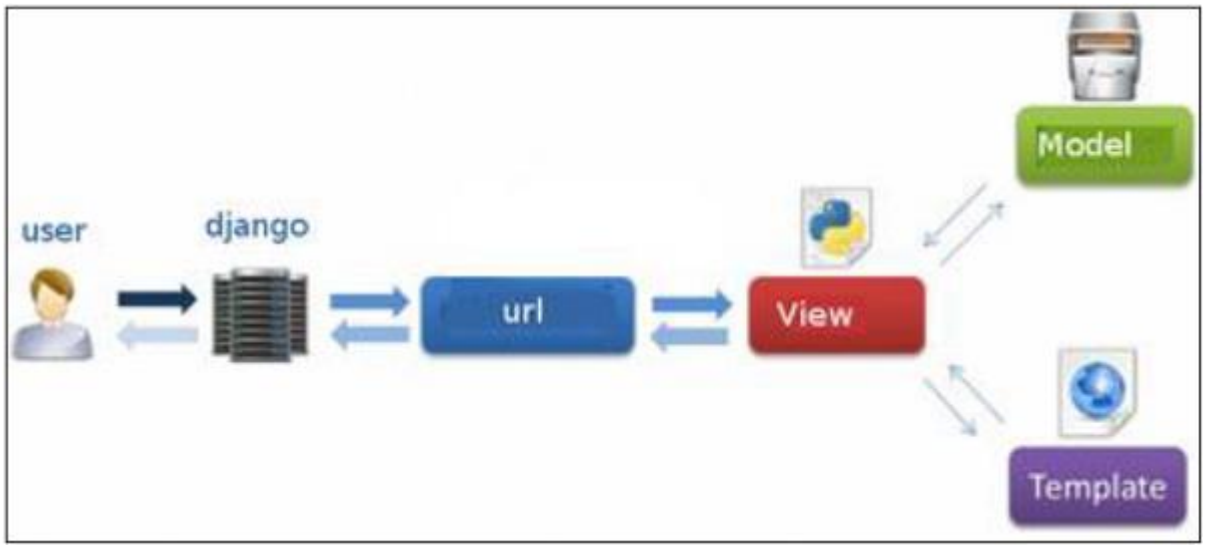


Fig: 5.2

The developer provides the Model, the view and the template then just maps it to a URL and Django does the magic to serve it to the user.

5.3 Creating a Project

Now that we have installed Django, let's start using it. In Django, every web app you want to create is called a project; and a project is a sum of applications. An application is a set of code files relying on the MVT pattern. As example let's say we want to build a website, the website is our project and, the forum, news, contact engine are applications. This structure makes it easier to move an application between projects since every application is independent.

Create a Project

Whether you are on Windows or Linux, just get a terminal or a cmd prompt and navigate to the place you want your project to be created, then use this code −

```
$ django-admin startproject myproject
```

This will create a "myproject" folder with the following structure −

```
myproject/
   manage.py
   myproject/
      __init__.py
      settings.py
      urls.py
      wsgi.py
```

The Project Structure

The “myproject” folder is just your project container, it actually contains two elements −

- manage.py − This file is kind of your project local django-admin for interacting with your project via command line (start the development server, sync db...). To get a full list of command accessible via manage.py you can use the code −

```
$ python manage.py help
```

- The “myproject” subfolder − This folder is the actual python package of your project. It contains four files −
    - __init__.py − Just for python, treat this folder as package.
    - settings.py − As the name indicates, your project settings.

- urls.py − All links of your project and the function to call. A kind of ToC of your project.
- wsgi.py − If you need to deploy your project over WSGI.

Setting Up Your Project

Your project is set up in the subfolder myproject/settings.py. Following are some important options you might need to set −

```
DEBUG = True
```

This option lets you set if your project is in debug mode or not. Debug mode lets you get more information about your project's error. Never set it to 'True' for a live project. However, this has to be set to 'True' if you want the Django light server to serve static files. Do it only in the development mode.

```
DATABASES = {
   'default': {
      'ENGINE': 'django.db.backends.sqlite3',
      'NAME': 'database.sql',
      'USER': '',
      'PASSWORD': '',
      'HOST': '',
      'PORT': '',
   }
}
```

Database is set in the 'Database' dictionary. The example above is for SQLite engine. As stated earlier, Django also supports −

- MySQL (django.db.backends.mysql)
- PostGreSQL (django.db.backends.postgresql_psycopg2)
- Oracle (django.db.backends.oracle) and NoSQL DB
- MongoDB (django_mongodb_engine)

Before setting any new engine, make sure you have the correct db driver installed.

You can also set others options like: TIME_ZONE, LANGUAGE_CODE, TEMPLATE…

Now that your project is created and configured make sure it's working −

```
$ python manage.py runserver
```

You will get something like the following on running the above code −

```
Validating models...

0 errors found
September 03, 2015 - 11:41:50
Django version 1.6.11, using settings 'myproject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

5.4 Creating Views

A view function, or “view” for short, is simply a Python function that takes a web request and returns a web response. This response can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image, etc. Example: You use view to create web pages, note that you need to associate a view to a URL to see it as a web page.

In Django, views have to be created in the app views.py file.

Simple View

We will create a simple view in myapp to say "welcome to my app!"

See the following view −

```
from django.http import HttpResponse

def hello(request):
   text = """<h1>welcome to my app !</h1>"""
   return HttpResponse(text)
```

In this view, we use HttpResponse to render the HTML (as you have probably noticed we have the HTML hard coded in the view). To see this view as a page we just need to map it to a URL (this will be discussed in an upcoming chapter).

We used HttpResponse to render the HTML in the view before. This is not the best way to render pages. Django supports the MVT pattern so to make the precedent view, Django - MVT like, we will need −

A template: myapp/templates/hello.html

And now our view will look like −

```
from django.shortcuts import render

def hello(request):
```

```
   return render(request, "myapp/template/hello.html", {})
```

Views can also accept parameters −

```
from django.http import HttpResponse

def hello(request, number):
   text = "<h1>welcome to my app number %s!</h1>"% number
   return HttpResponse(text)
```

When linked to a URL, the page will display the number passed as a parameter. Note that the parameters will be passed via the URL (discussed in the next chapter).

5.5 URL Mapping

Now that we have a working view as explained in the previous chapters. We want to access that view via a URL. Django has his own way for URL mapping and it's done by editing your project url.py file (myproject/url.py). The url.py file looks like −

```
from django.conf.urls import patterns, include, url
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
   #Examples
   #url(r'^$', 'myproject.view.home', name = 'home'),
   #url(r'^blog/', include('blog.urls')),

   url(r'^admin', include(admin.site.urls)),
)
```

When a user makes a request for a page on your web app, Django controller takes over to look for the corresponding view via the url.py file, and then return the HTML response or a 404 not found error, if not found. In url.py, the most important thing is the "urlpatterns" tuple. It’s where you define the mapping between URLs and views. A mapping is a tuple in URL patterns like −

```
from django.conf.urls import patterns, include, url
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
   #Examples
   #url(r'^$', 'myproject.view.home', name = 'home'),
```

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

   url(r'^admin', include(admin.site.urls)),
   url(r'^hello/', 'myapp.views.hello', name = 'hello'),
)
```

The marked line maps the URL "/home" to the hello view created in myapp/view.py file. As you can see above a mapping is composed of three elements −

- The pattern − A regexp matching the URL you want to be resolved and map. Everything that can work with the python 're' module is eligible for the pattern (useful when you want to pass parameters via url).
- The python path to the view − Same as when you are importing a module.
- The name − In order to perform URL reversing, you'll need to use named URL patterns as done in the examples above. Once done, just start the server to access your view via :http://127.0.0.1/hello

Sending Parameters to Views

We now know how to map URL, how to organize them, now let us see how to send parameters to views. A classic sample is the article example (you want to access an article via "/articles/article_id").

Passing parameters is done by capturing them with the regexp in the URL pattern. If we have a view like the following one in "myapp/view.py"

```
from django.shortcuts import render
from django.http import HttpResponse

def hello(request):
   return render(request, "hello.html", {})

def viewArticle(request, articleId):
   text = "Displaying article Number : %s"%articleId
   return HttpResponse(text)
```

We want to map it in myapp/url.py so we can access it via "/myapp/article/articleId", we need the following in "myapp/url.py" –

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

urlpatterns = patterns('myapp.views',
   url(r'^hello/', 'hello', name = 'hello'),
   url(r'^morning/', 'morning', name = 'morning'),
   url(r'^article/(\d+)/', 'viewArticle', name = 'article'),)
```

When Django will see the url: "/myapp/article/42" it will pass the parameters '42' to the viewArticle view, and in your browser you should get the following result −

Fig: 5.5.1

Note that the order of parameters is important here. Suppose we want the list of articles of a month of a year, let's add a viewArticles view. Our view.py becomes −

```
from django.shortcuts import render
from django.http import HttpResponse

def hello(request):
   return render(request, "hello.html", {})

def viewArticle(request, articleId):
   text = "Displaying article Number : %s"%articleId
   return HttpResponse(text)

def viewArticle(request, month, year):
   text = "Displaying articles of : %s/%s"%(year, month)
   return HttpResponse(text)
```

The corresponding url.py file will look like −

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

```
urlpatterns = patterns('myapp.views',
   url(r'^hello/', 'hello', name = 'hello'),
   url(r'^morning/', 'morning', name = 'morning'),
   url(r'^article/(\d+)/', 'viewArticle', name = 'article'),
   url(r'^articles/(\d{2})/(\d{4})', 'viewArticles', name =
'articles'),)
```

Now when you go to "/myapp/articles/12/2006/" you will get 'Displaying articles of: 2006/12' but if you reverse the parameters you won't get the same result.

Fig: 5.5.2

To avoid that, it is possible to link a URL parameter to the view parameter. For that, our url.py will become −

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

urlpatterns = patterns('myapp.views',
   url(r'^hello/', 'hello', name = 'hello'),
   url(r'^morning/', 'morning', name = 'morning'),
   url(r'^article/(\d+)/', 'viewArticle', name = 'article'),
   url(r'^articles/(?P\d{2})/(?P\d{4})', 'viewArticles', name =
'articles'),)
```

5.6 Template System

Django makes it possible to separate python and HTML, the python goes in views and HTML goes in templates. To link the two, Django relies on the render function and the Django Template language.

The Render Function

This function takes three parameters −

- Request − The initial request.
- The path to the template − This is the path relative to the TEMPLATE_DIRS option in the project settings.py variables.
- Dictionary of parameters − A dictionary that contains all variables needed in the template. This variable can be created or you can use locals() to pass all local variable declared in the view.

Django Template Language (DTL)

Django's template engine offers a mini-language to define the user-facing layer of the application.

Displaying Variables

A variable looks like this: {{variable}}. The template replaces the variable by the variable sent by the view in the third parameter of the render function. Let's change our hello.html to display today's date −

hello.html

```
<html>

   <body>
      Hello World!!!<p>Today is {{today}}</p>
   </body>

</html>
```

Then our view will change to −

```
def hello(request):
   today = datetime.datetime.now().date()
   return render(request, "hello.html", {"today" : today})
```

We will now get the following output after accessing the URL/myapp/hello −

```
Hello World!!!
Today is Sept. 11, 2015
```

As you have probably noticed, if the variable is not a string, Django will use the __str__ method to display it; and with the same principle you can access an object attribute just

like you do it in Python. For example: if we wanted to display the date year, my variable would be: {{today.year}}.

Filters

They help you modify variables at display time. Filters structure looks like the following: {{var|filters}}.

Some examples −

- {{string|truncatewords:80}} − This filter will truncate the string, so you will see only the first 80 words.
- {{string|lower}} − Converts the string to lowercase.
- {{string|escape|linebreaks}} − Escapes string contents, then converts line breaks to tags.

You can also set the default for a variable.

Tags

Tags lets you perform the following operations: if condition, for loop, template inheritance and more.

Tag if

Just like in Python you can use if, else and elif in your template −

```
<html>
   <body>

      Hello World!!!<p>Today is {{today}}</p>
      We are
      {% if today.day == 1 %}

      the first day of month.
      {% elif today.day == 30 %}

      the last day of month.
      {% else %}

      I don't know.
   {%endif%}

   </body>
</html>
```

In this new template, depending on the date of the day, the template will render a certain value.

Tag for

Just like 'if', we have the 'for' tag, that works exactly like in Python. Let's change our hello view to transmit a list to our template −

```
def hello(request):
   today = datetime.datetime.now().date()

   daysOfWeek = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
   return render(request, "hello.html", {"today" : today,
"days_of_week" : daysOfWeek})
```

The template to display that list using {{ for }} −

```
<html>
   <body>

      Hello World!!!<p>Today is {{today}}</p>
      We are
      {% if today.day == 1 %}

      the first day of month.
      {% elif today.day == 30 %}

      the last day of month.
      {% else %}

      I don't know.
      {%endif%}

      <p>
         {% for day in days_of_week %}
         {{day}}
      </p>

      {% endfor %}

   </body>
</html>
```

And we should get something like −

```
Hello World!!!
Today is Sept. 11, 2015
```

```
We are I don't know.
Mon
Tue
Wed
Thu
Fri
Sat
Sun
```

Block and Extend Tags

A template system cannot be complete without template inheritance. Meaning when you are designing your templates, you should have a main template with holes that the child's template will fill according to his own need, like a page might need a special css for the selected tab.

Let’s change the hello.html template to inherit from a main_template.html.

main_template.html

```
<html>
   <head>

      <title>
         {% block title %}Page Title{% endblock %}
      </title>

   </head>

   <body>

      {% block content %}
         Body content
      {% endblock %}

   </body>
</html>
```

hello.html

```
{% extends "main_template.html" %}
{% block title %}My Hello Page{% endblock %}
{% block content %}

Hello World!!!<p>Today is {{today}}</p>
We are
{% if today.day == 1 %}
```

```
the first day of month.
{% elif today.day == 30 %}

the last day of month.
{% else %}

I don't know.
{%endif%}

<p>
   {% for day in days_of_week %}
   {{day}}
</p>

{% endfor %}
{% endblock %}
```

In the above example, on calling /myapp/hello we will still get the same result as before but now we rely on extends and block to refactor our code −

In the main_template.html we define blocks using the tag block. The title block will contain the page title and the content block will have the page main content. In home.html we use extends to inherit from the main_template.html then we fill the block define above (content and title).

Comment Tag

The comment tag helps to define comments into templates, not HTML comments, they won't appear in HTML page. It can be useful for documentation or just commenting a line of code.

5.7 Models

A model is a class that represents table or collection in our DB, and where every attribute of the class is a field of the table or collection. Models are defined in the app/models.py (in our example: myapp/models.py)

Creating a Model

Following is a Dreamreal model created as an example −

```
from django.db import models
```

```
class Dreamreal(models.Model):

   website = models.CharField(max_length = 50)
   mail = models.CharField(max_length = 50)
   name = models.CharField(max_length = 50)
   phonenumber = models.IntegerField()

   class Meta:
      db_table = "dreamreal"
```

Every model inherits from django.db.models.Model.

Our class has 4 attributes (3 CharField and 1 Integer), those will be the table fields.

The Meta class with the db_table attribute lets us define the actual table or collection name. Django names the table or collection automatically: myapp_modelName. This class will let you force the name of the table to what you like.

There is more field's type in django.db.models, you can learn more about them on https://docs.djangoproject.com/en/1.5/ref/models/fields/#field-types

After creating your model, you will need Django to generate the actual database −

```
$python manage.py syncdb
```

Manipulating Data (CRUD)

Let's create a "crudops" view to see how we can do CRUD operations on models. Our myapp/views.py will then look like −

myapp/views.py

```
from myapp.models import Dreamreal
from django.http import HttpResponse

def crudops(request):
   #Creating an entry

   dreamreal = Dreamreal(
      website = "www.polo.com", mail = "sorex@polo.com",
      name = "sorex", phonenumber = "002376970"
   )

   dreamreal.save()

   #Read ALL entries
   objects = Dreamreal.objects.all()
```

```
   res ='Printing all Dreamreal entries in the DB : <br>'

   for elt in objects:
      res += elt.name+"<br>"

   #Read a specific entry:
   sorex = Dreamreal.objects.get(name = "sorex")
   res += 'Printing One entry <br>'
   res += sorex.name

   #Delete an entry
   res += '<br> Deleting an entry <br>'
   sorex.delete()

   #Update
   dreamreal = Dreamreal(
      website = "www.polo.com", mail = "sorex@polo.com",
      name = "sorex", phonenumber = "002376970"
   )

   dreamreal.save()
   res += 'Updating entry<br>'

   dreamreal = Dreamreal.objects.get(name = 'sorex')
   dreamreal.name = 'thierry'
   dreamreal.save()

   return HttpResponse(res)
```

Other Data Manipulation

Let's explore other manipulations we can do on Models. Note that the CRUD operations were done on instances of our model, now we will be working directly with the class representing our model.

Let's create a 'datamanipulation' view in myapp/views.py

```
from myapp.models import Dreamreal
from django.http import HttpResponse

def datamanipulation(request):
   res = ''

   #Filtering data:
   qs = Dreamreal.objects.filter(name = "paul")
   res += "Found : %s results<br>"%len(qs)
```

```
   #Ordering results
   qs = Dreamreal.objects.order_by("name")

   for elt in qs:
      res += elt.name + '<br>'

   return HttpResponse(res)
```

Linking Models

Django ORM offers 3 ways to link models −

One of the first case we will see here is the one-to-many relationships. As you can see in the above example, Dreamreal company can have multiple online websites. Defining that relation is done by using django.db.models.ForeignKey −

myapp/models.py

```
from django.db import models

class Dreamreal(models.Model):
   website = models.CharField(max_length = 50)
   mail = models.CharField(max_length = 50)
   name = models.CharField(max_length = 50)
   phonenumber = models.IntegerField()
   online = models.ForeignKey('Online', default = 1)

   class Meta:
      db_table = "dreamreal"

class Online(models.Model):
      domain = models.CharField(max_length = 30)

   class Meta:
      db_table = "online"
```

As you can see in our updated myapp/models.py, we added the online model and linked it to our Dreamreal model.

Let's check how all of this is working via manage.py shell −

First let’s create some companies (Dreamreal entries) for testing in our Django shell −

```
$python manage.py shell

>>> from myapp.models import Dreamreal, Online
```

```
>>> dr1 = Dreamreal()
>>> dr1.website = 'company1.com'
>>> dr1.name = 'company1'
>>> dr1.mail = 'contact@company1'
>>> dr1.phonenumber = '12345'
>>> dr1.save()
>>> dr2 = Dreamreal()
>>> dr1.website = 'company2.com'
>>> dr2.website = 'company2.com'
>>> dr2.name = 'company2'
>>> dr2.mail = 'contact@company2'
>>> dr2.phonenumber = '56789'
>>> dr2.save()
```

Now some hosted domains −

```
>>> on1 = Online()
>>> on1.company = dr1
>>> on1.domain = "site1.com"
>>> on2 = Online()
>>> on2.company = dr1
>>> on2.domain = "site2.com"
>>> on3 = Online()
>>> on3.domain = "site3.com"
>>> dr2 = Dreamreal.objects.all()[2]
>>> on3.company = dr2
>>> on1.save()
>>> on2.save()
>>> on3.save()
```

Accessing attribute of the hosting company (Dreamreal entry) from an online domain is simple −

```
>>> on1.company.name
```

And if we want to know all the online domain hosted by a Company in Dreamreal we will use the code −

```
>>> dr1.online_set.all()
```

To get a QuerySet, note that all manipulating method we have seen before (filter, all, exclude, order_by....)

You can also access the linked model attributes for filtering operations, let's say you want to get all online domains where the Dreamreal name contains 'company' −

```
>>> Online.objects.filter(company__name__contains = 'company'
```

Note − That kind of query is just supported for SQL DB. It won’t work for non-relational DB where joins doesn’t exist and there are two '_'.

But that's not the only way to link models, you also have OneToOneField, a link that guarantees that the relation between two objects is unique. If we used the OneToOneField in our example above, that would mean for every Dreamreal entry only one Online entry is possible and in the other way to.

Chapter 6: Summer Training Project overview
AKADEMIA

6.1 An overview of the project

Our desired result of the **AKADEMIA** is an integrated web-based system combining several systems in use. It will include the following four major components:

Notes component

This component is basically associated with course notes. The student can upload his/her notes to share with other students. For every subject/course in database, the student can upload notes according to year (batch-year). If any student who has just get into the semester and want to get notes of their seniors, user can go to that batch-year and get the notes for that particular course.

Question Papers component

This component is basically associated with course exams question papers. Generally, at the time of sessional and finals, every user wants to know what questions have been there in previous year papers and they ask on WhatsApp or any other chatting group, sometimes they don't find those question papers. So, college faculty and even students can upload question papers in this component year (batch-year) wise.

Study Material component

his component is basically associated with course related study material. During lectures teachers teach students using PPT's, PDFs and books. But sometimes teachers forget to share the study material with student and to prevent this the teacher can upload the study material into this component. Instead of sharing study material again and again with new student that come every year the student s can find all the study material at just one place. Students can browse study material teacher and course wise at just one place.

Calendar component

This component is basically associated with events like holidays, quiz, cultural events, fests etc. The events can be added by only college faculty and admin. Student can just view the events.

Besides these four major components, there is another component separate from all the above four. This component is called core data which contains information such as courses, batch, teachers, students and semesters. This data is used in the above four components.

6.2 Django framework development process

To build such a complicated web system, we need three major parts for each component: database, user interface and the functions to interact in between. Django framework provides sufficient functionalities to implement these three parts. Corresponding to database, user interface and functions in between, Django has model, template and view components to deal with each part respectively. Django's model component helps programmer to define and maintain tables in the database, while its template component helps to write html files using a combination of both html syntax and Django syntax. For those functions in between, Django provides a view component which reads the input from user interface and makes corresponding changes in the database.

6.3 Detail Overview of project

Our project is divided into 4 modules –

1. Profile
2. Dashboard
3. Study Corner
4. Calendar

Let’s get started what they offer to a user and get to know about them in much more detail.

Profile Section

Every website or mobile-app we visit in our daily life has profile page. Keeping this in mind we have designed a profile page for every user who has an account on our **AKADEMIA** web-app. The users are classified as staff or teacher or student. But the profile page is meaningful for only students and teachers as admin has no use of profile

page. Admin is just a maintainer that can maintain the records of our database and we will describe what all admin can do in later sections.

A student profile page will show basic information like name of student, father name, contact, semester, batch-year etc.

Whereas, a teacher profile page will show information like name of teacher, designation, education qualification, specialization area etc.

With every profile information user can edit the information if any wrong information is there and can also change the user profile pic.

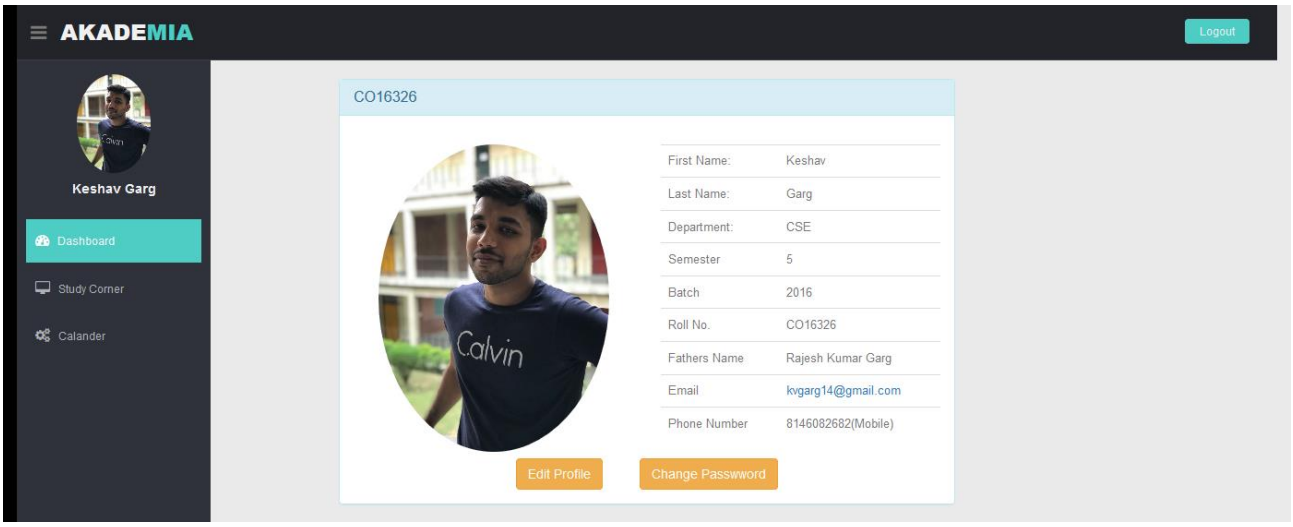


Fig 6.3.1 Student Profile Page

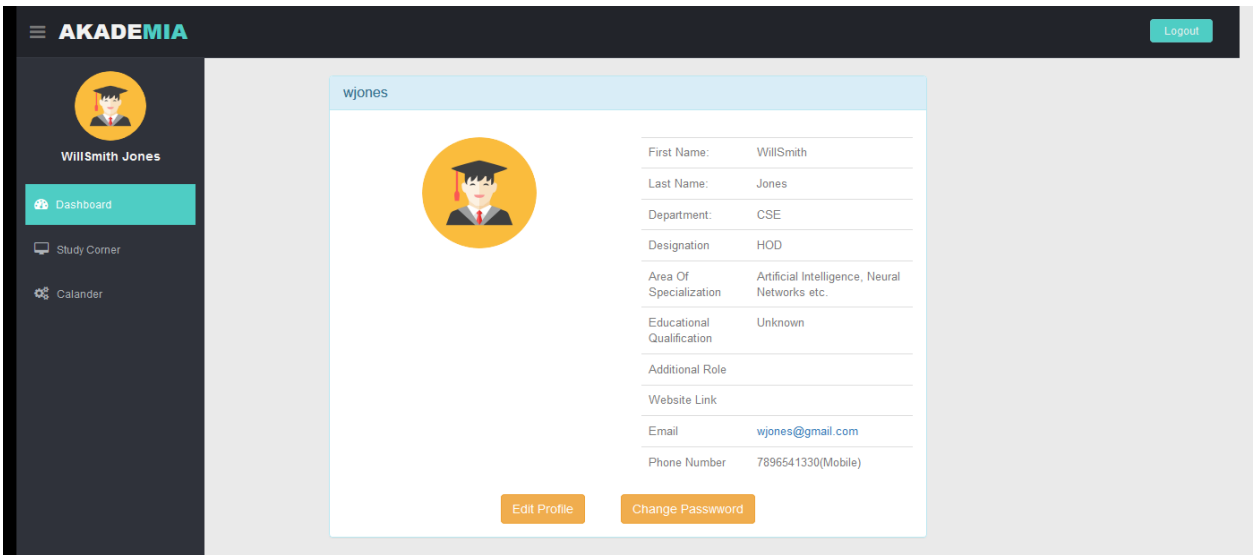


Fig 6.3.2 Teacher Profile Page

Dashboard Section

When the user login with its login credentials, the user is redirected to the **AKADEMIA** Dashboard page where user can navigate to different modules of the application. According to the user type, the modules in dashboard are displayed. A student and teacher user can only view the Study corner and calendar. But student and teacher user have different permissions to access the modules. A student user can upload any study material whereas teacher can upload it. Like this, each module has user-specific permissions to perform functions.

An admin dashboard has all the modules i.e. an admin has all the permissions to access all the data and can manipulate the data. The admin can even maintain the users.

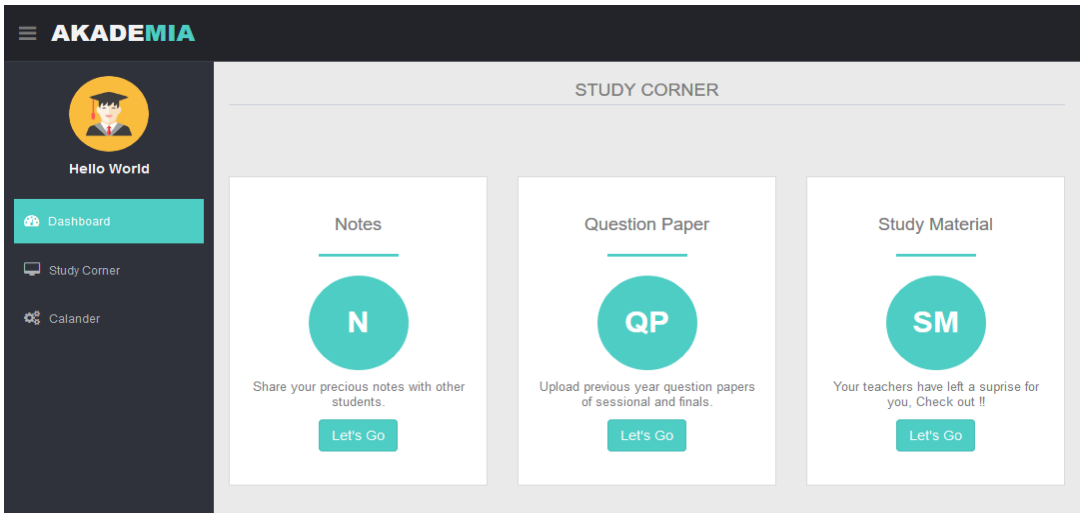


Fig 6.3.3 Student Dashboard

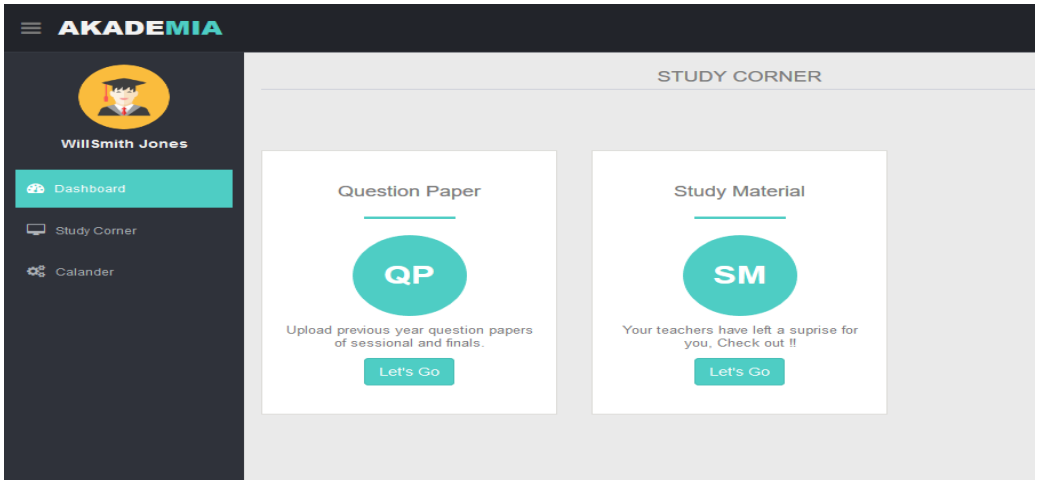


Fig 6.3.4 Teacher Dashboard

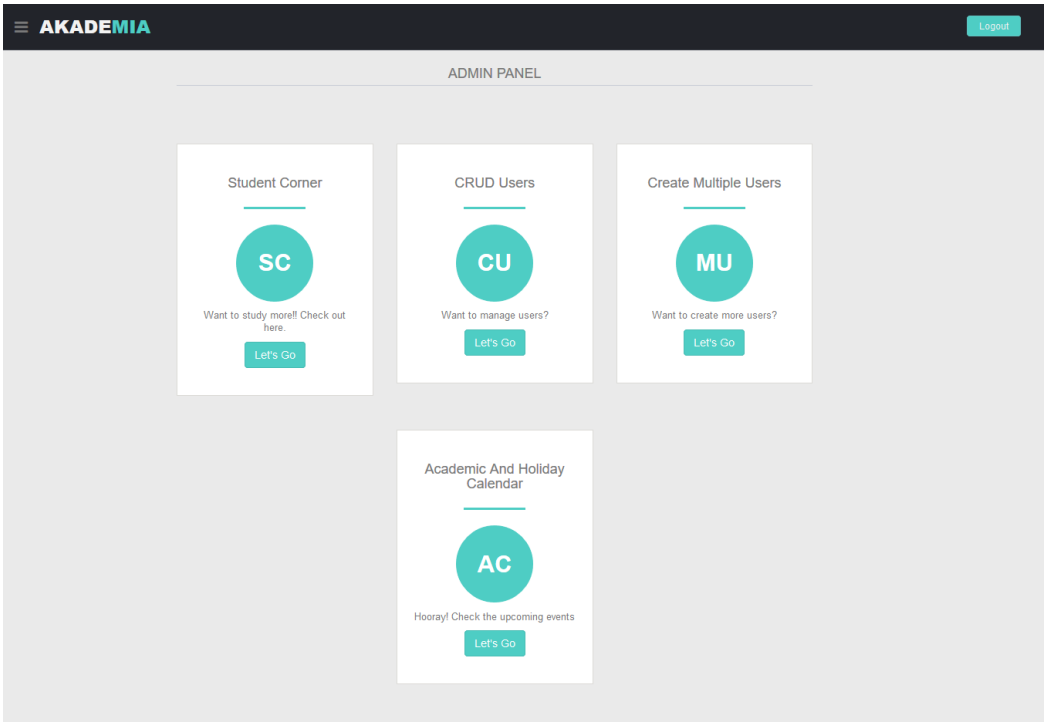


Fig 6.3.5 Admin Dashboard

Study Corner Section

Our project mainly focuses on this section. This is that module where students and teacher play a vital role. This section is comprised of 3 sub-sections, that are, Notes, Question Papers and Study Material. The student and admin can view all the 3 sub-sections as mentioned above but the teacher can view only Study Material and Question Papers Section. But each user-type has permissions to access each function of each sub-section i.e. not every user can upload study material or question papers or notes. Let's get to know which user has what function accessibility.

Notes Sub-Section

In this section, the user can share lecture notes with other users. Notes can only be uploaded by either student or admin but not by teacher. Teacher can't even view this module. This section is only visible to students and admin. As admin has also no benefit in viewing this but since admin has all permissions so admin manages all the notes.

The user can navigate to course notes by following a hierarchy:

Batch-Year -> Semester -> Subjects -> Files

Following this hierarchy, student can view any year or semester notes. Clicking on any file downloads the file for user.

A student can upload notes by clicking on “upload” button where all notes are displayed for a semester. The user just needs to browse the file, select the file type i.e. whether it is notes or question paper and can also specify the file URL incase the user download from any internet source. On clicking “submit” the file is uploaded in database, can be viewed by other users as well.

Question Papers Sub-Section

In this section, the user can share previous year sessional and final exams question paper with other users. Question Papers can be uploaded by student or admin or by teacher. This section is only visible to all the users.

The user can navigate to any year question papers by following the same hierarchy as for course notes:

Batch-Year -> Semester -> Subjects -> Files

Following this hierarchy, student can view any year or semester question papers. Clicking on any file downloads the file for user.

Any user can upload question papers by clicking on “upload” button where all question papers are displayed for a semester. The user just needs to browse the file, select the file type i.e. whether it is notes or question paper and can also specify the file URL incase the user download from any internet source. On clicking “submit” the file is uploaded in database, can be viewed by other users as well.

Study Material

In this section, the teacher or admin can share course study material with other users (students). Study Material can be uploaded by student or admin or by teacher. This section is only visible to all the users.

The user can navigate to any study material by following the hierarchy:

Subjects -> Teachers -> Material/Files

Following this hierarchy, user can view any course study material. Clicking on any file downloads the file for user.

A teacher can upload study material for only that course which is assigned to that teacher. The teacher cannot upload material for any other course which is not assigned to that teacher.

Only teacher and admin can upload study material by clicking on “upload” button where all study material files are displayed for a course. The user just needs to browse the file, select the file type i.e. whether it is notes or question paper and can also specify the file URL incase the user download from any internet source. On clicking “submit” the file is uploaded in database, can be viewed by other users as well.

Calendar Section

This section is concerned with all the academic and holiday events. Every user will be displayed a calendar car in the dashboard. The calendar is designed in such a way that only the teacher and admin can add events to it. The teacher and admin can add event to calendar by clicking on “Add Entry” button in calendar. An event can of type – Quiz, Notice, Holidays, Academic notice or information etc. On clicking a specific date, in calendar all the events that are /were scheduled on that day will be listed in events section beside the calendar.

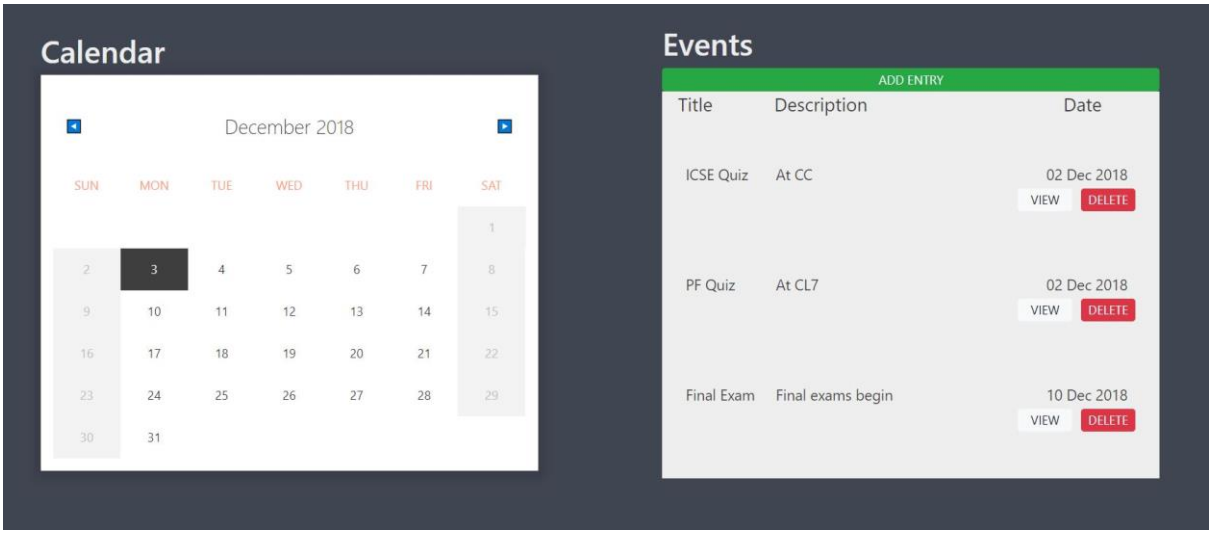


Fig: 6.3.6

6.4 Database Overview of project

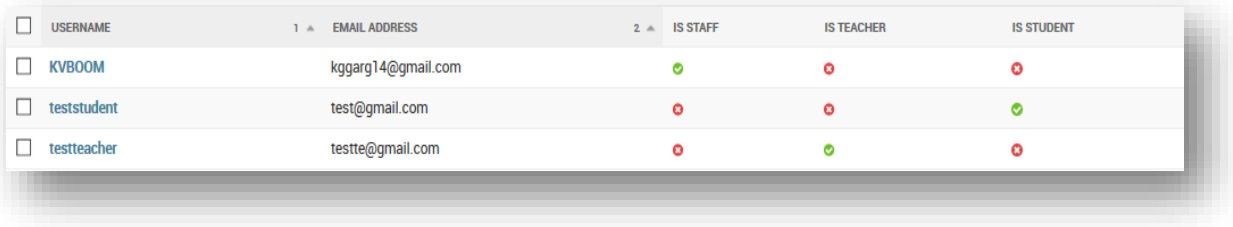
Our Project comprises of 8 schemas’:

1. MyCustomUser
2. TeacherInfo
3. StudentInfo

4. Files
5. Batch
6. Semester
7. Entry (for calendar events)
8. Subjects

MyCustomUser

Default user table was not satisfying our project requirements, so to satisfy our project requirements we defined a custom user model which inherits class 'AbstractBaseUser'. To create a new user following information is necessary – Username, Email address, user type- student or teacher or admin. After a user is created the user profile information is stored in another two tables- TeacherInfo and StudentInfo table. Following is the representation -

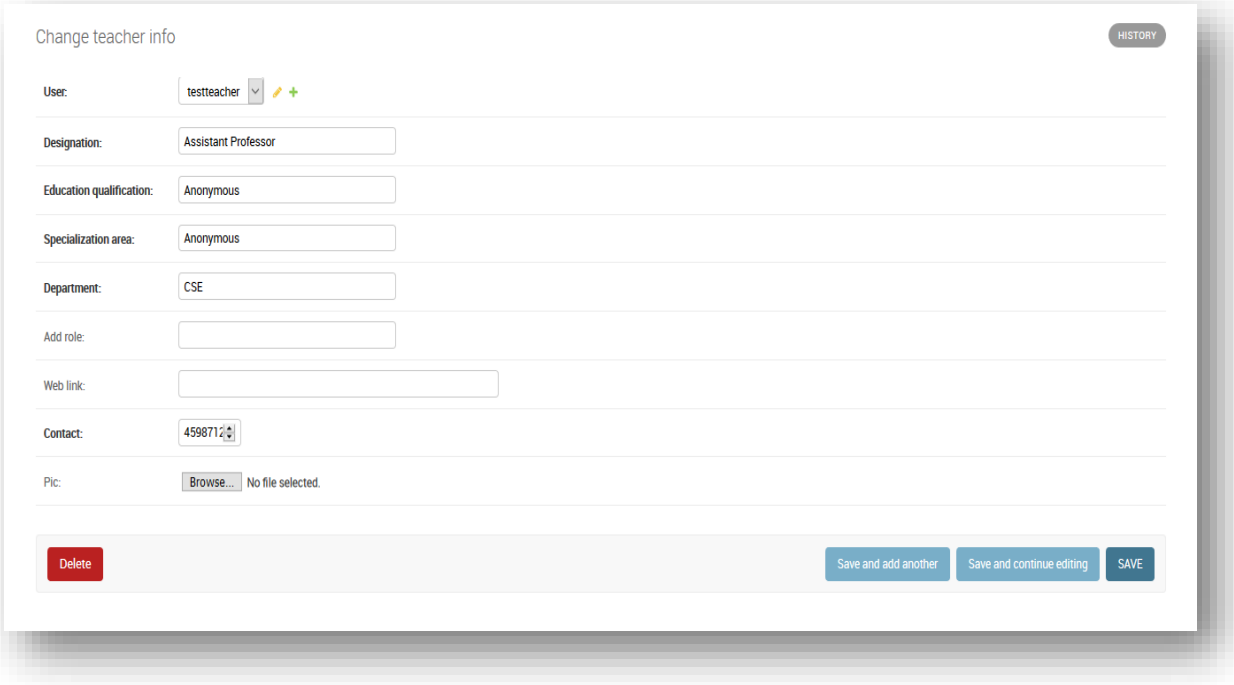


USERNAME	EMAIL ADDRESS	IS STAFF	IS TEACHER	IS STUDENT
KVBOOM	kggarg14@gmail.com	✓	✗	✗
teststudent	test@gmail.com	✗	✗	✓
testteacher	testte@gmail.com	✗	✓	✗

TeacherInfo

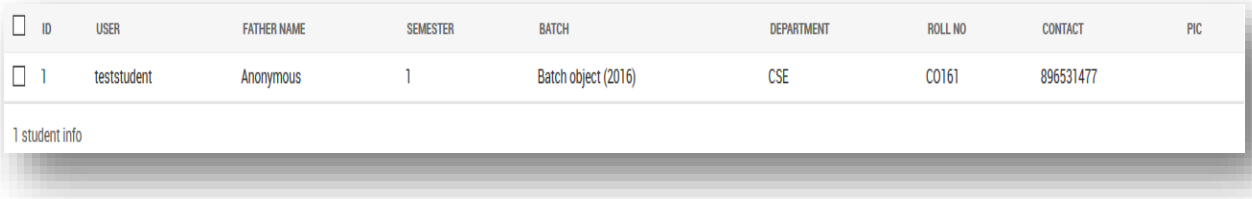
This table stores teacher personal profile information. A teacher info has 9 columns - User foreign key, Designation, Education Qualification, Specialization Area, Role, Profile Pic, Department, Web link and contact. Following is the representation how a teacher table

entry looks like:



StudentInfo

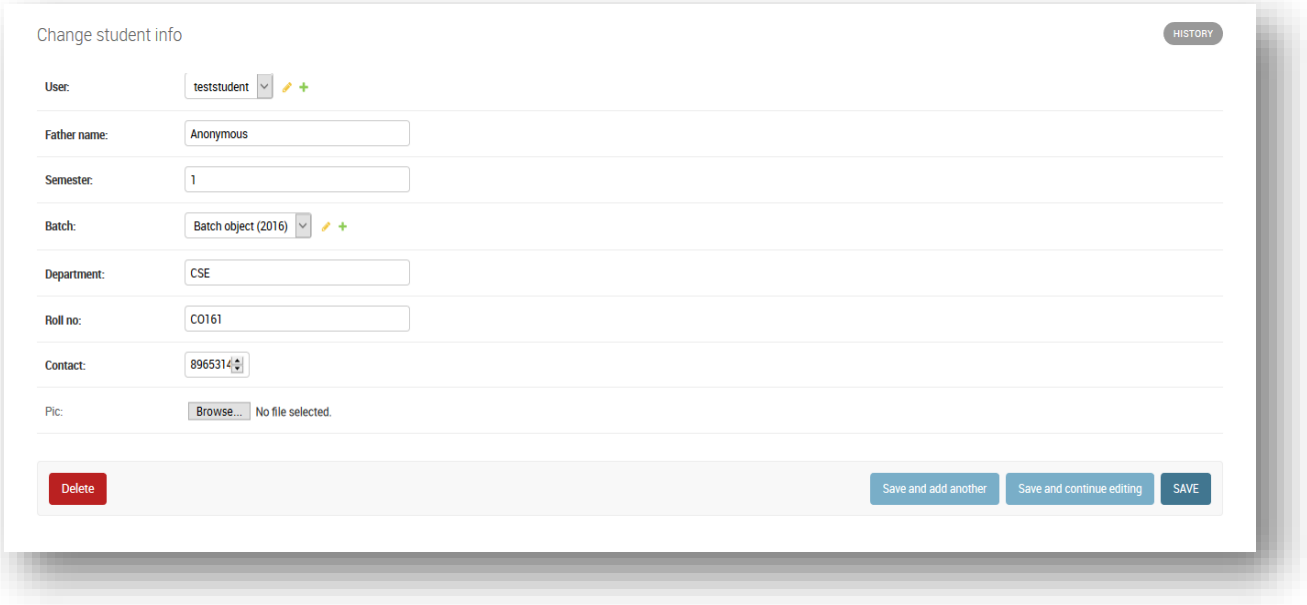
This table stores student personal profile information. A teacher info has 8 columns - User foreign key, Father name, Semester, Batch year, Roll number, Profile Pic, Department, and contact.



ID	USER	FATHER NAME	SEMESTER	BATCH	DEPARTMENT	ROLL NO	CONTACT	PIC
1	teststudent	Anonymous	1	Batch object (2016)	CSE	C0161	896531477	

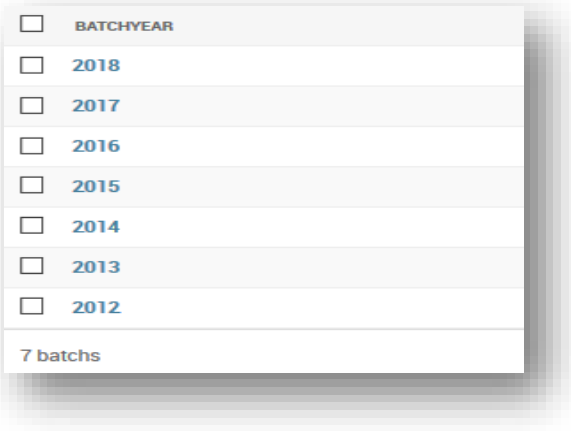
1 student info

Following is the representation how a teacher table entry looks like:



Batch year

This table stores all the batch year.



Files

This table stores all the information about a file uploaded by any user. It has 7 columns – File path, File name, File URL, File Type, Teacher Name, Subject Code and which user uploaded it.

ID	FILE	FILENAME	FILE URL	FILE TYPE	TEACHER NAME	SUBJECT CODE	UPLOADED BY
1	uploads/2018/12/02/Frame_relay.pdf	Frame relay.pdf	-	Notes	Subjects object (1)	Subjects object (1)	KVBOOM

1 files

Semester

This table stores the semsters with corresponding batch year.

SEMESTERNO	BATCHYEAR
1	Batch object (2018)
1	Batch object (2017)
1	Batch object (2016)
1	Batch object (2015)
1	Batch object (2014)
1	Batch object (2013)
1	Batch object (2012)

7 semesters

Calendar Events

This table stores all the events scheduled by teacher and admin.

ENTRY
Final Exam 2018-12-10
PF Quiz 2018-12-02
ICSE Quiz 2018-12-02

3 entrys

Subjects

This table stores information regarding subject. A subject information is defined by its – Subject code, Subject Name, Semester and Teacher name.

SUBJECTCODE	SUBJECTNAME	SEMESTERNO	TEACHERNAME
CS-102	PF	Semester object (5)	Er. Sarabjeet Singh
CS-101	ICSE	Semester object (5)	Dr. Sudhakar Kumar

2 subjectss

Conclusion

The Django framework gives us a simple and reliable way to create the course management system. It provides powerful functionalities and concise syntax to help programmers deal with the database, the web page and the inner logic. The experience of developing the group component in the system also helped us learning a lot of website development with Django. Within the Django framework, we have successfully accomplished the requirements of the system. Once this system passes the testing phase, it can be used to serve students and instructors and substitute several systems currently in service. It will make the work for teachers to manage the course much easier. It also can simplify the operations for students with notes, question papers and study material all in one system. In short, this system will bring great user experience to both teachers and students. The only limitation for this course system is that although the developers have been testing it with various use cases, it may still encounter problems during real time use. However, even if that happens, the edibility of Django would provide a simple way to fix the problem, as well as add new features into the system.

References

Full Stack Web Developer Course	https://www.udemy.com/python-and-django-full-stack-web-developer-bootcamp/
Django homepage	https://www.djangoproject.com/
Python documentation	http://www.python.org/doc
Django (web framework)	http://en.wikipedia.org/wiki/Django
Django documentation	http://docs.djangoproject.com
Python (programming language)	http://en.wikipedia.org/wiki/Python
Issues Helper	http://stackoverflow.com/