

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI-590018**



**A DBMS Mini-Project Report
On**

“My Classroom”

*Submitted in partial fulfillment of the requirements for the 5th semester of **Bachelor of Engineering in Computer Science and Engineering** of Visvesvaraya Technological University, Belagavi*

Submitted by:

P Asritha Sai 1RN19CS095

Pooja B R 1RN19CS098

Under the Guidance of:

Mrs. Soumya N G
Assistant Professor
Dept. of CSE

Mr. Sanjay P Kalas
Assistant Professor
Dept. of CSE



Department of Computer Science and Engineering
RNS Institute of Technology
Channasandra, Dr.Vishnuvardhan Road, Bengaluru-560 098
2021-2022

RNS Institute of Technology
Channasandra, Dr.Vishnuvardhan Road,
Bengaluru-560 098

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



CERTIFICATE

Certified that the DBMS mini-project work entitled “My Classroom” has been successfully carried out by **P Asritha Sai** bearing USN **1RN19CS095** and **Pooja B R** bearing USN **1RN19CS098**, bonafide students of **RNS Institute of Technology** in partial fulfillment of the requirements for the **5th semester Bachelor of Engineering in Computer Science and Engineering** of **Visvesvaraya Technological University, Belagavi**, during the academic year 2021-2022. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report. The project report has been approved as it satisfies the mini-project requirements of the DBMS lab of 5th semester BE in CSE.

Mrs. Soumya N G
Assistant Professor
Dept. of CSE

Mr. Sanjay P Kalas
Assistant Professor
Dept. of CSE

Dr. Kiran P
Prof. and Head
Dept of CSE

External Viva:

Name of the Examiners

Signature with Date

- 1.
- 2.

ACKNOWLEDGMENT

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped us in carrying out this project work. We would like to take this opportunity to thank them all.

We are grateful to **Dr. M K Venkatesha**, Principal, RNSIT, Bangalore, for his support towards completing this mini project.

We would like to thank **Dr. Kiran P** Prof. & Head, Department of Computer Science & Engineering, RNSIT, Bangalore, for his valuable suggestions and expert advice.

We deeply express our sincere gratitude to our guide **Mrs. Soumya N G**, Assistant Professor, **Mr. Sanjay P Kalas**, Assistant Professor, Department of CSE, RNSIT, Bangalore, for their able guidance, regular source of encouragement and assistance throughout this project.

We would like to thank all the teaching and non-teaching staff of department of Computer Science & Engineering, RNSIT, Bengaluru for their constant support and encouragement.

Date:

P Asritha Sai

1RN19CS095

Place: Bengaluru

Pooja B R

1RN15CS098

ABSTRACT

My classroom deals with the maintenance of the student's information, attendance, internal marks. It displays the student's time-table, tasks and the upcoming events.

There are two types of users : the admin, the students. The admin can add /modify/ delete students' information, attendance, internal marks, time-table, tasks and events.

The students can login using the user name and the password. The students can view their information, attendance, internal marks, time-table, tasks and events.

CONTENTS

Chapter No.	Title	Page No
I.	Acknowledgment	
II.	Abstract	
1.	Introduction	
1.1	Database technologies	1
1.2	Characteristics of database approach	2
1.3	Applications of DBMS	3
1.4	Problem description/ statement	4
2.	Requirements Analysis	
2.1	Hardware Requirements	5
2.2	Software Requirements	5
2.3	Functional Requirements	
2.3.1	Major Entities	6
2.3.2	End User Requirements	6
2.3.2.1	Bootstrap	6
2.3.2.2	Django	7
2.3.2.3	Python	9
2.3.2.4	SQLite	10
3.	Database Design	
3.1	Entities, Attributes and Relationships	13
3.2	ER Schema	15
3.3	Relational Schema	16
4.	Implementation	
4.1	Creating Project	17
4.2	How a Django code looks like	19
4.3	Pseudo code for major functionalities	21
5.	Results , snapshots and discussions	25
6.	Conclusion and Future Enhancements	29
	Bibliography	30

CHAPTER 1

INTRODUCTION

1.1 DATABASE TECHNOLOGIES

The essential feature of database technology is that it provides an internal representation (model) of the external world of interest. Examples are the representation of a particular date/time/flight/aircraft in airline reservation or of item code/item description/quantity on hand/reorder level/reorder quantity in a stock control system.

The technology involved is concerned primarily with maintaining the internal representation consistent with external reality; this involves the results of extensive R&D over the past 30 years in areas such as user requirements analysis, data modeling, process modeling, data integrity, concurrency, transactions, file organization, indexing, rollback and recovery, persistent programming, object-orientation, logic programming, deductive database systems, active database systems... and in all these (and other) areas there remains much to be done.

The essential point is that database technology is a CORE TECHNOLOGY with links to:

- Information management / processing
- Data analysis / statistics
- Data visualization / presentation
- Multimedia and hypermedia
- Office and document systems
- Business processes, workflow, CSCW (computer-supported cooperative work)

Relational DBMS is the modern base technology for many business applications. It offers flexibility and easy-to-use tools at the expense of ultimate performance. More recently relational systems have started to extend their facilities in the directions of information retrieval, object-orientation and deductive/active systems leading to the so-called 'Extended Relational Systems'.

Information Retrieval Systems began with handling library catalogs and then extended to full free-text utilizing inverted index technology with a lexicon or thesaurus. Modern systems utilize some KBS (knowledge-based systems) techniques to improve retrieval.

Object-Oriented DBMS started for engineering applications where objects are complex, have versions and need to be treated as a complete entity. OODBMSs share many of the OOPL features such as identity, inheritance, late binding, overloading and overriding. OODBMSs have found favors in engineering and office systems but have not yet been successful in traditional application areas.

Deductive / Active DBMS have emerged over the last 20 years and combine logic programming technology with database technology. This allows the database itself to react to external events and to dynamically maintain its integrity with respect to the real world.

1.2 CHARACTERISTICS OF DATABASE APPROACH

- an inconsistent state. A DBMS can provide greater consistency when compared to earTraditionally, data was organized in file formats. DBMS was a new concept then, and research was done to make it overcome the deficiencies of traditional style of data management. A modern DBMS has the following characteristics –
- Real-world entity – A modern DBMS is more realistic and uses real-world entities to design its architecture. It uses the behavior and attributes too. For example, a school database may use students as an entity and their age as an attribute.
- Relation-based tables – DBMS allows entities and relations among them to form tables. A user can understand the architecture of a database just by looking at the table names.
- Isolation of data and application – A database system is entirely different from its data. A database is an active entity, whereas data is said to be passive, on which the database works and organizes. DBMS also stores metadata, which is data about the data, to ease its own process.
- Less redundancy – DBMS follows the rules of normalization, which splits a relation when any of its attributes is having redundancy in values. Normalization is a mathematically rich and scientific process that reduces data redundancy.

- Consistency – Consistency is a state where every relation in a database remains consistent. There exist methods and techniques, which can detect attempts to leave the database in other forms of data storing applications like file-processing systems.
- Query Language – DBMS is equipped with query language, which makes it very efficient in retrieving and manipulating data. A user can apply as many and as different filtering options as required to retrieve a set of data. This was not possible when the file-processing system was used.
- ACID Properties – DBMS follows the concepts of Atomicity, Consistency, Isolation, and Durability (normally shortened as ACID). These concepts are applied on transactions, which manipulate data in a database. ACID properties help the database stay healthy in multi-transactional environments and in case of failure.
- Multiuser and Concurrent Access – DBMS supports multi-user environments and allows them to access and manipulate data in parallel. Though there are restrictions on transactions when a user attempts to handle the same data item, the users are always unaware of them.
- Multiple views – DBMS offers multiple views for different users. A user who is in the Sales department will have a different view of the database than a person working in the Production department. This feature enables the users to have a concentrated view of the database according to their requirements.
- Security – Features like multiple views offer security to some extent where users are unable to access data of other users and departments. DBMS provides methods to impose constraints while entering data into the database and retrieving the same at a later stage. DBMS offers many different levels of security features, which enables multiple users to have different views with different features. For example, a user in the Sales department cannot see the data that belongs to the Purchase department. Additionally, it can also be managed how much data of the Sales department should be displayed to the user. Since a DBMS is not saved on the disk as traditional file systems, it is very hard for miscreants to break the code.

1.3 APPLICATIONS OF DBMS

Applications of Database Management Systems are:

- **Telecom:** There is a database to keep track of the information regarding calls made, network usage, customer details etc. Without the database systems it is hard to maintain the huge amount of data that keeps getting updated every millisecond.
- **Industry:** Whether it is a manufacturing unit, or a warehouse or distribution center, each one needs a database to keep the records of ins and outs. For example, a distribution center should keep a track of the product units that are supplied into the center as well as the products that got delivered out from the distribution center on each day; this is where DBMS comes into picture.
- **Banking System:** For storing customer info, tracking his/her day to day credit and debit transactions, generating bank statements etc. All this work has been done with the help of Database management systems.
- **Education sector:** Database systems are frequently used in schools and colleges to store and retrieve the data regarding student, staff details, course details, exam details, payroll data, attendance details, fees details etc. There is lots of interrelated data that needs to be stored and retrieved in an efficient manner.
- **Online shopping:** You must be aware of the online shopping websites such as Amazon, Flip kart etc. These sites store the product information, your addresses and preferences, credit details and provide you with the relevant list of products based on your query. All this involves a Database management system.

1.4 PROBLEM DESCRIPTION/STATEMENT

My classroom is one stop for student's information, attendance, and internal marks. It displays the student's time-table, tasks and upcoming events.

The final attendance and final internal marks will be calculated automatically and displayed. It keeps the students updated about the tasks with the deadlines and the upcoming events. It reduces the need for manual labor which is prone to human errors. It reduces the time consumed and increases the speed of performing the tasks respectively.

CHAPTER 2

REQUIREMENT ANALYSIS

2.1 HARDWARE REQUIREMENTS

The Hardware requirements are very minimal and the program can be run on most of the machines.

Processor	:	Pentium4 processor
Processor Speed	:	2.4 GHz
RAM	:	1 GB
Storage Space	:	40 GB
Monitor Resolution	:	1024*768 or 1336*768 or 1280*1024

2.2 SOFTWARE REQUIREMENTS

Operating System : Windows XP

IDE : Visual Studio

Django : We have chosen to use Django for the back-end of the website

Database : SQLitedatabase

2.3 FUNCTIONAL REQUIREMENTS

2.3.1 Major Entities

- **Student:** Each student belongs to a class identified by semester and section. Each class belongs to a department and is assigned a set of courses. Therefore, these courses are common to all students of that class. The students are given a unique username and password to login.
- **Administrator:** The administrator will have access to all the information in the different tables in the database. They will access all the tables in a list form. They will be able to add an entry in any table and also edit them. The design of the view for the admin will provide a modular interface so that querying the tables will be optimized. They will be provided with search and filter features so that they can access data efficiently.

2.3.2 End User Requirements

The technical requirements for the project are mentioned below.

2.3.2.1 Bootstrap

Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS and (optionally) JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components.

Bootstrap is an HTML, CSS & JS Library that focuses on simplifying the development of informative web pages (as opposed to web apps). The primary purpose of adding it to a web project is to apply Bootstrap's choices of color, size, font and layout to that project. As such, the primary factor is whether the developers in charge find those choices to their liking. Once added to a project, Bootstrap provides basic style definitions for all HTML elements. The result is a uniform appearance for prose, tables and form elements across web browsers. In addition, developers can take advantage of CSS classes defined in Bootstrap to further customize the appearance of their contents. For example, Bootstrap has provisioned for light- and dark-colored tables, page headings, more prominent pull quotes, and text with a highlight.

Bootstrap also comes with several JavaScript components in the form of jQuery plugins. They provide additional user interface elements such as dialog boxes, tooltips, and carousels. Each Bootstrap component consists of an HTML structure, CSS declarations, and in some cases accompanying JavaScript code. They also extend the functionality of some existing interface elements, including for example an auto-complete function for input fields.

The most prominent components of Bootstrap are its layout components, as they affect an entire web page. The basic layout component is called "Container", as every other element in the page is placed in it. Developers can choose between a fixed-width container and a fluid-width container. While the latter always fills the width of the web page, the former uses one of the predefined fixed widths, depending on the size of the screen showing the page

Once a container is in place, other Bootstrap layout components implement a CSS Flexbox layout through defining rows and columns.

A precompiled version of Bootstrap is available in the form of one CSS file and three JavaScript files that can be readily added to any project. The raw form of Bootstrap, however, enables developers to implement further customization and size optimizations. This raw form is modular, meaning that the developer can remove unneeded components, apply a theme and modify the uncompiled Sass files.

2.3.2.2 Django

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Built by experienced developers, Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It is free and open source, has a thriving and active community, great documentation, and many options for free and paid-for support.

Django helps you write software that is:

Complete

Django follows the "Batteries included" philosophy and provides almost everything developers might want to do "out of the box". Because everything you need is part of the one "product", it all works seamlessly together, follows consistent design principles, and has extensive and [up-to-date documentation](#).

Versatile

Django can be (and has been) used to build almost any type of website — from content management systems and wikis, through to social networks and news sites. It can work with any client-side framework, and can deliver content in almost any format (including HTML, RSS feeds, JSON, XML, etc). Internally, while it provides choices for almost any functionality you might want (e.g. several

popular databases, templating engines, etc.), it can also be extended to use other components if needed.

Secure

Django helps developers avoid many common security mistakes by providing a framework that has been engineered to "do the right things" to protect the website automatically. For example, Django provides a secure way to manage user accounts and passwords, avoiding common mistakes like putting session information in cookies where it is vulnerable (instead cookies just contain a key, and the actual data is stored in the database) or directly storing passwords rather than a password hash.

A password hash is a fixed-length value created by sending the password through a cryptographic hash function. Django can check if an entered password is correct by running it through the hash function and comparing the output to the stored hash value. However due to the "one-way" nature of the function, even if a stored hash value is compromised it is hard for an attacker to work out the original password.

Scalable

Django uses a component-based "shared-nothing" architecture (each part of the architecture is independent of the others, and can hence be replaced or changed if needed). Having a clear separation between the different parts means that it can scale for increased traffic by adding hardware at any level: caching servers, database servers, or application servers. Some of the busiest sites have successfully scaled Django to meet their demands (e.g. Instagram and Disqus).

Maintainable

Django code is written using design principles and patterns that encourage the creation of maintainable and reusable code. In particular, it makes use of the Don't Repeat Yourself (DRY) principle so there is no unnecessary duplication, reducing the amount of code. Django also promotes the grouping of related functionality into reusable "applications" and, at a lower level, groups related code into modules (along the lines of the Model View Controller(MVC) pattern).

Portable

Django is written in Python, which runs on many platforms. That means that you are not tied to any particular server platform, and can run your applications on many flavors of Linux, Windows, and Mac OS X. Furthermore,

2.3.2.3Python

Python is a high-level programming language, meaning it contains words and phrases comprehensible to humans. To translate this high-level language into machine code, Python uses an interpreter. An interpreter converts source code into code understood by computers. As an interpreted language, Python uses memory efficiently, is easy to debug, and allows developers to perform complex tasks in a couple of steps and edit code quickly.

Most modern programming languages, including Python, are considered object-oriented. The idea behind object-oriented languages is that the program is split into independent objects that communicate between themselves. This pretty much looks like a collection of mini-programs. Objects help developers write better structured code, resulting in software that's easier and faster to build, maintain, and debug.

Python is a dynamic language. This means that developers don't have to declare variable types. Python verifies types and errors at runtime (when the program is launched), but not during compilation (when source code is converted to machine code). This results in less code, faster development, and greater flexibility and resilience.

Python is a multi-purpose language and can be used to build practically anything. Companies around the world use Python for artificial intelligence and machine learning, website development, scientific and numerical computing, gaming, and many other uses

2.3.2.4 SQLite

SQLite is a database engine, written in the C language. It is not a standalone app; rather, it is a library that software developers embed in their apps. As such, it belongs to the family of embedded databases. It is the most widely deployed database engine, as it is used by several of the top web browsers, operating systems, mobile phones, and other embedded systems.

SQLite has bindings to many programming languages. It generally follows PostgreSQL syntax but does not enforce type checking. This means that one can, for example, insert a string into a column defined as an integer.

Unlike client-server database management systems, the SQLite engine has no standalone processes with which the application program communicates. Instead, the SQLite library is linked in and thus becomes an integral part of the application program. Linking may be static or dynamic. The application program uses SQLite's functionality through simple function calls, which reduce latency in database access: function calls within a single process are more efficient than inter-process communication.

SQLite stores the entire database (definitions, tables, indices, and the data itself) as a single cross-platform file on a host machine. It implements this simple design by locking the entire database file during writing. SQLite read operations can be multitasked, though writes can only be performed sequentially.

Due to the server-less design, SQLite applications require less configuration than client-server databases. SQLite is called *zero-conf* because it does not require service management (such as startup scripts) or access control based on GRANT and passwords. Access control is handled by means of file-system permissions given to the database file itself. Databases in client-server systems use file-system permissions that give access to the database files only to the daemon process.

Another implication of the serverless design is that several processes may not be able to write to the database file. In server-based databases, several writers will all connect to the same daemon, which is able to handle its locks internally. SQLite, on the other hand, has to rely on file-system locks. It has less knowledge of the other processes that are accessing the database at the same time. For simple queries with little concurrency, SQLite performance profits from avoiding the overhead of passing its data to another process.

SQLite uses PostgreSQL as a reference platform. "What would PostgreSQL do" is used to make sense of the SQL standard. One major deviation is that, with the exception of primary keys, SQLite does not enforce type checking; the type of a value is dynamic and not strictly constrained by the schema (although the schema will trigger a conversion when storing, if such a conversion is potentially reversible). SQLite strives to follow Postel's rule.

SQLite implements most of the SQL-92 standard for SQL, but lacks some features. For example, it only partially provides triggers and cannot write to views (however, it provides INSTEAD OF triggers that provide this functionality). Its support of ALTER TABLE statements is limited.

SQLite uses an unusual type system for a SQL-compatible DBMS: instead of assigning a type to a column as in most SQL database systems, types are assigned to individual values; in language terms it is *dynamically typed*. Moreover, it is *weakly typed* in some of

the same ways that Perl is: one can insert a string into an integer column (although SQLite will try to convert the string to an integer first, if the column's preferred type is integer). This adds flexibility to columns, especially when bound to a dynamically typed scripting language.

Tables normally include a hidden *rowid* index column, which gives faster access. If a database includes an Integer Primary Key column, SQLite will typically optimize it by treating it as an alias for *rowid*, causing the contents to be stored as a strictly typed 64-bit signed integer and changing its behavior to be somewhat like an auto-incrementing column. Future versions of SQLite may include a command to introspect whether a column has behavior like that of *rowid* to differentiate these columns from weakly typed, non-auto incrementing Integer Primary Keys.

Several computer processes or threads may access the same database concurrently. Several read accesses can be satisfied in parallel. A write access can only be satisfied if no other accesses are currently being serviced. Otherwise, the write access fails with an error code (or can automatically be retried until a configurable timeout expires). This concurrent access situation would change when dealing with temporary tables.

CHAPTER 3

DATABASE DESIGN

3.1 Entities, Attributes and Relationships

Branch:

Branch_id

Branch_Name

Sem:

Sem

Classroom:

Class_id

Branch_id

Sec

Sem

Student:

User_name

Password

Usn

Class_id

Courses:

Course_id

Course_name

Branch_name

Sem

is_elective

Professor:

Professor_id

Name

Branch_name

Designation

Class_courses:

Class_id

Course_id

Professor_id

Attendance:

Usn

Course

Status

Internals:

IA_marks

2IA_marks

3IA_marks

FinalIA_marks

Time table:

Class_id

Course_id

Day

Timeslot

To do:

Class_id

Task

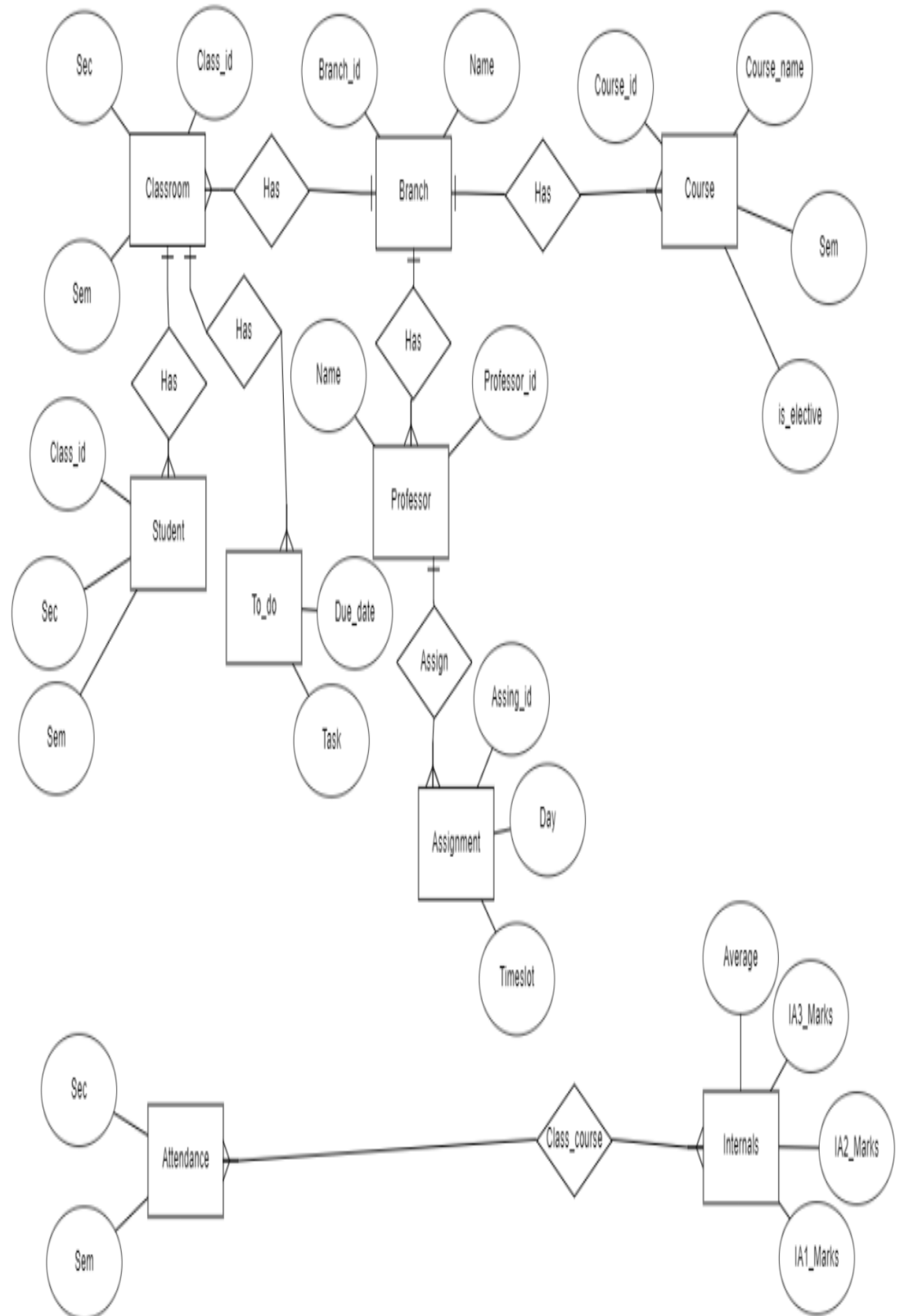
Due_date

Events:

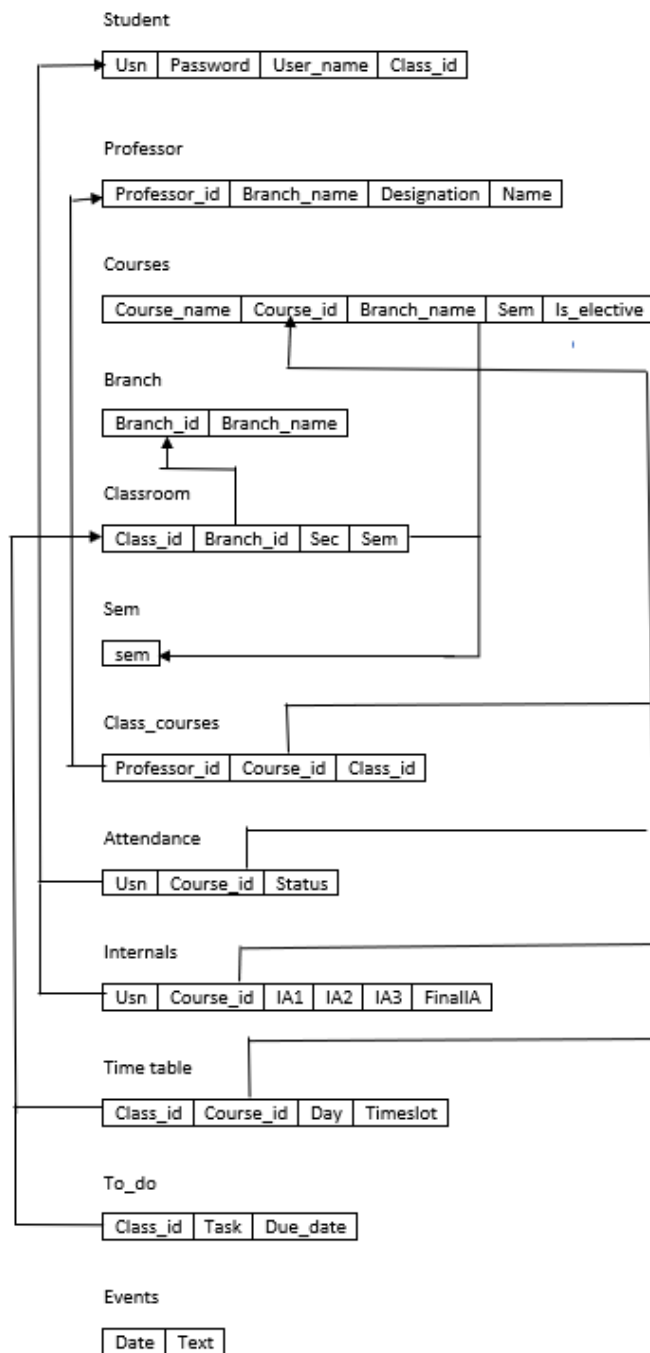
Date

Text

3.2 ER DIAGRAM



3.3 RELATIONAL SCHEMA



CHAPTER 4

IMPLEMENTATION

4.1 Creating Project

- In Django, every web app you want to create is called a project; and a project is a sum of applications.
- The first step is to create a new project, that can be done by running the below command in the command prompt

\$ django-admin startproject mysite

This will create a mysite directory in your current directory.

- The below files will be created by startproject

```
mysite/  
    manage.py  
mysite/  
    __init__.py  
    settings.py  
    urls.py  
    asgi.py  
    wsgi.py
```

These files are:

1. The outer **mysite/** root directory is a container for your project. Its name doesn't matter to Django; you can rename it to anything you like.
2. **manage.py**: A command-line utility that lets you interact with this Django project in various ways.
3. The inner **mysite/** directory is the actual Python package for your project. Its name is the Python package name you'll need to use to import anything inside it (e.g. **mysite.urls**).
4. **mysite/ __init__.py**: An empty file that tells Python that this directory should be considered a Python package.

5. **mysite/urls.py**: The URL declarations for this Django project; a “table of contents” of your Django-powered site.
 6. **mysite/asgi.py**: An entry-point for ASGI-compatible web servers to serve your project.
 7. **mysite/wsgi.py**: An entry-point for WSGI-compatible web servers to serve your project.
- To verify if the project works ,execute the below command in the command prompt

\$ python manage.py runserver

You'll see the following output on the command line:

```
Performing system checks...
System check identified no issues (0 silenced).

You have unapplied migrations; your app may not work properly until they are applied.
Run 'python manage.py migrate' to apply them.

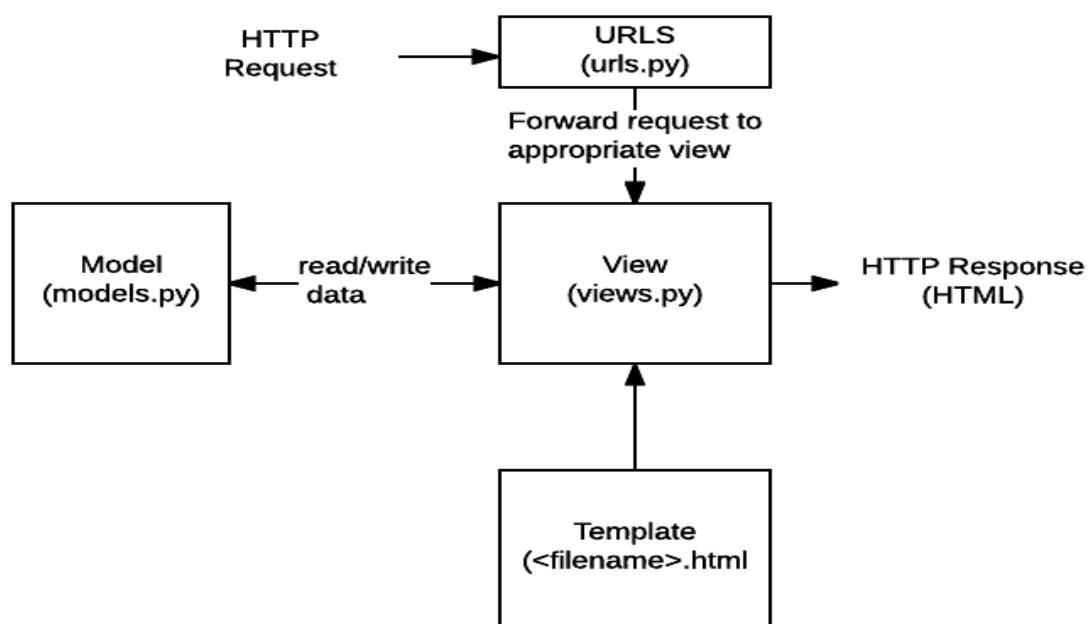
February 09, 2022 - 15:50:53
Django version 4.0, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/

Quit the server with CONTROL-C.
```

4.2 How a Django code looks like:

In a traditional data-driven website, a web application waits for HTTP requests from the web browser (or other client). When a request is received the application works out what is needed based on the URL and possibly information in `POST` data or `GET` data. Depending on what is required it may then read or write information from a database or perform other tasks required to satisfy the request. The application will then return a response to the web browser, often dynamically creating an HTML page for the browser to display by inserting the retrieved data into placeholders in an HTML template.

Django web applications typically group the code that handles each of these steps into separate files:



- **URLs:** While it is possible to process requests from every single URL via a single function, it is much more maintainable to write a separate view function to handle each resource. A URL mapper is used to redirect HTTP requests to the appropriate view based on the request URL. The URL mapper can also match particular patterns of strings or digits that appear in a URL and pass these to a view function as data.

- **View:** A view is a request handler function, which receives HTTP requests and returns HTTP responses. Views access the data needed to satisfy requests via *models*, and delegate the formatting of the response to *templates*.
- **Models:** Models are Python objects that define the structure of an application's data, and provide mechanisms to manage (add, modify, delete) and query records in the database.
- **Templates:** A template is a text file defining the structure or layout of a file (such as an HTML page), with placeholders used to represent actual content. A *view* can dynamically create an HTML page using an HTML template, populating it with data from a *model*. A template can be used to define the structure of any type of file; it doesn't have to be HTML.

4.3 Pseudo Code For Major Functionalities

- **Login and Logout page:** It is used for login and logout purposes. When a student enters the correct usn and password it will take you to the next page.

```
def login(request):
    if request.method=='POST':
        form = StudentSignInForm(request.POST)
        if form.is_valid():
            usn_ma = form.cleaned_data['usn']
            password_ma = form.cleaned_data['password']
            user = auth.authenticate(
                usn=usn_ma,password=password_ma)
            if(user):
                print("Authenticated")
                auth.login(request,user,backend='my_classroom_app.authentication.backend.StudentBackend')
                print('done')
                return redirect('/index')
            else:
                return redirect('/accounts/login')

        else:
            form = StudentSignInForm(request.POST)
            return render(request,'registration/login.html',{'form':form})

def logout(request):
    auth.logout(request)
    return redirect('/accounts/login')
    print("LOGOUT")

@login_required()
def index(request):
    current_user = request.user
    event=events.objects.raw("SELECT * FROM my_classroom_app_events")
    return render(request,"index.html",{'events':event,'user':current_user})

@login_required()
```

Fig 4.1 Login and Logout Code Snippet

- **HomePage:** After successful login, the student is presented a homepage with sections like their profiles, attendance, marks and timetable,todo, events.

```
{% extends "base.html" %}

{% block content %}
<h2>Homepage</h2>
```

```
<p>Welcome to My Site, {% if user.is_authenticated %} {{ user.username }} {% else %} visitor {% endif %}!</p>
<p>
  <strong>Total users registered: </strong> {{ count }}
</p>
{% endblock %}
```

Fig 4.2Homepage (html)Code snippet

- **Profile:** Profile section will have student information like name,usn,class,date of birth,gender,email id,phone number.

```
def profile(request):
    stud = request.user

    return render(request,"profile.html",{ 'student':stud})
```

Fig 4.3 Profile code snippet

```
<div class="panel-heading">
  <h3 class="panel-title">{{ student.name }}</h3>
</div>
<div class="panel-body">
  <div class="row">
    <div class="col-md-3 col-lg-3 " align="center"> </div>
    <div class=" col-md-9 col-lg-9 ">
      <table class="table table-user-information">
        <tbody>
          <tr>
            <td>Name</td>
            <td>{{ student.user_name }}</td>
          </tr>
          <tr>
            <td>USN</td>
            <td>{{ student.usn }}</td>
          </tr>
          <tr>
            <td>Class</td>
            <td>{{ student.class_id }}</td>
          </tr>
          <tr>
            <td>Date of Birth</td>
            <td>{{ student.DOB }}</td>
          </tr>
          <tr>
            <td>Gender</td>
            <td>{{ student.gender }}</td>
          </tr>
        </tbody>
      </table>
    </div>
  </div>
```

```

</tr>
<tr>
<td>Email</td>
<td>{{student.email}}</td>
</tr>
<td>Phone Number</td>
<td>{{student.phoneno}}
</td>

```

Fig 4.4 Profile(html) code snippet

- **Internals and Attendance:** On the attendance page , a student can view their attendance in each course.

On the internals page , a student can view the internal marks obtained for each subject and the average marks obtained.

```

def intattd(request):
    internal_scores = internals.objects.raw("SELECT * from my_classroom_app_internals where usn_id = %s",[request.user.usn])
    print(str(internal_scores))
    att = attendance.objects.raw("SELECT * from my_classroom_app_attendance where usn_id = %s",[request.user.usn])
    for att_per in att:
        att_per.avg = (att_per.a1+att_per.a2+att_per.a3)/3
    for score in internal_scores:
        score.final_ia = (score.ia1+score.ia2+score.ia3)*30/150
    return render(request,"internals&attendance.html",{ 'att':att,'scores':internal_scores})

```

Fig 4.5 Internals and Attendance code snippet

```

<div class="container-fluid p-5">
<h2>Attendance</h2>
<table class="table table-hover">
<thead>
<tr>
<th scope="col">Course</th>
<th scope="col">Attendance_1</th>
<th scope="col">Attendance_2</th>
<th scope="col">Attendance_3</th>
<th scope="col">Final_Attendance</th>
</tr>
</thead>
<tbody>
{%for att_per in att%}
<tr>
<th scope="row">{{att_per.course_id}}</th>
<td>{{att_per.a1}}</td>
<td>{{att_per.a2}}</td>
<td>{{att_per.a3}}</td>

```

```
<td>{{att_per.avg}}</td>
</tr>
{%endfor%}
```

Fig 4.6 Attendance (html) code snippet

```
<h2>Internal-marks</h2>
<table class="table table-hover">
<thead>
<tr>
<th scope="col">Course</th>
<th scope="col">First_IA_marks</th>
<th scope="col">Second_IA_marks</th>
<th scope="col">Third_IA_marks</th>
<th scope="col">Final_IA_marks</th>
</tr>
</thead>
<tbody>
{%for score in scores%}
<tr>
<th scope="row">{{score.course_id}}</th>
<td>{{score.ia1}}</td>
<td>{{score.ia2}}</td>
<td>{{score.ia3}}</td>
<td>{{score.final_ia}}</td>
</tr>
{%endfor%}
```

Fig 4.7 Internals(html) code snippet

CHAPTER 5

RESULTS, SNAPSHOTS AND DISCUSSIONS

5.1 For student

- **Login page:** Students will login by entering their usn and password.

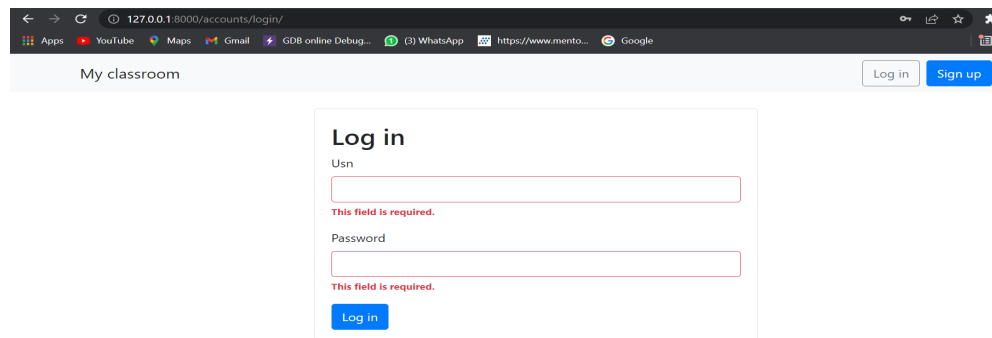
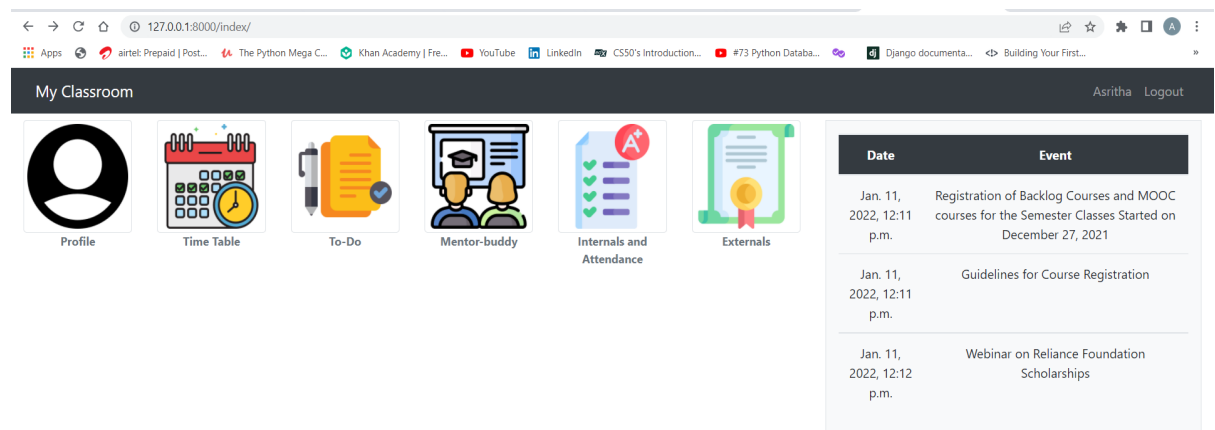


Fig 5.1 Loginpage

- **Homepage:** Once a student successfully logs in , he/she will be directed to the home page. A student can logout by clicking on the logout button in the top right corner.



Date	Event
Jan. 11, 2022, 12:11 p.m.	Registration of Backlog Courses and MOOC courses for the Semester Classes Started on December 27, 2021
Jan. 11, 2022, 12:11 p.m.	Guidelines for Course Registration
Jan. 11, 2022, 12:12 p.m.	Webinar on Reliance Foundation Scholarships

Fig 5.3 Homepage

- **Sign up Page:** Students who do not have an account can sign up and create a new account by filling out the details present in the sign up page.
- **Profile:** Profile section has student information like name, usn, dob, gender, mail, phone number.

Name	Asritha
USN	95
Class	CSE5B
Date of Birth	Nov. 12, 2000
Gender	female
Email	1rn19cs095@gmail.com
Phone Number	9620828373

Fig 5.4 Profile

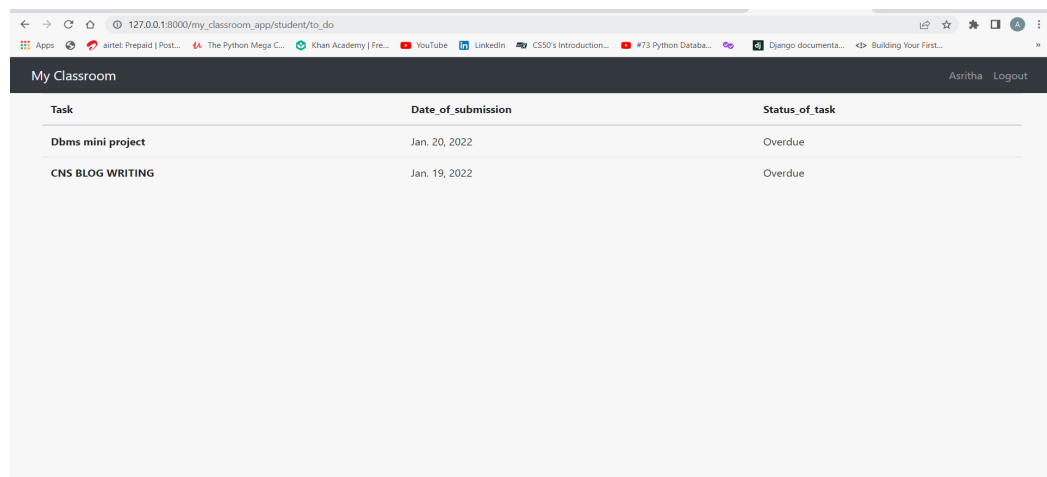
- **Time-Table:** Time-table section has the time-table of the student of a particular class

Subject	Day	Time
DBMS	Monday	8:30 - 9:30
MEIT	Tuesday	9:30 - 10:30

Subject	Professor
DBMS	Soumya N
MEIT	Sanjay K

Fig 5.5 Time-Table

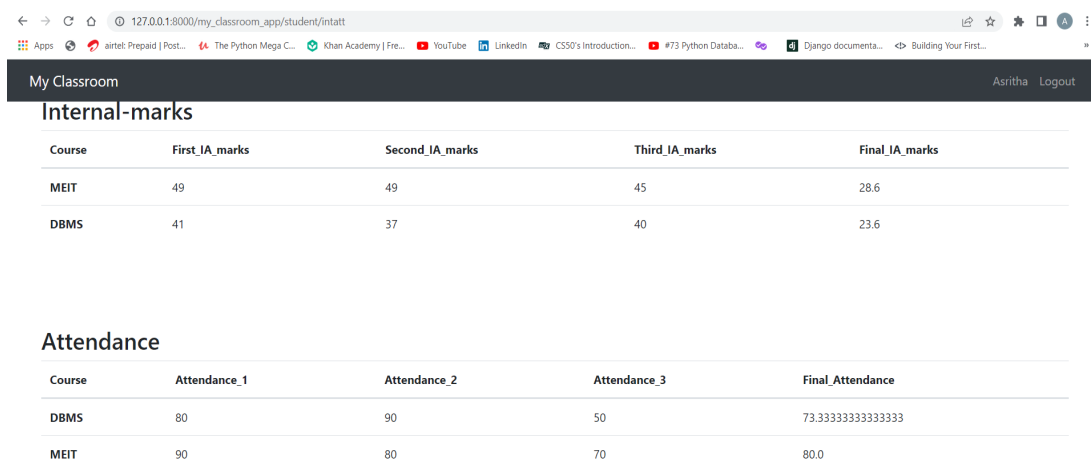
- **Tasks:** Tasks section displays tasks for the student with the due date.



Task	Date_of_submission	Status_of_task
Dbms mini project	Jan. 20, 2022	Overdue
CNS BLOG WRITING	Jan. 19, 2022	Overdue

Fig 5.6 Tasks

- **Internal marks and attendance:** Displays the internal marks and the final internal marks of the student. Displays the total attendance after every internal and the total attendance of the student



Course	First_IA_marks	Second_IA_marks	Third_IA_marks	Final_IA_marks
MEIT	49	49	45	28.6
DBMS	41	37	40	23.6

Course	Attendance_1	Attendance_2	Attendance_3	Final_Attendance
DBMS	80	90	50	73.33333333333333
MEIT	90	80	70	80.0

Fig 5.7 Internal marks and attendance

5.2 For Administrators:

- **Django administration login:** Admin can log into the admin page of django by providing the correct username and password.

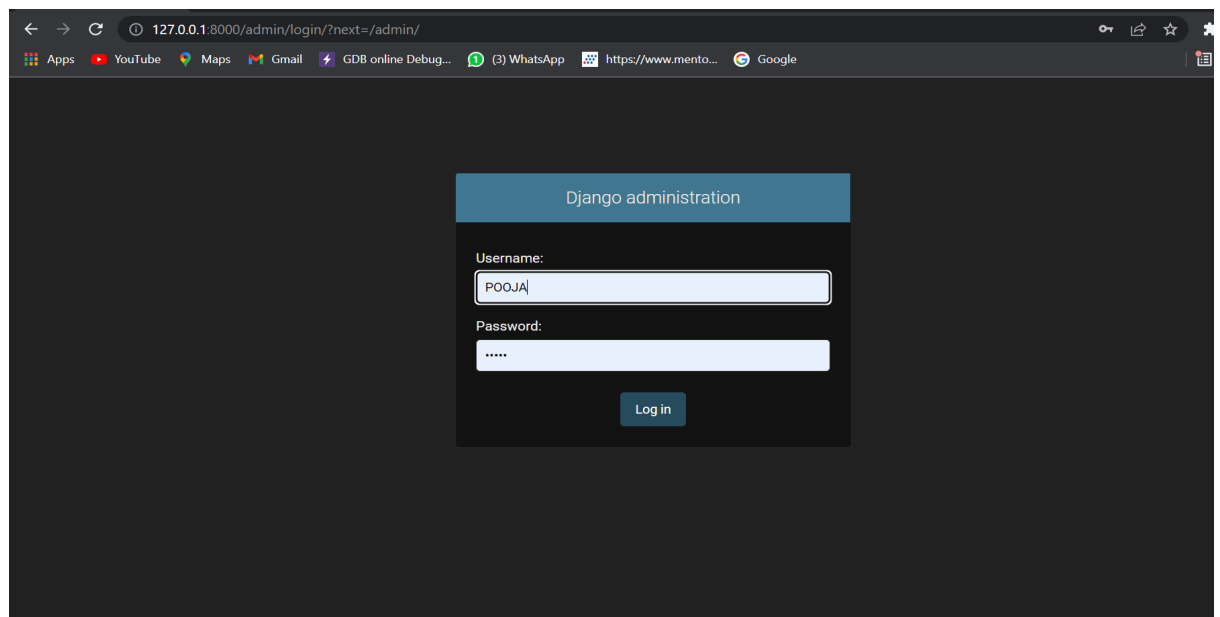


Fig 5.8 Login page

- **Site administration page:** Here an admin can add, delete or update the entries to the database.

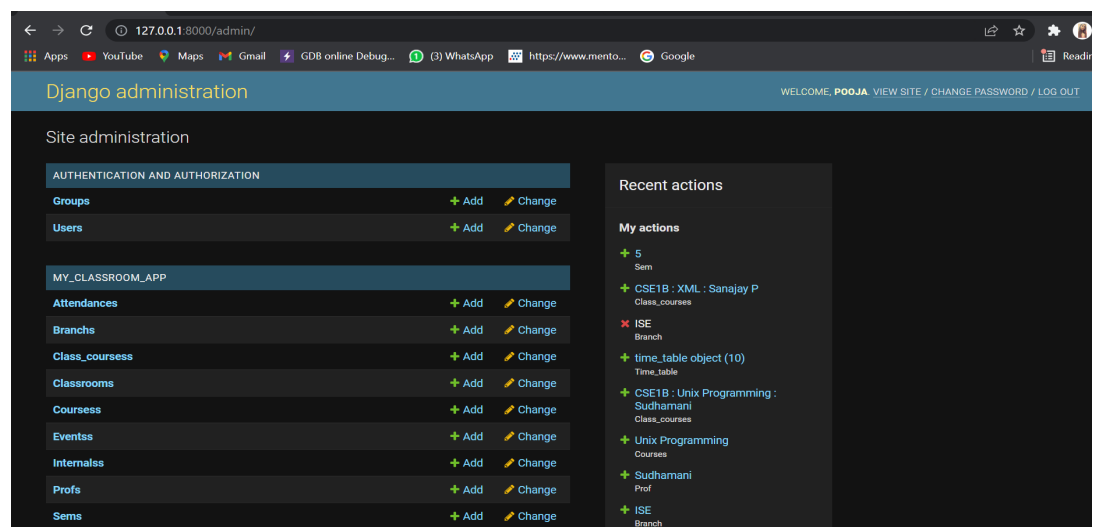


Fig 5.9 Site administration page

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENTS

6.1 Conclusion

The My Classroom web based application is developed using HTML,CSS,Bootstrap, Python,Django and SQLite.

There is no quick and easy way to keep the records of students and staff.The aim of this web application is to reduce the workload ,save significant staff time and make it more organized to access information by the students.

Some other benefits are:

1. Automated and web-based for easy accessibility.
2. It excludes paperwork and the possibility of making mistakes while calculating the final attendance and final internal marks.
3. It is user-friendly and handy

6.2 Future Enhancements

The future enhancement of the project involves intimidating the parents of the ward's attendance and internal score after every internal.The front-end can be advanced too.

BIBLIOGRAPHY

- Django documentation
- Blogs
- <https://www.djangoproject.com/>
- <https://getbootstrap.com/>
- <https://www.tutorialspoint.com/>