**1. INTRODUCTION**

# 1.1 About the Organization

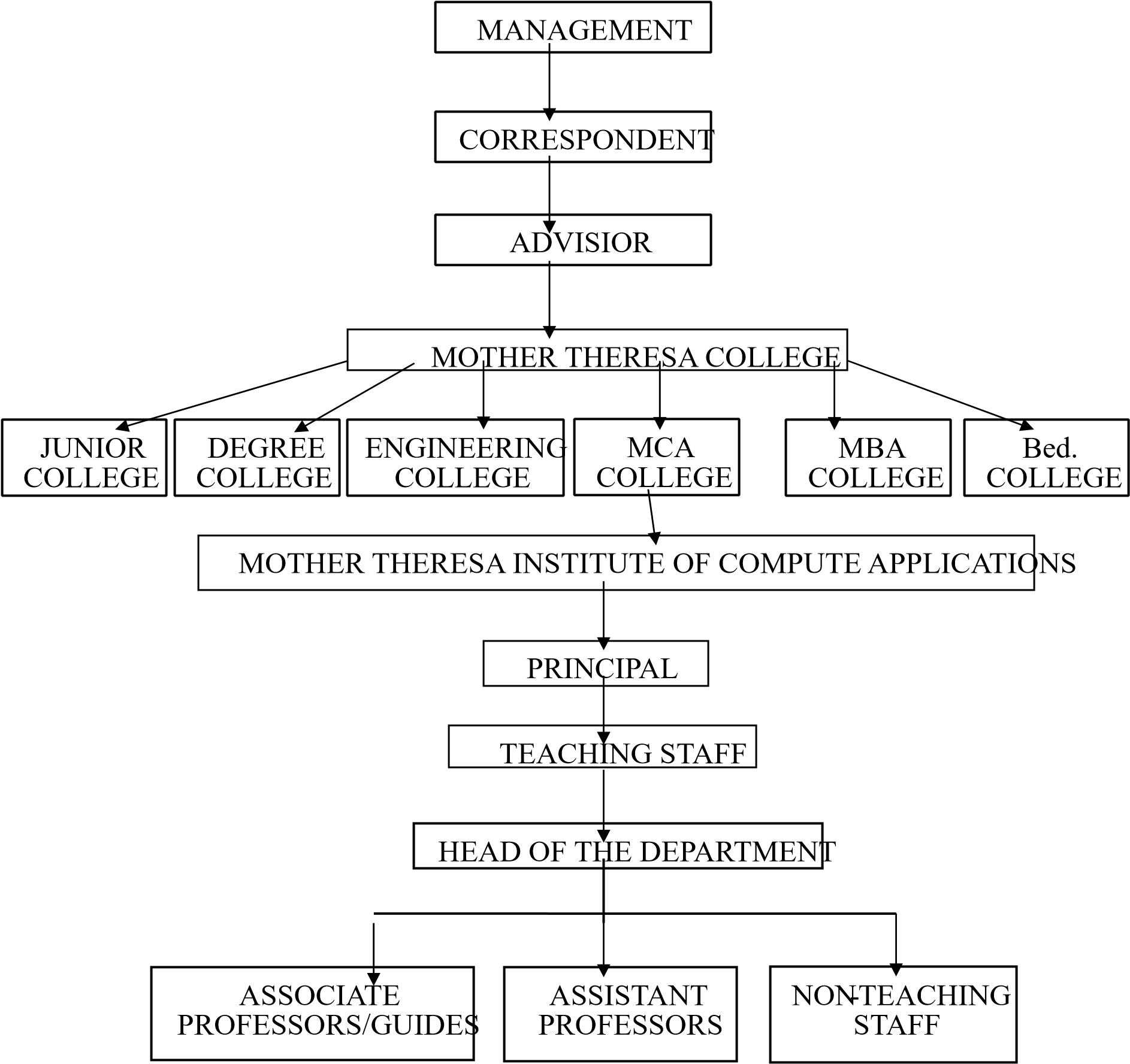
The college MOTHER THERESA INSTITUTIONS was located 5km away from Palamaner. It is one of the best colleges in Sri Venkateswara University. The Chairman of the college is Mr. SUNIL M R. The college is affiliated to S.V University. The principal of the college is Mr. R. PABHAKAR NAIDU The college was maintained well with well experienced lecturers and having courses like MBA, MCA,Degree, B. TECH Etc.

Colleges has well equipped facilities which include Lab, Library, and Playground. In the library number of latest edition text books, Journals, Monthly Magazines etc., and in the labs nearly 450+ latest configured systems are available for imparting Hands-on Experience to Students. It is located in a very Clam and Go Green Environment Which Helps students to make Mind piece and learn better. There are 4 blocks in the college. The maintenance of the college is very Systematic Manner with Qualified lecturers for every campus.

Nothing is permanent except change. The ongoing advances in computer communications technology continue to have profound effect on the way people work and play. Both the technology itself and the expectations of the people who use it are altering the features of the information system that analysis, design and the widespread deployment of information system in changing the very nature of the society in which the systems are used. The deployment of the information system has played a dominant role in evolution of information economy.

MTCA Institute of Post graduate Studies very popularly known as MTCA has emerged as a major technological Institute managed by MTCA SOCIAL AND EDUCATIONAL SOCIETY, Palamaner. The MTCA Society has taken the lead role of establish in the institute in academic year 2002. Sprawled over an area of 25acres, permanent infrastructural facilities are being developed near Palamaner-Madanapalli state Highway at Melmoi village. MTCA has secured the approval from the AllIndia Council of Technical Education (AICTE), New Delhi, and Government of Andhra Pradesh and is affiliated to Sri Venkateswara University, Tirupathi.

MTCA offers admission into the professional courses MCA with annual intake of 60 seats each. The Institution is governed by the chairman Sri RavindraBabu Garu as Correspondent/Secretary with the support of Sri Rajendra Reddy Garu an Academic Director, and other Board Members consisting of the eminent personalities from different fields.



# Case Study On Student Behaviour Management System

Mother Theresa Institute of Computer Applications (MTICA), established in 2002, is a renowned institution with a strong reputation for academic excellence. The institution offers a Master's degree in Computer Applications (MCA) and has successfully graduated 20 batches of students. MTICA currently caters to over 6,500 students across various streams, with the MCA department having an annual intake of 180 students, resulting in a total intake of 360 students over two years.

MTICA is known for its disciplined academic environment, but the administration recognizes that managing student behavior, ensuring adherence to academic standards, and promoting a positive learning environment require more proactive and structured solutions.

**Project Scope and Objectives:**

The project, **Student Behaviour Management System (SBMS)**, was conceptualized to track, monitor, and improve student behavior within the institution. The MCA department was chosen as the primary scope for this study due to the department’s growth, increasing student body, and the need for a tailored system to manage behavior effectively.

The **primary objectives** of the SBMS are:

1. **Track and Monitor Student Behaviour:** To create a centralized database for recording student behavior, including both positive and negative actions.
2. **Feedback Mechanism:** To provide a real-time feedback system for students on their behavior and academic progress.
3. **Disciplinary Action Recording:** To ensure that any disciplinary actions, warnings, or suspensions are logged accurately for follow-up.
4. **Analytics and Reporting:** To generate reports that can be reviewed by faculty, department heads, and the academic committee to identify behavior trends and take necessary actions.
5. **Promote Positive Behaviour:** To encourage good conduct through rewards, recognition, and incentives.
6. **Collaboration and Communication:** To facilitate effective communication between students, faculty, administration, and the academic committee for better coordination.

**Institutional Background:**

MTICA has established itself as a reputable institution offering advanced courses in Computer Applications. The institute's administration and faculty are committed to upholding high academic standards and fostering a learning environment that supports personal and professional growth for students.

However, due to the increasing number of students and the growing complexity of managing their behavior, there is a need for a more structured approach to:

* Monitor and manage student conduct effectively.
* Ensure better communication and follow-up on student behavior.
* Empower faculty and administrators with tools to address behavior-related concerns in a timely and efficient manner.

The **MCA Department**, which has witnessed an increase in student intake over the years, was selected as the pilot department for this project due to its size, visibility, and influence in the institution.

**Stakeholder Involvement:**

The development and implementation of SBMS required close collaboration with various stakeholders:

1. **Director/Principal:**
   * **Role:** Provide approval, strategic direction, and oversight of the project.
   * **Responsibilities:** Ensure the system aligns with institutional policies and objectives, approve budgets, and monitor overall progress.
2. **Academic Committee:**
   * **Role:** Review behavior reports and make policy recommendations.
   * **Responsibilities:** Monitor academic performance and behavior outcomes, suggest policy improvements, and assess system efficiency.
3. **Department Heads (MCA Department):**
   * **Role:** Oversee the system's implementation and evaluate student behavior. o **Responsibilities:** Ensure department-specific issues are addressed and feedback is provided for improving the system.
4. **Faculty Members:**
   * **Role:** Daily interaction with students, reporting incidents, and providing behavioral feedback. o **Responsibilities:** Record and report student behavior, engage with the system to monitor student progress, and collaborate with other faculty members.
5. **Students:**
   * **Role:** Direct participants in the behavior monitoring system.
   * **Responsibilities:** Adhere to the institution's conduct policies, engage with the feedback system, and work toward improving their behavior.
6. **IT Department:**
   * **Role:** Develop and maintain the SBMS.
   * **Responsibilities:** Design and implement the system, ensure security and accessibility, and provide technical support.

**Project Design and Approach:**

1. **System Features:**

* **Student Behaviour Tracker:** A dashboard where faculty can record any behavioral incident, whether positive or negative. This includes class attendance, participation, punctuality, adherence to codes of conduct, etc. o **Disciplinary Action Tracker:** A system to log warnings, sanctions, or any other disciplinary measures for students, ensuring there’s a complete record of student actions.
* **Feedback & Reporting:** Students receive immediate feedback on their behavior, which can be positive (good attendance, active participation) or corrective (disciplinary action, poor performance).
* **Analytics Dashboard:** Administrative users can analyze trends in student behavior across departments, identifying at-risk students and assessing the effectiveness of disciplinary actions.

## 3.Existing System

In our study, we took MCA College/Department as a scope to be investigated

preliminary because the college has good reputation and was established in 2002 and till now 20 batches are passed from this college with their master’s degree. In this Study we found that title given for our project has a scope for implementation on this institution, hence we contacted respective Director / Principal / Department Head for Details.

In Existing System all the data is being maintained as Handwritten and in Note Books are being maintained no proper format are adhered, and it cumbersome to handle situation when it comes to Accreditation Inspections and Affiliation inspection.

**4.Proposed System:**

Proposed system deals with right data tracking, because every year the committee will be changing and the following the committees should be constituted with existing staff and based on environment and circumstances. The data related to the committee constituted and its executions should be tracked and committed reports should be maintained and also status of execution will be tracked and evaluated timely basis for which application planned will act as wizard between the system and user. This application will hold the data of previous batches and its related activities there by department can take this data for further analysis and future correcting and for smooth execution. The proposed system has user friendly interactive menu’s which will be give Economical, Technical and Operational easiness for implementation.

The proposed system has following advantages they are:

* Reduce paperwork.
* Reduce the manual work.
* Time redundancy.
* Data security is high.
* Greater Efficiency.
* Easily understandable system and easy to use.
* Recording of details and searching for information is easily in these system,

## 5.Hardware and Software Requirements

As the Minor Project is developed under the Guidance of Department, the Hardware and Software are provided in our Lab, Hence the Specifications have compatibility. The required software and hardware are installed and configured. Software Requirements

Windows Operating System: Windows 10 (32 bit (or) 64 bit) – **Interface Require Min 2 GB RAM, Max 4 GB**

Django (Frame Work)

Python - **Language**

Java Script/HTML & CSS – **Frontend Tool**

Report Generation in Django Frame Work – **Report Designing Tool**

MS-Office **– Documenting Tool**

**Front End Tool: Django (frame work 4)**

Processor : Dual Core 2.6Ghz (Recommended)

RAM : 2 GB

Hard disk : 3 GB Free space

Screen Resolution : 1024 X 768 (Direct X capable 9)

Back End Tool: SQL Lite

Processor : Dual Core 2.6 GHz (minimum)

RAM : 2 GB

Hard disk : 3 GB Free space

Reporting Tool: On click Tool Patching Django & Sql Lite

Processor : Single Core 1.0 Ghz (minimum)

RAM : 2 GB Minimum

Hard disk : 10 GB Free Space

Language: Python (3.10.6)

Processor : Single Core 1.0 Ghz (minimum)

RAM : 2 GB Minimum

Hard disk : 10 GB Free Space

Front End Design Tools: Java Script/HTML & CSS

Processor : Dual Core 2.6 Ghz (minimum)

RAM : 4 GB Minimum

Hard disk : 20 GB Free Space

### 6.Feasibility Study Report

A Feasibility study was conducted to select the best system that meets performance requirements and this was entailed an identification description, an evaluation of candidate systems, and the selection of the best systems.

In Deciding on this system to design, the three considerations are involved in feasibility analysis: **Technically, Economically** and **operationally** feasible as per the following grounds.

The result of the Feasibility Study provides us with the following facts:

* The automated system would increase the efficiency of the system.
* The automated system would increase customer's satisfaction.
* The automated system has many requirements such as Efficiency cost effectiveness, prompt service, Reliability.
* The automated system would add to the security features of the system
* The automated system should be simple to use, incorporate all necessary services and maintainable.
* This will cause some changes in the organization.

### Technical Feasibility

This is concerned with specifying the equipment’s and the software to satisfy the user requirements. The technical needs of the system vary considerably but might include:

* The facility to produce outputs in a given time.
* Response time under certain conditions.
* Ability to process a certain volume of transactions at a specified speed.

Technical feasibility centers on the existing computer system, hardware, software etc., and to what extent it can support the system. In examining the technical feasibility, the configuration of the system is given more importance than the actual hardware.

The result of the Technical Feasibility Study is the basis for the documents against Specific hardware and software products can then be evaluated keeping in view the logical needs. In case of **STUDENT BEHAVIOUR MANAGEMENT SYSTEM** project all the requirements

needed technically like Hardware/ Software required or Environment to Deploy the developed Application is having feasibility, which are specified in Hardware and Software

Requirements. Hence we say that the

**STUDENT BEHAVIOUR MANAGEMENT SYSTEM** is having Technical Feasibility. **Economic Feasibility**

Economic feasibility is a study of cost benefit, keeping in view the project is evaluated with the Principle of Cost Benefit Analysis which, estimates and totals up the equivalent money value of the benefits and costs of projects to establish whether they are worthwhile. We could able to acquire the Economic Feasibility because of utilization of **Existing resources** which client has, if the **Meet the proposed requirem**ents in course of action, the context of direct and indirect benefits and costs to the organization will be reduced, hence we came to conclusion that the project **STUDENT BEHAVIOUR MANAGEMENT SYSTEM** is economically Feasible.

**Operational Feasibility.**

Operational Feasibility is a measure of how well a Proposed System Solve the Problems and Takes Advantage of the Opportunities identified during scope definition. The Project also complies with this sort of feasibility as it would solve e the problems of Database connectivity and lack of an interactive Front end in the Application. Moreover, while Designing the Front End, the Design standards are maintained which would present a Feel Good environment to the viewer/user while operating, hence an operational person with minimum data flow awareness about Application and with minimum Computer Knowledge can handle application successfully. Hence we can say that **STUDENT BEHAVIOUR MANAGEMENT SYSTEM** Project is operationally feasible.

# 7.System Design

System Design phase follows system analysis phase. Design is maintaining record proof divisions and providing a blue print for the implementation phase. Design is the bridge between; system analysis and system implementation.

System design is transition from a user oriented, document oriented to programmers or database personnel. The design is a solution a “how to” approach to the creation a new system. This is composed of several steps. It provides the understanding and procedural details necessary for implementing the system recommended in the feasibility; study. Design goes through logical and physical stage of development, logical design review the present physical system, prepare input and output specification, detail the implementation plan, and prepare a logical design walkthrough.

## OBJECTIVES OF DESIGN SDLC METHOLOGIES

This document play a vital role in the development of life style (SDLC) as it describes the complete requirement of the system. It means for use by developers and will be the basis during testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

**PROCESS MODELS**

* The **Waterfall** Model.
* The process of prototype development.
* The **Incremental** Development Model.
* The **spiral** model.
* The **phases** of **iterative development**.
* The principles of **agile methods**.

**Waterfall** Model.

**The Waterfall model** is the earliest SDLC **approach** that was used for **software development**. **The waterfall Model** illustrates the **software development process** in a linear sequential flow. This means that any phase in the **development process** begins only if the previous phase is complete.

**Prototype Development**

The Prototyping Model is a systems **development** method (SDM) in which p**rototype** (an early approximation of a final system or product) is built, tested, and then reworked as necessary until an acceptable **prototype** is finally achieved from which the complete system or product can now be **developed**. **Incremental** Development Model

The incremental build model is a method of software development where the product is designed, implemented and tested incrementally until the product is finished. It involves both development and maintenance. The product is defined as finished when it satisfies all of its requirements.

**SPIRAL MODEL:**

The spiral model is a risk-driven software development process model. Based on the unique risk patterns of a given project, the spiral model guides a team to adopt elements of one or more process models, such as incremental, waterfall, or evolutionary prototyping.

**Iterative** and **Incremental development** is any combination of both **iterative** design or **iterative** method and **incremental** build model for **development**. ... In software, the relationship between **iterations** and increments is determined by the overall software **development** methodology and software **development process**.

**SPIRAL MODEL:**

SPIRAL MODEL was defined by Barry Boehm in his 1988 article. “A Spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration models. As originally envisioned, the iterations were typically 6 months to 2 year long. Each phase starts with a design goal and ends with a client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.

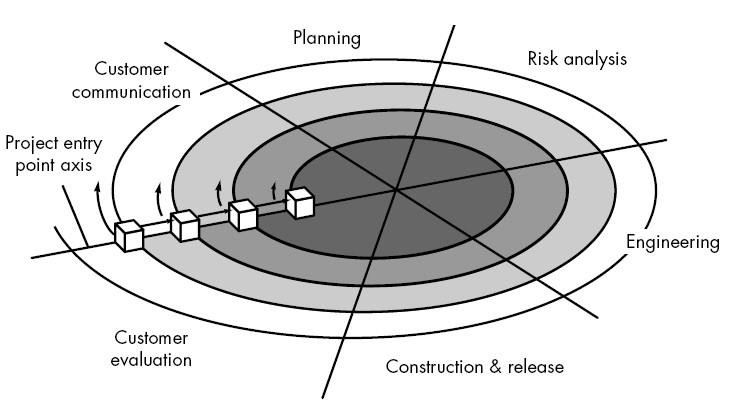
The steps for Spiral Model can be generalized as follows:

* The new system requirements are defined in as much details as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
* A preliminary design is created for the new system.
* A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
* A second prototype is evolved by a fourfold procedure:
  1. Evaluating the first prototype in terms of its strengths, weakness and risks.
  2. Defining the requirements of the second prototype
  3. Planning a designing the second prototype
  4. Constructing and testing the second prototype
* At the customer option, the entire project can be aborted if the risk is deemed but great. Risk factors might involve development cost overruns, operating-cost, miscalculations or any other factor that could, in the customer’s judgement, resulting

in a less-than satisfactory final product.

* The existing prototype is evaluated in the same manner as was the previous prototype and if necessary, another prototype is developed from its according to the fourfold procedure outlined above.
* The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
* The final system is constructed, based on the refined prototype
* The final system is thoroughly evaluated and tested. Routine maintenance is carried on a continuing basis to prevent large scale failures and to minimize down time

The following diagram shows how a spiral model acts like:



**ADVANTAGES**

* Estimates\*i.e. budget, schedule etc.,) because more realistic as work progressed, because important issues discovered earlier.
* It is more able to cope with the changes that are software development generally entails.
* Software engineers can get their hands in and start worrying on the core of a project earlier

**APPLICATION DEVELOMENT**

* Software Architecture: Software Architecture consists of One Tier, Two Tier, Three Tier and A “tier” can also be referred to as a “layer”.
* Three layers involved in the application namely Presentation Layer, Business Layer and

Data Layer. Let’s see each layer in detail:

* Presentation Layer: It is also known as Client layer. Top most layer of an application. This is the layer we see when we use software. By using this layer we can access the webpages. The main functionality of this layer is to communicate with the at most Application layer. This layer passes the information which is given by the user in terms of keyboard actions, mouse Click to the Application Layer.

* Application Layer: It is also known as Business Logic Layer which is also known as logical layer. As per the Gmail login page example, once user clicks on the login button, Application layer interacts with Database layer and sends required information to the Presentation layer. It controls an application’s functionality by performing detailed processing. This layer acts as a mediator between the Presentation and the Database layer. Complete business logic will be written in this layer.

* Data Layer: The data is stored in this layer. Application layer communicates with Database layer to retrieve the data. It contains methods that connects the database and performs required action e.g.: insert, update, delete etc.

**PERFORMANCE REQUIREMENTS**

Performance is measured in terms of the output provided by the application. Requirement specification plays an important part in the analysis of a system. Only when the requirement specifications are properly given, it is possible to design a system, which will fit into required environment. It rests largely in the part of the users of the existing system to give the requirement specifications because they are the people who finally use the system. This is because the requirements; have to be known during the initial stages so that the system can be designed according to those requirements. It is very difficult to change the system once it has been designed and on the other hand designing a system, which does not cater to the requirements of the user, is of no use.

The requirement specification for any system can be broadly stated as given below:

* The system should be able to interface with the existing system.
* The system should be accurate.
* The system should be better than the existing system.

The existing system is completely dependent on the user to perform all the duties

## Module Description

After detail address of case study existing and proposal system finally we have concluded to go with six modules of which four modules are been consider to over scope for development the modules are:

1. Login
2. Master
3. Transaction
4. Reports
5. Backup/Restore
6. Exit

**Login:**

This gateway interface to the application which enables authorized person to login to application with authenticated username and password.

**Master:**

This module is aimed to develop the key factors which supports the move the transaction of an application development in this module all primary data is been maintained for data integrity of an application.

1. Student Master
2. Batch Master
3. Course Master
4. College Master
5. Indispline Master
6. Sem Master
7. Offence Master

**Transaction:**

The functionality of this modules is to record, the real time transaction happening in the application development with reference against the masters for ensuring data integrity.

**Reports:**

The Reporting Module was designed to provide a feature-rich and user-friendly web interface for managing reports within openMRS.in addition, the Reporting Module provides a flexible and extensible API that module developers can develop against to build their own reports and tools.

**Backup/Restore:**

Data is wealth, hence this module places a major role in saving the data externally, if any Technical Issues occur for the data regarding project the backup will help in getting data back which will be taken periodically

**Exit:**

This module is end to make the user to come out of the application after successfully Execution or if user wishes abrupt exit.

# Normalization

Normalization in DBMS: 1NF, 2NF, 3NF and BCNF in Database. Normalization is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly.

Here are the most commonly used normal forms:

* First normal form(1NF)
* Second normal form(2NF)
* Third normal form(3NF)
* Boyce & Codd normal form (BCNF)

## First normal form (1NF)

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values.

It should hold only atomic values.

## Second normal form (2NF)

A table is said to be in 2NF if both the following conditions hold:

* Table is in 1NF (First normal form)
* No non-prime attribute is dependent on the proper subset of any candidate key of table.

An attribute that is not part of any candidate key is known as non-prime attribute.

## Third Normal form (3NF)

A table design is said to be in 3NF if both the following conditions hold:

* Table must be in 2NF
* [Transitive functional dependency o](https://beginnersbook.com/2015/04/transitive-dependency-in-dbms/)f non-prime attribute on any super key should be removed.

An attribute that is not part of any [candidate key i](https://beginnersbook.com/2015/04/candidate-key-in-dbms/)s known as non-prime attribute.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

* X is a [super key o](https://beginnersbook.com/2015/04/super-key-in-dbms/)f table
* Y is a prime attribute of table

## Boyce Code normal form (BCNF)

It is an advance version of 3NF that’s why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every [functional dependency X](https://beginnersbook.com/2015/04/functional-dependency-in-dbms/)->Y, X should be the super key of the table.

The attributes of a table is said to be dependent on each other when an attribute of a table uniquely identifies another attribute of the same table.

For example: Suppose we have a student table with attributes: Stu\_Id, Stu\_Name, Stu\_Age. Here Stu\_Id attribute uniquely identifies the Stu\_Name attribute of student table because if we know the student id we can tell the student name associated with it. This is known as functional dependency and can be written as Stu\_Id->Stu\_Name or in words we can say Stu\_Name is functionally dependent on Stu\_Id.

**Formally**:

If column A of a table uniquely identifies the column B of same table then it can represented as A->B (Attribute B is functionally dependent on attribute A)

## Types of Functional Dependencies

* Trivial functional dependency
* non-trivial functional dependency
* Multivalued dependency
* Transitive dependency

## BACKEND TOOL DESCRIPTION

**Introduction to SQLite what is sqllite:**

SQLite is a lightweight, serverless, self-contained, SQL database engine. It's ideal for small to medium-sized applications due to its simplicity, speed, and zero-configuration nature. SQLite is commonly used in mobile apps, desktop software, and embedded systems. It offers a reliable and efficient way to store and manage data locally within your applications.

**Sqlite Basic:**

* **SQLite:** Lightweight, self-contained database.
* **Create Table:** CREATE TABLE users (id INTEGER PRIMARY KEY, name TEXT, age INTEGER);
* **Insert Data:** INSERT INTO users (name, age) VALUES ('Alice', 30);
* **Retrieve Data:** SELECT \* FROM users;
* **Update Data:** UPDATE users SET age = 31 WHERE name = 'Alice';
* **Delete Data:** DELETE FROM users WHERE id = 1;
* **Command Line:** sqlite3 mydatabase.db to interact.
* **Programming:** Use libraries like sqlite3 in Python or SQLiteOpenHelper in Android.
* **Security:** Consider encryption for sensitive data.
* **Explore More:** Learn advanced queries, transactions, and indexing.

**History of sqlite:**

* **Birth of SQLite:** Created by D. Richard Hipp in 2000.
* **Early Releases:** Initial versions relied on the GDBM library for storage.
* **B-Tree Adoption:** SQLite 2.0 introduced a custom B-tree storage engine for improved performance and scalability.
* **Transaction Support:** This version also brought robust transaction capabilities, ensuring data integrity.
* **Internationalization:** SQLite 3.0 added support for various character encodings and internationalization features.
* **Manifest Typing:** This version enhanced data type support, allowing for more precise data definition and manipulation.
* **Open-Source Philosophy:** SQLite has always been open-source, making it freely available and customizable.
* **Widespread Adoption:** Its simplicity, reliability, and zero-configuration nature led to its adoption in numerous applications, including mobile devices, web browsers, and embedded systems.
* **Continuous Development:** The SQLite project continues to evolve, with regular updates and improvements.
* **Modern Impact:** SQLite remains a popular choice for developers due to its ease of use, performance, and security. **SQLite Syntax and Structure sqlite keywords:**

Here are some of the most common SQLite keywords:

**Data Definition Language (DDL) Keywords:**

* **CREATE TABLE:** Defines a new table.
* **DROP TABLE:** Deletes an existing table.
* **ALTER TABLE:** Modifies the structure of a table.
* **CREATE INDEX:** Creates an index on a table column.
* **DROP INDEX:** Deletes an index.

**Data Manipulation Language (DML) Keywords:**

* **INSERT INTO:** Inserts new rows into a table.
* **SELECT:** Retrieves data from a table.
* **UPDATE:** Modifies existing data in a table.
* **DELETE FROM:** Deletes rows from a table.

**Other Important Keywords:**

* **PRIMARY KEY:** Defines a unique identifier for a table.
* **FOREIGN KEY:** Defines a reference to a primary key in another table.
* **NOT NULL:** Ensures that a column cannot contain null values.
* **UNIQUE:** Ensures that a column contains unique values.
* **CHECK:** Defines a constraint that must be satisfied for a column.
* **DEFAULT:** Specifies a default value for a column.
* **AUTOINCREMENT:** Automatically increments a numeric value for each new row.
* **WHERE:** Specifies a condition for filtering data.
* **ORDER BY:** Sorts the result set.
* **GROUP BY:** Groups rows based on a specific column.
* **HAVING:** Filters groups of rows.
* **JOIN:** Combines rows from two or more tables.
* **UNION:** Combines the result sets of two or more SELECT statements.
* **INTERSECT:** Returns the intersection of two result sets.
* **EXCEPT:** Returns the rows from the first result set that are not in the second.
* **LIMIT:** Limits the number of rows returned.
* **OFFSET:** Skips a specified number of rows.
* **AS:** Used to assign an alias to a column or table.
* **IN:** Checks if a value is in a list of values.
* **BETWEEN:** Checks if a value is within a range.
* **LIKE:** Matches patterns in strings.
* **IS NULL:** Checks if a value is null.
* **IS NOT NULL:** Checks if a value is not null.
* **AND:** Logical AND operator.
* **OR:** Logical OR operator.
* **NOT:** Logical NOT operator.
* **EXISTS:** Checks if a subquery returns any rows.
* **CASE:** Conditional expression.

**SQLLITE CREATE TABLE**

In Sqlite, CREATE TABLE statement is used to create a new table in the database. To create a table, you have to name that table and define its columns and datatype for each column.

**Syntax:**

* 1. CREATE TABLE table\_name
  2. (
  3. column1 datatype [ NULL | NOT NULL ],
  4. column2 datatype [ NULL | NOT NULL ],
  5. ...
  6. column\_n datatype [ NULL | NOT NULL ]
  7. );

**Parameters used in syntax**

* **table\_name:** It specifies the name of the table which you want to create.
* **column1, column2, ... column n:** It specifies the columns which you want to add in the table. Every column must have a datatype. Every column should either be defined as "NULL" or "NOT NULL". In the case, the value is left blank; it is treated as "NULL" as default.

**Sqlite CREATE TABLE Example with primary key**

* 1. CREATE TABLE users (
  2. user\_id INTEGER PRIMARY KEY AUTOINCREMENT,
  3. username TEXT NOT NULL UNIQUE,
  4. email TEXT NOT NULL UNIQUE,
  5. password TEXT NOT NULL
  6. );

**What is Primary key**

A primary key is a single field or combination of fields that contains a unique record. It must be filled. None of the field of primary key can contain a null value. A table can have only one primary key.

**CREATE TABLE AS Statement**

The CREATE TABLE AS statement is used to create a table from an existing table by copying the columns of existing table.

**Syntax:**

* 1. CREATE TABLE new\_table
  2. AS (SELECT \* FROM old\_table);

**Create Table Example: copying all columns of another table**

In this example, we are creating a "newcustomers" table by copying all the columns from the already existing table "Customers".

* + 1. CREATE TABLE newcustomers
    2. AS (SELECT \* FROM customers WHERE customer\_id < 5000);

Table created.

This table is named as "newcustomers" and having the same columns of "customers" table.

**Create Table Example: copying selected columns of another table Syntax:**

* + - 1. CREATE TABLE new\_table AS
      2. SELECT column1, column2, column3
      3. FROM original\_table
      4. WHERE condition; Let's take an example:

1. CREATE TABLE newcustomers2
2. AS (SELECT customer\_id, customer\_name
3. FROM customers
4. WHERE customer\_id < 5000);

The above example will create a new table called "newcustomers2". This table includes the specified columns customer\_id and customer\_name from the customers table.

**Create Table Example: copying selected columns from multiple tables Syntax:**

* 1. CREATE TABLE new\_table
  2. AS (SELECT column\_1, column2, ... column\_n
  3. FROM old\_table\_1, old\_table\_2, ... old\_table\_n);

Let's take an example: Consider that you have already created two tables "regularcustomers" and "irregularcustomers".

The table "regularcustomers" has three columns rcustomer\_id, rcustomer\_name and rc\_city.

* 1. CREATE TABLE "regularcustomers"
  2. ( "RCUSTOMER\_ID" NUMBER(10,0) NOT NULL ENABLE,
  3. "RCUSTOMER\_NAME" VARCHAR2(50) NOT NULL ENABLE,
  4. "RC\_CITY" VARCHAR2(50)
  5. )
  6. /

The second table "irregularcustomers" has also three columns ircustomer\_id, ircustomer\_name and irc\_city.

* 1. CREATE TABLE "irregularcustomers"
  2. ( "IRCUSTOMER\_ID" NUMBER(10,0) NOT NULL ENABLE,
  3. "IRCUSTOMER\_NAME" VARCHAR2(50) NOT NULL ENABLE,
  4. "IRC\_CITY" VARCHAR2(50)
  5. ) /

In the following example, we will create a table name "newcustomers3" form copying columns from both tables.

**Example:**

* 1. CREATE TABLE newcustomers3
  2. AS (SELECT regularcustomers.rcustomer\_id, regularcustomers.rc\_city, irreg ularcustomers.ircustomer\_name
  3. FROM regularcustomers, irregularcustomers
  4. WHERE regularcustomers.rcustomer\_id =

irregularcustomers.ircustomer\_id

* 1. AND regularcustomers.rcustomer\_id < 5000);

**Sqlite ALTER TABLE Statement**

The ALTER TABLE statement in SQLite allows you to modify the structure of an existing table

The ALTER TABLE statement in SQLite allows you to modify the structure of an existing table. However, it has limited capabilities compared to other database systems.

**How to add column in a table Syntax:**

ALTER TABLE table\_name

ADD COLUMN column\_name data\_type;

**Example:**

Consider that already existing table customers. Now, add a new column customer\_age into the table customers.

1. ALTER TABLE customers
2. ADD customer\_age varchar2(50);

Now, a new column "customer\_age" will be added in customers table.

**How to add multiple columns in the existing table Syntax:**

* 1. ALTER TABLE table\_name
  2. ADD (column\_1 column-definition,
  3. column\_2 column-definition,
  4. ...
  5. column\_n column\_definition);

**Example**

1. ALTER TABLE customers
2. ADD (customer\_type varchar2(50),
3. customer\_address varchar2(50));

Now, two columns customer\_type and customer\_address will be added in the table customers.

**How to modify column of a table Syntax:**

* 1. ALTER TABLE table\_name
  2. MODIFY column\_name column\_type; **Example:**
  3. ALTER TABLE customers
  4. MODIFY customer\_name varchar2(100) not null; Now

the column column\_name in the customers table is modified to varchar2 (100) and forced the column to not allow null values.

ALTER TABLE table\_name

**How to modify multiple columns of a table Syntax:** 1. UPDATE table\_name

* + 1. SET column1 = new\_value1, column2 = new\_value2, ...
    2. WHERE condition;

**Example:**

1. UPDATE customers
2. SET email = 'new\_email@example.com', phone\_number = '123-456-7890'
3. WHERE customer\_id = 123;

This will modify both the customer\_name and city columns in the table.

**How to drop column of a table Syntax:**

* 1. ALTER TABLE table\_name
  2. DROP COLUMN column\_name; **Example:**

1. ALTER TABLE customers
2. DROP COLUMN customer\_name;

This will drop the customer\_name column from the table.

**How to rename column of a table Syntax:**

1. ALTER TABLE table\_name
2. RENAME COLUMN old\_name to new\_name;

**Example:**

1. ALTER TABLE customers
2. RENAME COLUMN customer\_name to cname; This will rename the column customer\_name into cname.

**How to rename table Syntax:**

* 1. ALTER TABLE table\_name
  2. RENAME TO new\_table\_name;

**Example:**

* 1. ALTER TABLE customers
  2. RENAME TO retailers;

**SQLite DROP TABLE Statement**

The DROP TABLE statement is used to delete an entire table from a SQLite database, including all its data and structure.  **Syntax**

* + 1. DROP TABLE table\_name;
    2. [ CASCADE CONSTRAINTS ]
    3. [ PURGE ];

**Parameters schema\_name:** It specifies the name of the schema that owns the table.

**table\_name:** It specifies the name of the table which you want to remove from the SQLite database.

**CASCADE CONSTRAINTS:** It is optional. If specified, it will drop all referential integrity constraints as well.

**PURGE:** It is also optional. If specified, the table and its dependent objects are placed in the recycle bin and can?t be recovered.

**DROP TABLE Example**

1. DROP TABLE customers;

This will drop the table named customers.

**DROP TABLE Example with PURGE parameter**

1. DROP TABLE customers PURGE

This statement will drop the table called customers and issue a PURGE so that the space associated with the customers table is released and the customers table is not placed in recycle bin. So, it is not possible to recover that table if required.

**SQLite Global Temporary tables**

In SQLite, global temporary tables are temporary tables that are visible to all database connections within the same process. They are created using the CREATE TEMP TABLE syntax.

**Syntax**

1. CREATE GLOBAL TEMPORARY TABLE table\_name
2. ( column1 datatype [ NULL | NOT NULL ],
3. column2 datatype [ NULL | NOT NULL ],
4. ...
5. column\_n datatype [ NULL | NOT NULL ]
6. );

**Parameters table\_name:** The parameter table\_name specifies the global temporary table that you want to create.

**column1, column2, ... column\_ n:** It specifies the column that you want create in the global temporary table. Every column must have a datatype and should be defined as NULL or NOTNULL. If the value is left blank, it is by default treated as NULL.

**Example**

The following example specifies how to create a global temporary table

1. CREATE GLOBAL TEMPORARY TABLE students
2. ( student\_id numeric(10) NOT NULL,
3. student\_name varchar2(50) NOT NULL,

student\_address varchar2(50)

1. );

**SQLite Local Temporary tables**

Local temporary tables are temporary tables that are visible only to the current database connection. They are created using the CREATE TEMPORARY TABLE syntax. **Declare local temporary table Syntax**

1. DECLARE LOCAL TEMPORARY TABLE table\_name
2. ( column1 datatype [ NULL | NOT NULL ],
3. column2 datatype [ NULL | NOT NULL ],
4. ...
5. column\_n datatype [ NULL | NOT NULL ]
6. );

**Parameters table\_name:** The parameter table\_name specifies the local temporary table that you want to create.

**column1, column2,... column\_ n:** It specifies the column that you want create in the local temporary table. Every column must have a datatype and should be defined as NULL or NOTNULL. If the value is left blank, it is by default treated as NULL.

**SQLite View**

A view in SQLite is a virtual table based on the result-set of an SQL statement. It doesn't store actual data but rather provides a different way to look at existing data.

A view is created by a query joining one or more tables.

|  |  |
| --- | --- |
| **SQLite CREATE VIEW**  **Syntax:** |  |
| 1. | CREATE VIEW view\_name AS |
| 2. | SELECT column1, column2, ... |
| 3. | FROM table\_name |
| 4. | WHERE condition; **Parameters:** |

• **view\_name:** It specifies the name of the SQLite VIEW that you want to create.

**Example:**

Let's take an example to create view. In this example, we are creating two tables suppliers and orders first.

**Suppliers table:**

1.

* + 1. CREATE TABLE "SUPPLIERS"
    2. ( "SUPPLIER\_ID" NUMBER,
    3. "SUPPLIER\_NAME" VARCHAR2(4000),
    4. "SUPPLIER\_ADDRESS" VARCHAR2(4000)
    5. ) /

**Orders table:**

* + 1. CREATE TABLE "ORDERS"
    2. ( "ORDER\_NO." NUMBER,
    3. "QUANTITY" NUMBER,
    4. "PRICE" NUMBER
    5. ) /

Execute the following query to create a view name sup\_orders. **Create View Query:**

* + - 1. CREATE VIEW sup\_orders AS
      2. SELECT suppliers.supplier\_id, orders.quantity, orders.price
      3. FROM suppliers
      4. INNER JOIN orders
      5. ON suppliers.supplier\_id = supplier\_id
      6. WHERE suppliers.supplier\_name = 'VOJO';

**Output:**

View created.

0.21 seconds

You can now check the Oracle VIEW by this query:

1. SELECT \* FROM sup\_orders; **Output:**

SUPPLIER\_ID QUANTITY PRICE

3 35 70

3 26 125

3 18 100 3 rows returned in 0.00 seconds

**SQLite Update VIEW**

Views are virtual tables based on the result set of an SQL statement. They don't store actual data, so modifying them directly isn't possible.

|  |  |
| --- | --- |
| **Syntax:** |  |
| 1. | CREATE OR REPLACE VIEW view\_name AS |
| 2. | SELECT columns |
| 3. | FROM table |
| 4. | WHERE conditions; Or |

* + CREATE VIEW expensive\_products AS
  + SELECT product\_id, product\_name, price
  + FROM products
  + WHERE price > 100; **Example:**

Execute the following query to update the definition of Oracle VIEW called sup\_orders without dropping it.

* 1. CREATE or REPLACE VIEW sup\_orders AS
  2. SELECT suppliers.supplier\_id, orders.quantity, orders.price
  3. FROM suppliers
  4. INNER JOIN orders
  5. ON suppliers.supplier\_id = supplier\_id
  6. WHERE suppliers.supplier\_name = 'HCL'; You can now

check the Oracle VIEW by this query:

1. SELECT \* FROM sup\_orders;

**Output:**

|  |  |
| --- | --- |
| SUPPLIER\_ID | QUANTITY PRICE |
| 1 35 | 70 |
| 1 26 | 125 |
| 1 18 | 100 |
| row(s) 1 - 3 of 3 |  |

**SQLite DROP VIEW**

The DROP VIEW statement is used to remove or delete the VIEW completely.

**Syntax:**

1. DROP VIEW view\_name;

1. DROP VIEW sup\_orders;

**SQLite TRUNCATE TABLE**

However, you can achieve the same effect by using the DELETE statement without a In SQLite, TRUNCATE TABLE statement is used to remove all records from a table. It works same as DELETE statement but without specifying a WHERE clause. It is generally used when you don?t have to worry about rolling back

Once a table is truncated, it can?t be rolled back. The TRUNCATE TABLE statement does not affect any of the table?s indexes, triggers or dependencies.

**Syntax**

TRUNCATE TABLE [schema\_name.]table\_name

**Parameters**

**1)schema name:This** parameter specifies the name of the schema that the table belongs to it Is optional.

**2) table\_name:** It specifies the table that you want to truncate.

**SQLite TRUNCATE Table Example**

This will remove all rows from the customers table, leaving the table structure intact.

1. TRUNCATE TABLE customers;

**Output**

Table truncated.

1.11 seconds

While this effectively removes all rows, it's not as efficient as a true TRUNCATE operation in other database systems. SQLite optimizes this operation to be relatively fast, but it still involves processing each row individually.

**SQLite DELETE Table Example**

* 1. DELETE FROM table\_name; or
  2. DELETE TABLE customers;

**SQLite DISTINCT Clause**

The DISTINCT keyword in SQLite is used to eliminate duplicate rows from a query result set. It ensures that each row in the result is unique based on the specified columns.

**Syntax:**

* + - * 1. SELECT DISTINCT column1, column2, ...
        2. FROM table\_name
        3. WHERE condition; **Parameters:**

**Column Specificity:** You can apply DISTINCT to specific columns to eliminate duplicates based on those columns.

**NULL Values:** SQLite considers NULL values as distinct.

**Performance Considerations:** Using DISTINCT can sometimes impact query performance, especially for large datasets. Consider indexing the columns involved in the DISTINCT clause to improve performance.

**SQLite DISTINCT Example: (with multiple expressions)**

This query will return a list of unique pairs of customer IDs and product IDs, eliminating duplicate combinations.

**Execute this query:**

* + - 1. SELECT DISTINCT column1, column2, ...
      2. FROM table\_name
      3. WHERE condition;

**SQLite FROM Clause**

In this query, the FROM clause specifies the table from which you want to retrieve data.FROM clause is a mandatory clause in SELECT expression. It specifies the tables from which data is to be retrieved.

**Syntax:**

* + - 1. SELECT column1, column2, ...
      2. FROM table\_name
      3. WHERE condition;

**SQLite ORDER BY Clause**

In SQLite, ORDER BY Clause is used to sort or re-arrange the records in the result set. The ORDER BY clause is only used with SELECT statement.or This query will return all products, sorted by price in ascending order (lowest to highest).

**Syntax:**

* + 1. SELECT expressions
    2. FROM tables
    3. WHERE conditions
    4. ORDER BY expression [ ASC | DESC ]; **Parameters:**

**expressions:** It specifies columns that you want to retrieve. **tables:** It specifies the table name from where you want to retrieve records. **conditions:** It specifies the conditions that must be fulfilled for the records to be selected.

**ASC:** It is an optional parameter that is used to sort records in ascending order.

**DESC:** It is also an optional parameter that is used to sort records in descending order.

**SQLite ORDER BY Example: (without ASC/DESC attribute)**

The above example returns the first\_name ordered by last\_name in ascending order.

* + - * SELECT \*
      * FROM products
      * ORDER BY price;

**SQLite ORDER BY Example: (sorting in descending order)**

If you want to sort your result in descending order, you should use the DESC attribute in your ORDER BY clause:

* + - * SELECT \*
      * FROM products
      * ORDER BY price DESC;

**SQLite GROUP BY Clause**

In SQLite GROUP BY clause is used with SELECT statement to collect data from multiple records and group the results by one or more columns. By effectively using the GROUP BY clause, you can analyze and summarize data in your SQLite database.

**Syntax:**

* + - * 1. SELECT column1, aggregate\_function(column2)
        2. FROM table\_name
        3. GROUP BY column1;

**Parameters:**

**expression1, expression2, ... expression\_n:** It specifies the expressions that are not encapsulated within aggregate function. These expressions must be included in GROUP BY clause.

**aggregate\_function:** It specifies the aggregate functions i.e. SUM, COUNT, MIN, MAX or AVG functions.

**aggregate\_expression:** It specifies the column or expression on that the aggregate function is based on. **tables:** It specifies the table from where you want to retrieve records.

**conditions:** It specifies the conditions that must be fulfilled for the record to be selected.

* + - 1. ( "ITEM" VARCHAR2(4000),
      2. "SALE" NUMBER,
      3. "BILLING\_ADDRESS" VARCHAR2(4000)
      4. ) /

**SQLite GROUP BY Example: (with COUNT function)**

Let's take a table "customers"

Here we are creating a table named customers. This table doesn't have any primary key.

**Customer table:**

* + - 1. SELECT category,
      2. COUNT(\*) AS product\_count
      3. FROM products
      4. GROUP BY category;

SQLite HAVING Clause

In Oracle, HAVING Clause is used with GROUP BY Clause to restrict the groups of returned rows where condition is TRUE.

**Syntax:**

* + 1. SELECT expression1, expression2, ... expression\_n,
    2. aggregate\_function (aggregate\_expression)
    3. FROM tables
    4. WHERE conditions
    5. GROUP BY expression1, expression2, ... expression\_n
    6. HAVING having\_condition;

**Parameters:**

**expression1, expression2, ... expression\_n:** It specifies the expressions that are not encapsulated within aggregate function. These expressions must be included in GROUP BY clause.

**aggregate\_function:** It specifies the aggregate functions i.e. SUM, COUNT, MIN, MAX or AVG functions.

**aggregate\_expression:** It specifies the column or expression on that the aggregate function is based on. **tables:** It specifies the table from where you want to retrieve records. **conditions:** It specifies the conditions that must be fulfilled for the record to be selected.

**having\_conditions:** It specifies the conditions that are applied only to the aggregated results to restrict the groups of returned rows.

**SQLite UNION Operator**

The UNION operator in SQLite is used to combine the result sets of two or more SELECT statements. It's particularly useful when you want to combine data from different tables or different parts of the same table.

Each SELECT statement within the UNION operator must have the same number of fields in the result sets with similar data types.

**Syntax**

1. SELECT column1, column2, ...
2. FROM table1
3. WHERE condition1
4. UNION
5. SELECT column1, column2, ...
6. FROM table2
7. WHERE condition2;

**Parameters**

1. **expression1, expression2, ... expression\_n:** It specifies the columns that you want to retrieve.
2. **table1, table2:** it specifies the tables from where you retrieve the records.
3. **conditions:** it specifies the conditions that must be fulfilled for the records to be selected.

**Note:** The number of expressions must be same in both of the SELECT statements.

**SQLite INTERSECT Operator**

The INTERSECT operator in SQLite is used to combine the result sets of two or more SELECT statements and returns only the rows that are present in all of the result sets. It's like finding the intersection of sets in mathematics. **Syntax**

* 1. SELECT column1, column2, ...
  2. FROM table1
  3. WHERE condition1
  4. INTERSECT
  5. SELECT column1, column2, ...
  6. FROM table2
  7. WHERE condition2;

**Parameters**

1. **expression1, expression2, ... expression\_n:** It specifies the columns that you want to retrieve.
2. **table1, table2:** It specifies the tables that you want to retrieve records from.
3. **conditions:** it specifies the conditions that must be fulfilled for the records to be selected.

**SQLite INTERSECT Example: (with single expression)**

**SQLite EXCEPT operator**

SQLite doesn't directly support a **MINUS** operator.

However, you can achieve the same result by using the EXCEPT operator.

**EXCEPT Operator:**

The EXCEPT operator returns rows from the first query that are not present in the second query.

**Syntax:**

* + - 1. SELECT column1, column2, ...
      2. FROM table1
      3. WHERE condition1
      4. EXCEPT
      5. SELECT column1, column2, ...
      6. FROM table2
      7. WHERE condition2;

**Example:**

Consider two tables: orders\_north and orders\_south. Both tables have the same structure: order\_id, customer\_id, and order\_date.

To find orders that are only in the orders\_north table and not in the orders\_south table, you can use the EXCEPT operator:

Example1:

* + 1. SELECT order\_id
    2. FROM orders\_north
    3. EXCEPT
    4. SELECT order\_id
    5. FROM orders\_south;

This query will return a list of order\_id values that are unique to the orders\_north table.

**Key Points:**

* + The EXCEPT operator is a powerful tool for finding differences between datasets.
  + The columns in both SELECT statements must have the same data types.
  + The order of columns in both SELECT statements must be the same.

By effectively using the EXCEPT operator, you can perform complex data analysis and reporting tasks in SQLite.

**SQLite Joins**

Join is a query that is used to combine rows from two or more tables, views, or materialized views. It retrieves data from multiple tables and creates a new table.

**Join Conditions**

There may be at least one join condition either in the FROM clause or in the WHERE clause for joining two tables. It compares two columns from different tables and combines pair of rows, each containing one row from each table, for which join condition is true. **Types of Joins**

* + Inner Joins (Simple Join)
  + Outer Joins o Left Outer Join (Left Join) o Right Outer Join (Right Join) o Full Outer Join (Full Join)
  + Equijoins
  + Self Joins
  + Cross Joins (Cartesian Products)
  + Antijoins
  + Semijoins

**SQLite Procedures**

A procedure is a group of PL/SQL statements that can be called by name. The call specification (sometimes called call spec) specifies a java method or a third-generation language routine so that it can be called from SQL and PL/SQL.

**SQLite Function**

A function is a subprogram that is used to return a single value. You must declare and define a function before invoking it. It can be declared and defined at a same time or can be declared first and defined later in the same block.

**SQLiteCursor**

A cursor is a pointer to a private SQL area that stores information about the processing of a SELECT or DML statements like INSERT, UPDATE, DELETE or MERGE.

Cursor is a mechanism which facilitates you to assign a name to a SELECT statement and manipulate the information within that SQL statement.

## Front End Tools

### Django Frame Work

Django sometimes stylized as Django is a free and open-source, Python-based web framework that runs on a web server. It follows the model–template–views (MTV) architectural pattern It is maintained by the Django Software Foundation (DSF), an independent organization established in the US as a non-profit.

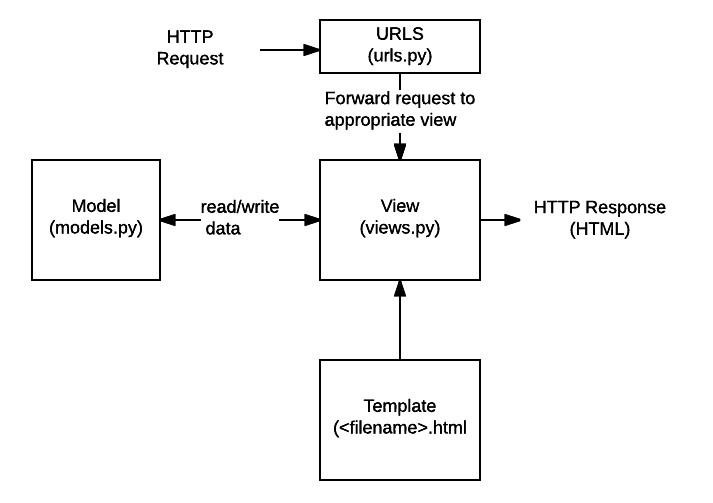
Django's primary goal is to ease the creation of complex, database-driven websites. The framework emphasizes [reusability a](https://en.wikipedia.org/wiki/Reusability)nd "plug ability" of components, less code, low coupling, rapid development, and the principle of [don't repeat yoursel](https://en.wikipedia.org/wiki/Don%27t_repeat_yourself)[f.[9] P](https://en.wikipedia.org/wiki/Django_(web_framework)#cite_note-9)ython is used throughout, even for settings, files, and [data models.](https://en.wikipedia.org/wiki/Data_model) Django also provides an optional administrative [create, read, update and delete i](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete)nterface that is generated dynamically through [introspection a](https://en.wikipedia.org/wiki/Type_introspection)nd configured via admin models.

Components

Despite having its own nomenclature, such as naming the callable objects generating the HTTP responses "views", the core Django framework can be seen as an MVC architecture It consists of an object-relational mapper (ORM) that mediates between data models (defined as Python classes) and a relational database ("Model"), a system for processing HTTP requests with a web templating system ("View"), and a regularexpressionbased URL dispatcher ("Controller").

Django's configuration system allows third party code to be plugged into a regular project, provided that it follows the reusable app conventions. More than 5000 packages are available to extend the framework's original behavior, providing solutions to issues the original tool didn't tackle: registration, search, API provision and consumption, Content Management System,

This extensibility is, however, mitigated by internal components' dependencies. While the Django philosophy implies loose coupling the template filters and tags assume one engine implementation, and both the author and admin bundled applications require the use of the internal ORM (Object Relational Model) None of these filters or bundled apps are mandatory to run a Django project, but reusable apps tend to depend on them, encouraging developers to keep using the official stack in order to benefit fully from the apps ecosystem.



Also included in the core framework are:

* a lightweight and standalone web server for development and testing
* a form serialization and validation system that can translate between HTML forms and values suitable for storage in the database
* a template system that utilizes the concept of inheritance borrowed from object-oriented programming
* a caching framework that can use any of several cache methods
* support for middleware classes that can intervene at various stages of request processing and carry out custom functions
* an internal dispatcher system that allows components of an application to communicate events to each other via pre-defined signals
* an internationalization system, including translations of Django's own components into a variety of languages
* a serialization system that can produce and read XML and/or JSON representations of Django model instances
* a system for extending the capabilities of the template engine
* an interface to Python's built-in unit test framework

Django is a Python framework that makes it easier to create web sites using Python. Django takes care of the difficult stuff so that you can concentrate on building your web applications.

Django emphasizes reusability of components, also referred to as DRY (Don't Repeat Yourself), and comes with ready-to-use features like login system, database connection and CRUD operations (Create Read Update Delete).

Django follows the MVT design pattern (Model View Template).

* Model - The data you want to present, usually data from a database.
* View - A request handler that returns the relevant template and content - based on the request from the user.
* Template - A text file (like an HTML file) containing the layout of the web page, with logic on how to display the data.

## Model

The model provides data from the database.

In Django, the data is delivered as an Object Relational Mapping (ORM), which is a technique designed to make it easier to work with databases.

The most common way to extract data from a database is SQL. One problem with SQL is that you have to have a pretty good understanding of the database structure to be able to work with it.

Django, with ORM, makes it easier to communicate with the database, without having to write complex SQL statements.

The models are usually located in a file called models.py.

## View

A view is a function or method that takes http requests as arguments, imports the relevant model(s), and finds out what data to send to the template, and returns the final result.

The views are usually located in a file called views.py.

## Template

A template is a file where you describe how the result should be represented.

Templates are often .html files, with HTML code describing the layout of a web page, but it can also be in other file formats to present other results, but we will concentrate on .html files.

Django uses standard HTML to describe the layout, but uses Django tags to add logic:



<



h1



>



My Homepage



<



/



h1



>



<



p



>



My name is



{{



firstname



}}



.



<



/



p



>



The templates of an application is located in a folder named



templates



.



## URLs

Django also provides a way to navigate around the different pages in a website.

When a user requests a URL, Django decides which *view* it will send it to.

This is done in a file called urls.py.

# JavaScript

JavaScript often abbreviated as JS, is a programming language and core technology of [the Web,](https://en.wikipedia.org/wiki/World_Wide_Web) alongside [HTML a](https://en.wikipedia.org/wiki/HTML)nd [CSS.](https://en.wikipedia.org/wiki/CSS) 99% of [websites u](https://en.wikipedia.org/wiki/Website)se JavaScript on the [client s](https://en.wikipedia.org/wiki/Client_(computing))ide for [webpage b](https://en.wikipedia.org/wiki/Web_page)ehavior.

[Web browsers h](https://en.wikipedia.org/wiki/Web_browser)ave a dedicated [JavaScript engine t](https://en.wikipedia.org/wiki/JavaScript_engine)hat executes the client [code.](https://en.wikipedia.org/wiki/Source_code) These engines are also utilized in some [servers a](https://en.wikipedia.org/wiki/Server_(computing))nd a variety of [apps.](https://en.wikipedia.org/wiki/Application_software) The most popular [runtime system f](https://en.wikipedia.org/wiki/Runtime_system)or non-browser usage is [Node.js.](https://en.wikipedia.org/wiki/Node.js)

JavaScript is a [high-level,](https://en.wikipedia.org/wiki/High-level_programming_language) often [just-in-time compiled l](https://en.wikipedia.org/wiki/Just-in-time_compilation)anguage that conforms to the [ECMAScript s](https://en.wikipedia.org/wiki/ECMAScript)tandard.[[11] I](https://en.wikipedia.org/wiki/JavaScript#cite_note-tc39-11)t has [d](https://en.wikipedia.org/wiki/Dynamic_typing)ynamic typin[g](https://en.wikipedia.org/wiki/Dynamic_typing)[, prototype-base](https://en.wikipedia.org/wiki/Prototype-based_programming)[d object-orientation,](https://en.wikipedia.org/wiki/Object-oriented_programming) and [first-c](https://en.wikipedia.org/wiki/First-class_function)lass function[s.](https://en.wikipedia.org/wiki/First-class_function) It is [multi-paradigm,](https://en.wikipedia.org/wiki/Programming_paradigm) supporting [event-driven](https://en.wikipedia.org/wiki/Event-driven_programming)[, functional,](https://en.wikipedia.org/wiki/Functional_programming) and [imperativ](https://en.wikipedia.org/wiki/Imperative_programming)[e programming styles.](https://en.wikipedia.org/wiki/Programming_paradigm) It has [application programming interfaces (](https://en.wikipedia.org/wiki/Application_programming_interface)APIs) for working with text, dates, [regular expressions,](https://en.wikipedia.org/wiki/Regular_expression) standard [data structures,](https://en.wikipedia.org/wiki/Data_structure) and the [Document Object Model (](https://en.wikipedia.org/wiki/Document_Object_Model)DOM).

The ECMAScript standard does not include any [input/output (](https://en.wikipedia.org/wiki/Input/output)I/O), such as [networking](https://en.wikipedia.org/wiki/Computer_network)[, storage,](https://en.wikipedia.org/wiki/Data_storage) or [graphics f](https://en.wikipedia.org/wiki/Computer_graphics)acilities. In practice, the web browser or other runtime system provides JavaScript APIs for I/O. Although [Java a](https://en.wikipedia.org/wiki/Java_(programming_language))nd JavaScript are similar in name, [syntax,](https://en.wikipedia.org/wiki/Syntax_(programming_languages)) and respective [standard libraries,](https://en.wikipedia.org/wiki/Standard_library) the two languages are distinct and differ greatly in design.

[Brendan Eich l](https://en.wikipedia.org/wiki/Brendan_Eich)ater said of this period: "It's still kind of a [sidekick l](https://en.wikipedia.org/wiki/Sidekick)anguage. It's considered slow or annoying. People do [pop-ups o](https://en.wikipedia.org/wiki/Pop-up_ad)r those scrolling messages in the old [status bar a](https://en.wikipedia.org/wiki/Status_bar)t the bottom of your old [browser."](https://en.wikipedia.org/wiki/Web_browser)

In November 1996, [Netscape s](https://en.wikipedia.org/wiki/Netscape)ubmitted JavaScript to [Ecma International,](https://en.wikipedia.org/wiki/Ecma_International) as the starting point for a standard specification that all browser vendors could conform to. This led to the official release of the first [ECMAScript l](https://en.wikipedia.org/wiki/ECMAScript)anguage specification in June 1997. The standards process continued for a few years, with the release of ECMAScript 2 in June 1998 and ECMAScript 3 in December 1999. Work on ECMAScript 4 began in 2000.

However, the effort to fully standardize the language was undermined by [Microsoft](https://en.wikipedia.org/wiki/Microsoft) gaining an increasingly dominant position in the browser market. By the early 2000s, [Internet Explorer's](https://en.wikipedia.org/wiki/Internet_Explorer) market share reached 95%.[[22] T](https://en.wikipedia.org/wiki/JavaScript#cite_note-searchenginejournal.com-22)his meant that [JScript b](https://en.wikipedia.org/wiki/JScript)ecame the de facto standard for [client-side scripting o](https://en.wikipedia.org/wiki/Client-side_scripting)n the Web.

Microsoft initially participated in the standards process and implemented some proposals in its JScript language, but eventually it stopped collaborating on ECMA work.

Thus ECMAScript 4 was mothballed.

## Growth and standardization

During the period of [Internet Explorer d](https://en.wikipedia.org/wiki/Internet_Explorer)ominance in the early 2000s, client-side scripting was stagnant. This started to change in 2004, when the successor of Netscape, [Mozilla,](https://en.wikipedia.org/wiki/Mozilla) released the [Firefox b](https://en.wikipedia.org/wiki/Firefox)rowser. Firefox was well received by many, taking significant market share from Internet Explorer.

In 2005, Mozilla joined ECMA International, and work started on the [ECMAScript for XML (](https://en.wikipedia.org/wiki/ECMAScript_for_XML)E4X) standard. This led to Mozilla working jointly with [Macromedia (](https://en.wikipedia.org/wiki/Macromedia)later acquired by [Adobe Systems),](https://en.wikipedia.org/wiki/Adobe_Systems) who were implementing E4X in their ActionScript 3 language, which was based on an ECMAScript 4 draft. The goal became standardizing ActionScript 3 as the new ECMAScript 4. To this end, Adobe Systems released the [Tamarin](https://en.wikipedia.org/wiki/Tamarin_(software)) implementation as an [open source p](https://en.wikipedia.org/wiki/Open-source_model)roject. However, Tamarin and ActionScript 3 were too different from established client-side scripting, and without cooperation from [Microsoft,](https://en.wikipedia.org/wiki/Microsoft) ECMAScript 4 never reached fruition.

Meanwhile, very important developments were occurring in open-source communities not affiliated with ECMA work. In 2005, [Jesse James Garrett r](https://en.wikipedia.org/wiki/Jesse_James_Garrett)eleased a white paper in which he coined the term [Ajax a](https://en.wikipedia.org/wiki/Ajax_(programming))nd described a set of technologies, of which JavaScript was the backbone, to create [web applications w](https://en.wikipedia.org/wiki/Web_application)here data can be loaded in the background, avoiding the need for full page reloads. This sparked a renaissance period of JavaScript, spearheaded by open-source libraries and the communities that formed around them. Many new libraries were created, including [jQuer](https://en.wikipedia.org/wiki/JQuery)[y, Prototype](https://en.wikipedia.org/wiki/Prototype_JavaScript_Framework)[, Dojo Toolkit,](https://en.wikipedia.org/wiki/Dojo_Toolkit) and [MooTools.](https://en.wikipedia.org/wiki/MooTools)

[Google d](https://en.wikipedia.org/wiki/Google)ebuted its [Chrome b](https://en.wikipedia.org/wiki/Google_Chrome)rowser in 2008, with the [V8 J](https://en.wikipedia.org/wiki/V8_(JavaScript_engine))avaScript engine that was faster than its competition. The key innovation was [just-in-time compilation (](https://en.wikipedia.org/wiki/Just-in-time_compilation)JIT), so other browser vendors needed to overhaul their engines for JIT.

### Reaching maturity

Ambitious work on the language continued for several years, culminating in an extensive collection of additions and refinements being formalized with the publication of [ECMAScript 6](https://en.wikipedia.org/wiki/ECMAScript) in 2015.

The creation of [Node.js i](https://en.wikipedia.org/wiki/Node.js)n 2009 by [Ryan Dahl s](https://en.wikipedia.org/wiki/Ryan_Dahl)parked a significant increase in the usage of JavaScript outside of web browsers. Node combines the [V8 e](https://en.wikipedia.org/wiki/V8_(JavaScript_engine))ngine, an [event loop,](https://en.wikipedia.org/wiki/Event_loop) and [I/](https://en.wikipedia.org/wiki/Input/output)[O APIs,](https://en.wikipedia.org/wiki/Application_programming_interface) thereby providing a stand-alone JavaScript runtime system. As of 2018, Node had been used by millions of developers,[[31] a](https://en.wikipedia.org/wiki/JavaScript#cite_note-31)nd [npm h](https://en.wikipedia.org/wiki/Npm_(software))ad the most modules of any [package manager i](https://en.wikipedia.org/wiki/Package_manager)n the world.

The ECMAScript draft specification is currently maintained openly on [GitHub,](https://en.wikipedia.org/wiki/GitHub) and editions are produced via regular annual snapshots. Potential revisions to the language are

vetted through a comprehensive proposal process.[[](https://en.wikipedia.org/wiki/JavaScript#cite_note-34)[34][35] N](https://en.wikipedia.org/wiki/JavaScript#cite_note-35)ow, instead of edition numbers, developers check the status of upcoming features individually.

The current JavaScript ecosystem has many [libraries a](https://en.wikipedia.org/wiki/List_of_JavaScript_libraries)nd [frameworks,](https://en.wikipedia.org/wiki/Web_framework) established programming practices, and substantial usage of JavaScript outside of web browsers. Plus, with the rise of [single-p](https://en.wikipedia.org/wiki/Single-page_application)age application[s a](https://en.wikipedia.org/wiki/Single-page_application)nd other JavaScript-heavy websites, several [transpires h](https://en.wikipedia.org/wiki/Source-to-source_compiler)ave been created to aid the development process.

## Execution

### JavaScript engine

[A JavaScript engine i](https://en.wikipedia.org/wiki/JavaScript_engine)s a [software component t](https://en.wikipedia.org/wiki/Software_component)hat executes JavaScript [code.](https://en.wikipedia.org/wiki/Source_code) The first JavaScript [engines w](https://en.wikipedia.org/wiki/Software_engine)ere mere [interpreters,](https://en.wikipedia.org/wiki/Interpreter_(computing)) but all relevant modern engines use [just-in-time compilation f](https://en.wikipedia.org/wiki/Just-in-time_compilation)or improved performance.

JavaScript engines are typically developed by [web browser v](https://en.wikipedia.org/wiki/Web_browser)endors, and every major browser has one. In a browser, the JavaScript engine runs in concert with the [rendering engine v](https://en.wikipedia.org/wiki/Browser_engine)ia the [Document Object Model a](https://en.wikipedia.org/wiki/Document_Object_Model)nd [Web IDL b](https://en.wikipedia.org/wiki/Web_IDL)indings. However, the use of JavaScript engines is not limited to browsers; for example, the [V8 engine i](https://en.wikipedia.org/wiki/V8_(JavaScript_engine))s a core component of the [Node.j](https://en.wikipedia.org/wiki/Node.js)[s runtime system.](https://en.wikipedia.org/wiki/Runtime_system)

Since [ECMAScript i](https://en.wikipedia.org/wiki/ECMAScript)s the standardized specification of JavaScript, ECMAScript engine is another name for these [implementations.](https://en.wikipedia.org/wiki/Programming_language_implementation) With the advent of [Web Assembly,](https://en.wikipedia.org/wiki/WebAssembly) some engines can also execute this code in the same [sandbox a](https://en.wikipedia.org/wiki/Sandbox_(computer_security))s regular JavaScript code.

### Cascading Style Sheets

**Cascading Style Sheets** (**CSS**) is a [style sheet language u](https://en.wikipedia.org/wiki/Style_sheet_language)sed for specifying the [presentation a](https://en.wikipedia.org/wiki/Presentation_semantics)nd styling of a document written in a [m](https://en.wikipedia.org/wiki/Markup_language)arkup languag[e s](https://en.wikipedia.org/wiki/Markup_language)uch as [HTML o](https://en.wikipedia.org/wiki/HTML)[r XML (](https://en.wikipedia.org/wiki/XML)including XML dialects such as [SVG](https://en.wikipedia.org/wiki/SVG)[, MathML o](https://en.wikipedia.org/wiki/MathML)[r XHTML](https://en.wikipedia.org/wiki/XHTML)[).[1] C](https://en.wikipedia.org/wiki/CSS#cite_note-1)SS is a cornerstone technology of the [World Wide Web,](https://en.wikipedia.org/wiki/World_Wide_Web) alongside HTML and

[JavaScrip](https://en.wikipedia.org/wiki/JavaScript)[t.[2]](https://en.wikipedia.org/wiki/CSS#cite_note-2)

CSS is designed to enable the [separation of content and presentation,](https://en.wikipedia.org/wiki/Separation_of_content_and_presentation) including [layout](https://en.wikipedia.org/wiki/Page_layout)[, colors,](https://en.wikipedia.org/wiki/Color) and [fonts.](https://en.wikipedia.org/wiki/Typeface) This separation can improve content [accessibility,](https://en.wikipedia.org/wiki/Accessibility) since the content can be written without concern for its presentation; provide more flexibility and control in the specification of presentation characteristics; enable multiple [web pages t](https://en.wikipedia.org/wiki/Web_page)o share formatting by specifying the relevant CSS in a separate .css file, which reduces complexity and repetition in the structural content; and enable the .css file to be [cached t](https://en.wikipedia.org/wiki/Cache_(computing))o improve the page load speed between the pages that share the file and its formatting.

Separation of formatting and content also makes it feasible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (via speech-based browser or [screen reader),](https://en.wikipedia.org/wiki/Screen_reader) and on [Braille-based](https://en.wikipedia.org/wiki/Braille_display) tactile devices. CSS also has rules for alternate formatting if the content is accessed on a [mobile devic](https://en.wikipedia.org/wiki/Mobile_device)[e.[4]](https://en.wikipedia.org/wiki/CSS#cite_note-4)

The name *cascading* comes from the specified priority scheme to determine which declaration applies if more than one declaration of a property match a particular element. This cascading priority scheme is predictable.

The CSS specifications are maintained by the [World Wide Web Consortium (](https://en.wikipedia.org/wiki/World_Wide_Web_Consortium)W3C).

Internet media type ([MIME type)](https://en.wikipedia.org/wiki/MIME_media_type) text/css is registered for use with CSS by RFC 2318 (March 1998). The W3C operates a free [CSS validation service f](https://en.wikipedia.org/wiki/W3C_Markup_Validation_Service#CSS_validation)or CSS documents.[[5]](https://en.wikipedia.org/wiki/CSS#cite_note-5)

In addition to HTML, other markup languages support the use of CSS including [XHTML](https://en.wikipedia.org/wiki/XHTML)[, plain XML](https://en.wikipedia.org/wiki/Plain_Old_XML)[, SVG,](https://en.wikipedia.org/wiki/SVG) and [XUL.](https://en.wikipedia.org/wiki/XUL) CSS is also used in the [GT](https://en.wikipedia.org/wiki/GTK)[K widget toolkit.](https://en.wikipedia.org/wiki/Widget_toolkit)

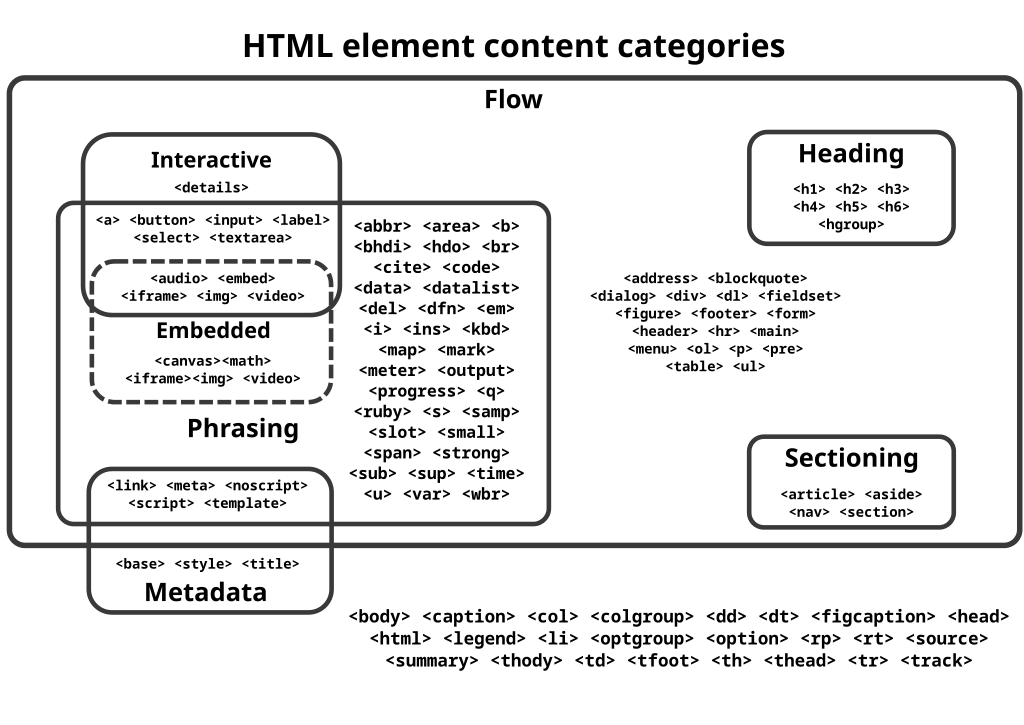
#### Hypertext Markup Language (HTML)

**Hypertext Markup Language** (**HTML**) is the standard [markup language f](https://en.wikipedia.org/wiki/Markup_language)or documents designed to be displayed in a [web browser.](https://en.wikipedia.org/wiki/Web_browser) It defines the content and structure of [web content.](https://en.wikipedia.org/wiki/Web_content) It is often assisted by technologies such as [Cascading Style Sheets (](https://en.wikipedia.org/wiki/Cascading_Style_Sheets)CSS) and [scripting languages s](https://en.wikipedia.org/wiki/Scripting_language)uch as [JavaScript.](https://en.wikipedia.org/wiki/JavaScript)

[Web browsers r](https://en.wikipedia.org/wiki/Web_browser)eceive HTML documents from a [web server o](https://en.wikipedia.org/wiki/Web_server)r from local storage and [render t](https://en.wikipedia.org/wiki/Browser_engine)he documents into multimedia web pages. HTML describes the structure of a [web pag](https://en.wikipedia.org/wiki/Web_page)[e semantically a](https://en.wikipedia.org/wiki/Semantic_Web)nd originally included cues for its appearance.

[HTML elements a](https://en.wikipedia.org/wiki/HTML_element)re the building blocks of HTML pages. With HTML constructs, [images a](https://en.wikipedia.org/wiki/HTML_element#Images_and_objects)nd other objects such as [interactive forms m](https://en.wikipedia.org/wiki/Fieldset)ay be embedded into the rendered page. HTML provides a means to create [structured documents b](https://en.wikipedia.org/wiki/Structured_document)y denoting structural [semantics f](https://en.wikipedia.org/wiki/Semantics)or text such as headings, paragraphs, lists, [links,](https://en.wikipedia.org/wiki/Hyperlink) quotes, and other items. HTML elements are delineated by *tags*, written using [angle brackets.](https://en.wikipedia.org/wiki/Bracket#Angle_brackets) Tags such as <**img**> and <**input**> directly introduce content into the page. Other tags such as <**p**> and </**p**> surround and provide information about document text and may include subelement tags. [Browsers d](https://en.wikipedia.org/wiki/Web_browser)o not display the HTML tags but use them to interpret the content of the page.

HTML can embed programs written in a [scripting language s](https://en.wikipedia.org/wiki/Scripting_language)uch as [JavaScript,](https://en.wikipedia.org/wiki/JavaScript) which affects the behavior and content of web pages. The inclusion of CSS defines the look and layout of content. The [World Wide Web Consortium (](https://en.wikipedia.org/wiki/World_Wide_Web_Consortium)W3C), former maintainer of the HTML and current maintainer of the CSS standards, has encouraged the use of [CSS o](https://en.wikipedia.org/wiki/CSS)ver explicit presentational HTML since 1997.[[2] A](https://en.wikipedia.org/wiki/HTML#cite_note-deprecated-2) form of HTML, known as [HTML5,](https://en.wikipedia.org/wiki/HTML5) is used to display video and audio, primarily using the <**canvas**> element, together with JavaScript.



HTML version timeline

HTML 2

November 24, 1995

HTML 3

January 14, 1997

HTML 4

December 18, 1997

HTML 5

October 28, 2014

November 1, 2016

HTML 5.1 was published as a W3C Recommendation.

December 14, 2017

HTML 5.2 was published as a W3C Recommendation.

Elements

HTML documents imply a structure of nested [HTML elements.](https://en.wikipedia.org/wiki/HTML_element) These are indicated in the document by HTML *tags*, enclosed in angle brackets thus: <**p**>.

In the simple, general case, the extent of an element is indicated by a pair of tags: a "start tag" <**p**> and "end tag" </**p**>. The text content of the element, if any, is placed between these tags.

Tags may also enclose further tag markup between the start and end, including a mixture of tags and text. This indicates further (nested) elements, as children of the parent element.

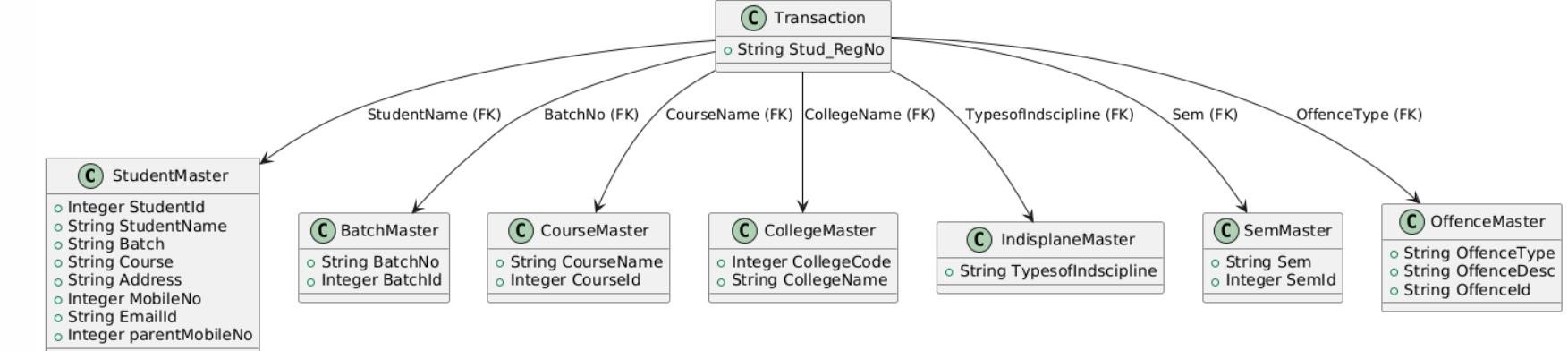
The start tag may also include the element's *attributes* within the tag. These indicate other information, such as identifiers for sections within the document, identifiers used to bind style information to the presentation of the document, and for some tags such as the <**img**> used to embed images, the reference to the image resource in the format like this: <**img** src="example.com/example.jpg">

Some elements, such as the [line break <](https://en.wikipedia.org/wiki/Line_breaking_character)**br** /> do not permit *any* embedded content, either text or further tags. These require only a single empty tag (akin to a start tag) and do not use an end tag.

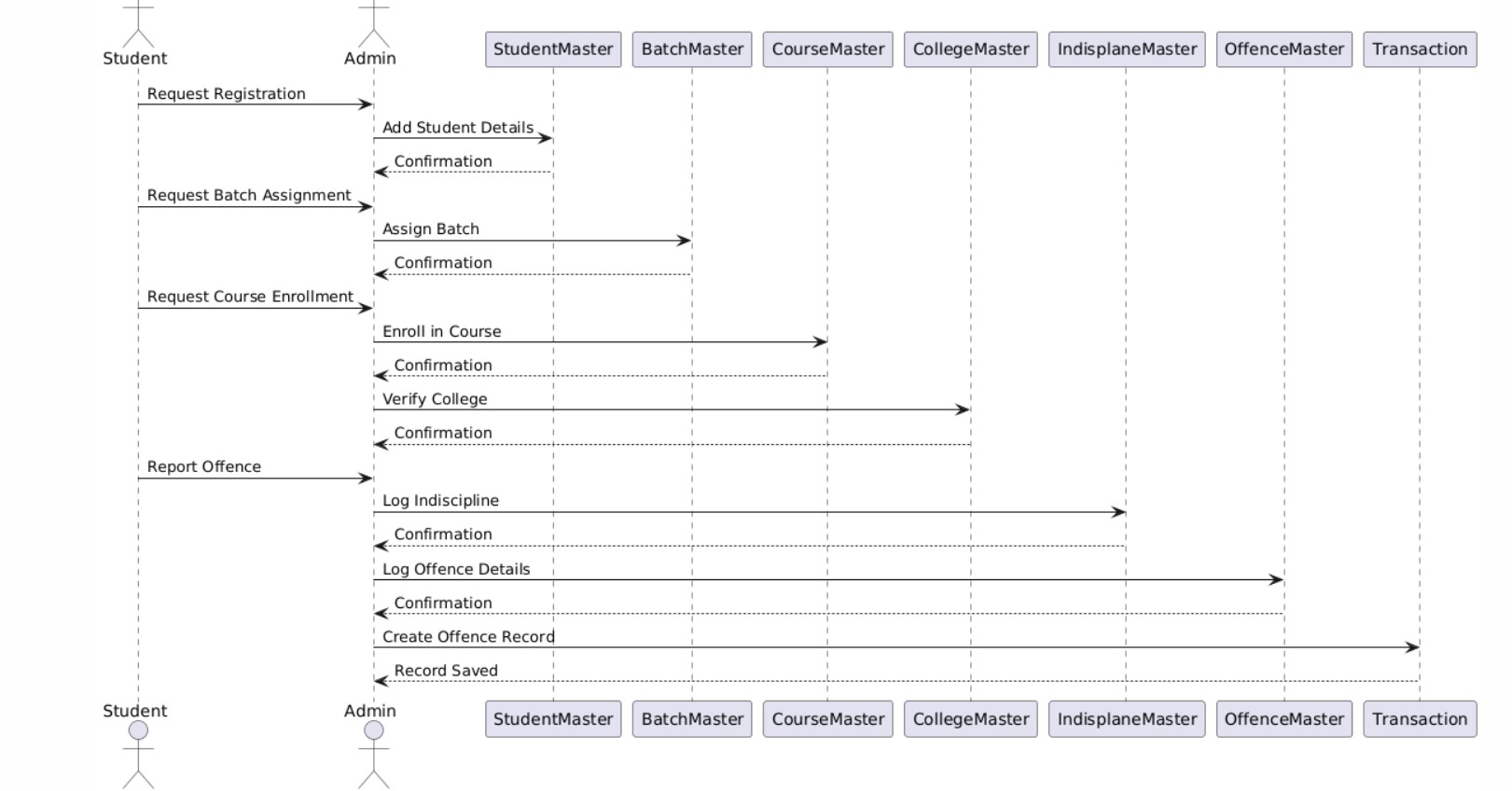
Many tags, particularly the closing end tag for the very commonly used paragraph element <**p**>, are optional. An HTML browser or other agent can infer the closure for the end of an element from the context and the structural rules defined by the HTML standard. These rules are complex and not widely understood by most HTML authors.

The general form of an HTML element is therefore: <**tag** attribute1="value1" attribute2="value2">''content''</**tag**>. Some HTML elements are defined as *empty elements* and take the form <**tag** attribute1="value1" attribute2="value2">. Empty elements may enclose no content, for instance, the <**br** /> tag or the inline <**img**> tag. The name of an HTML element is the name used in the tags. The end tag's name is preceded by a slash character, /, and that in empty elements the end tag is neither required nor allowed. If attributes are not mentioned, default values are used in each case.

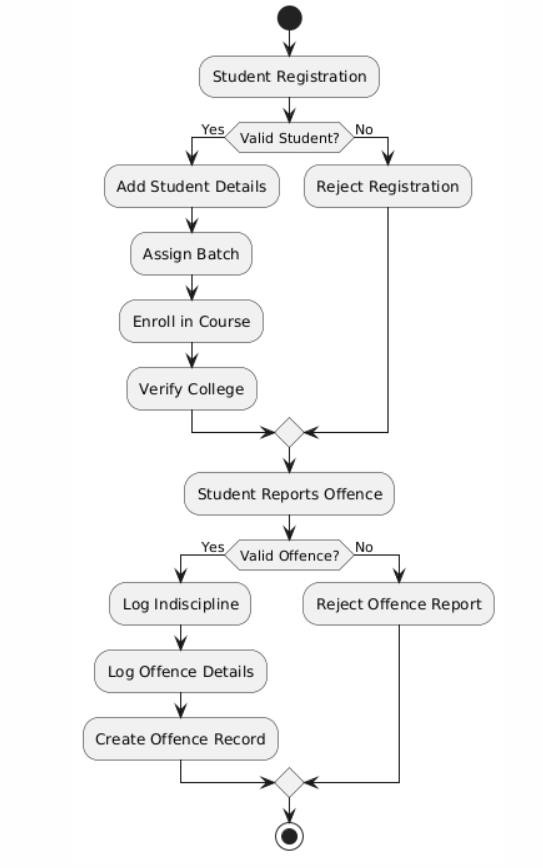
**8.UML Diagram’s:**



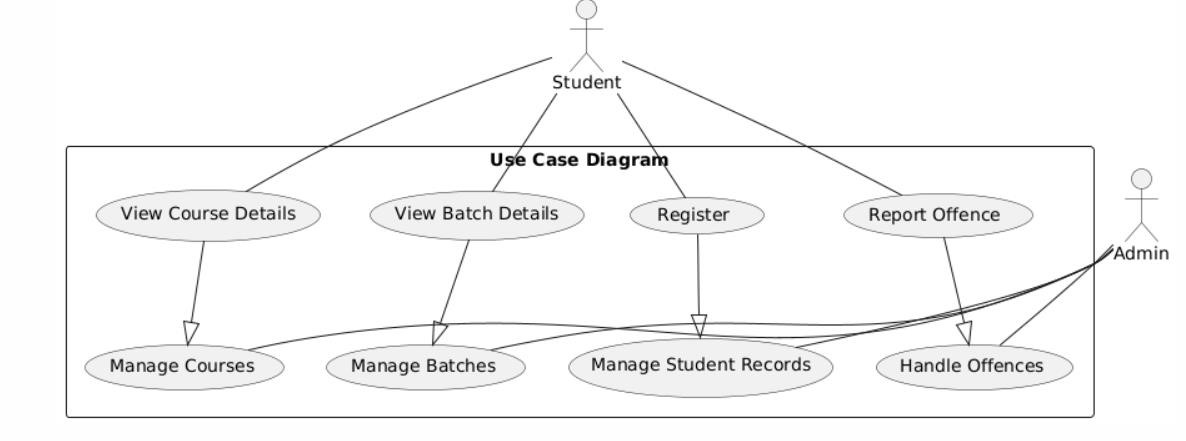
**Sequence Diagram:**



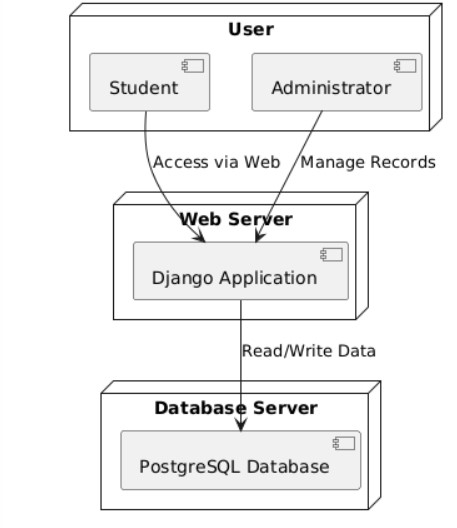
### Activity Diagram:-



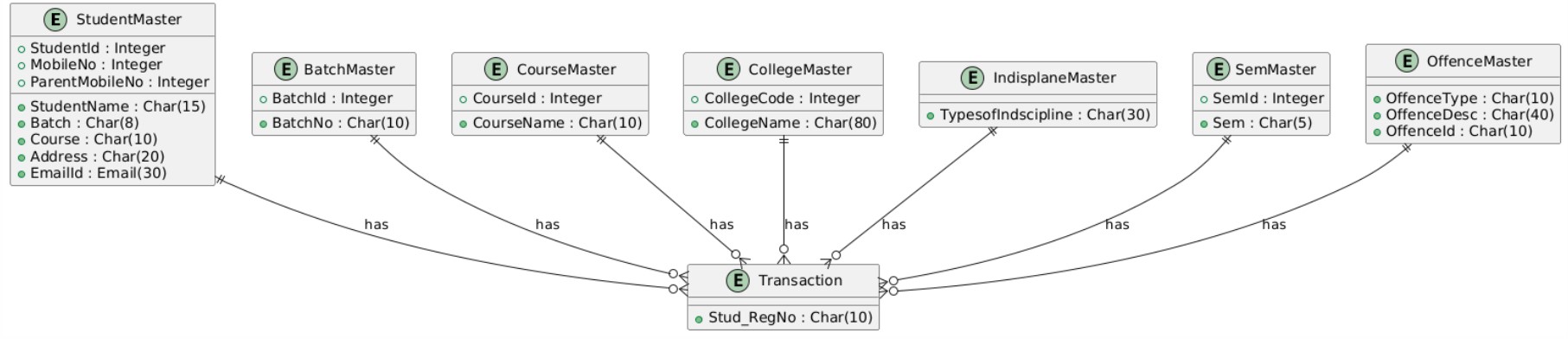
### Usecase Diagrams:-



**Deployement Diagram:**

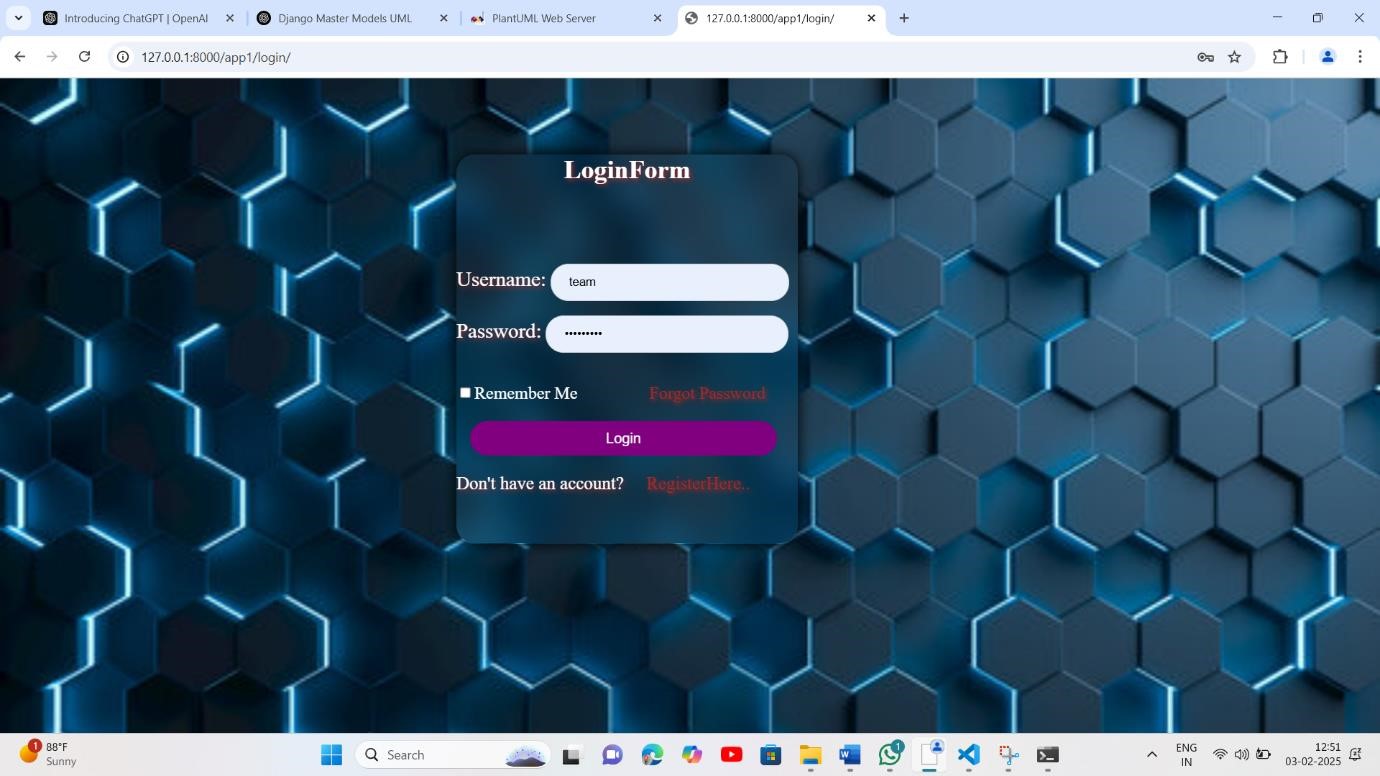
**SqlLite Database**

### Database Diagram:-

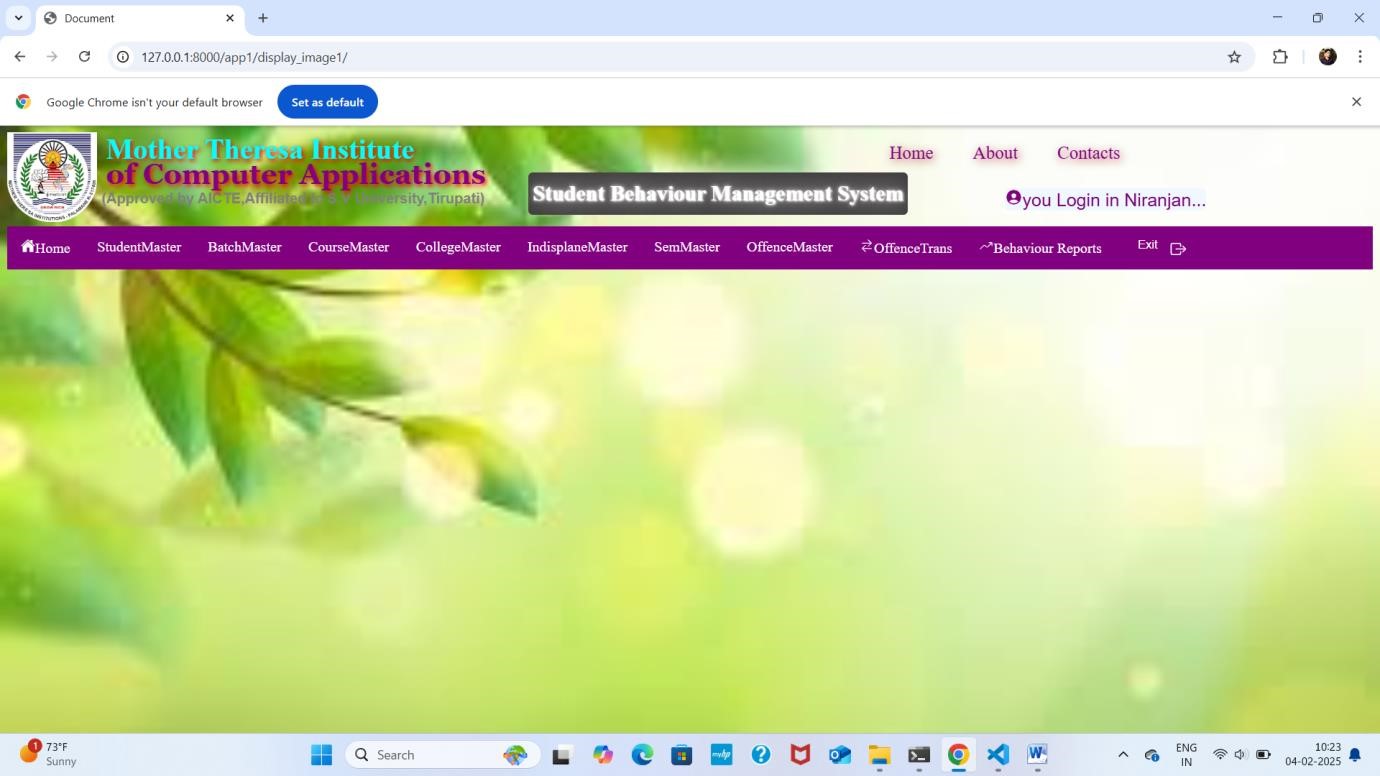


**9.Input/Output Screen:**

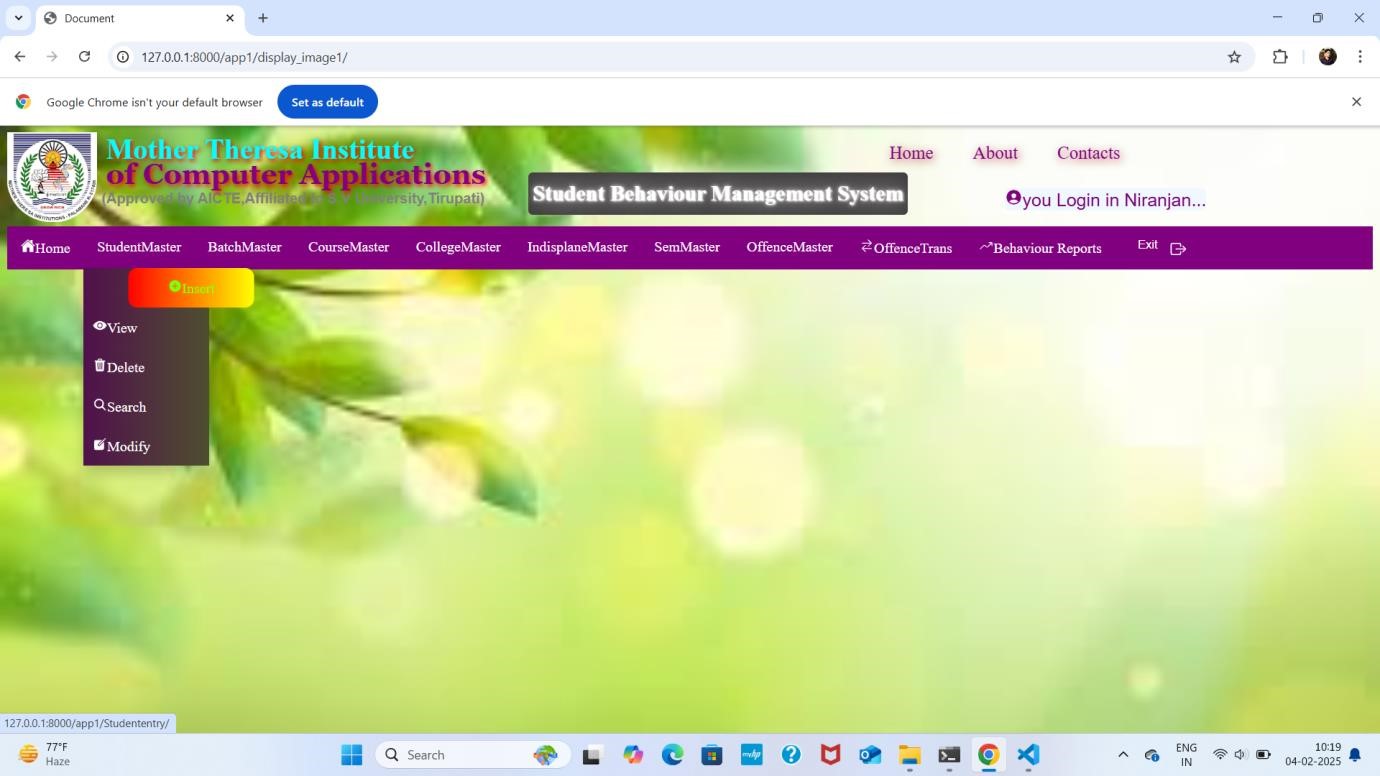
**Login Page:**



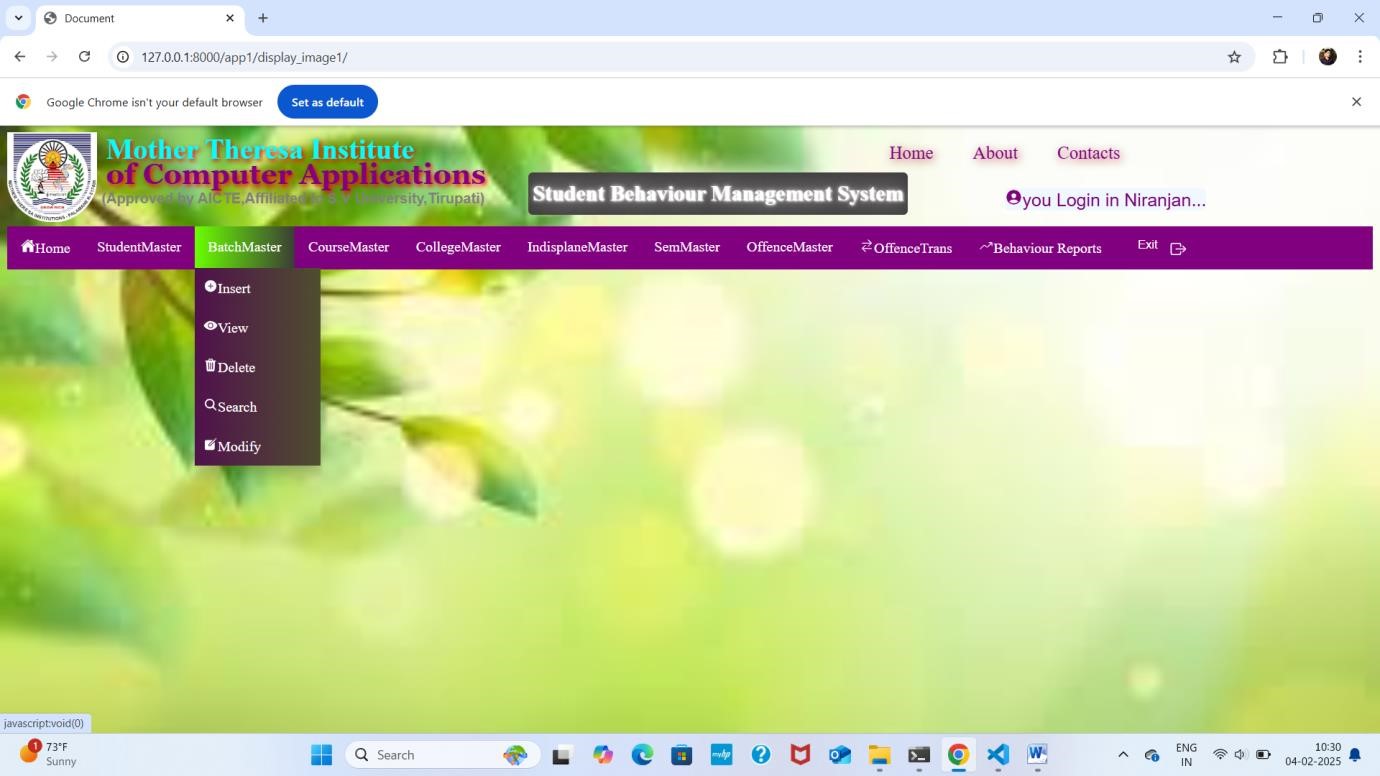
**Home Page:-**



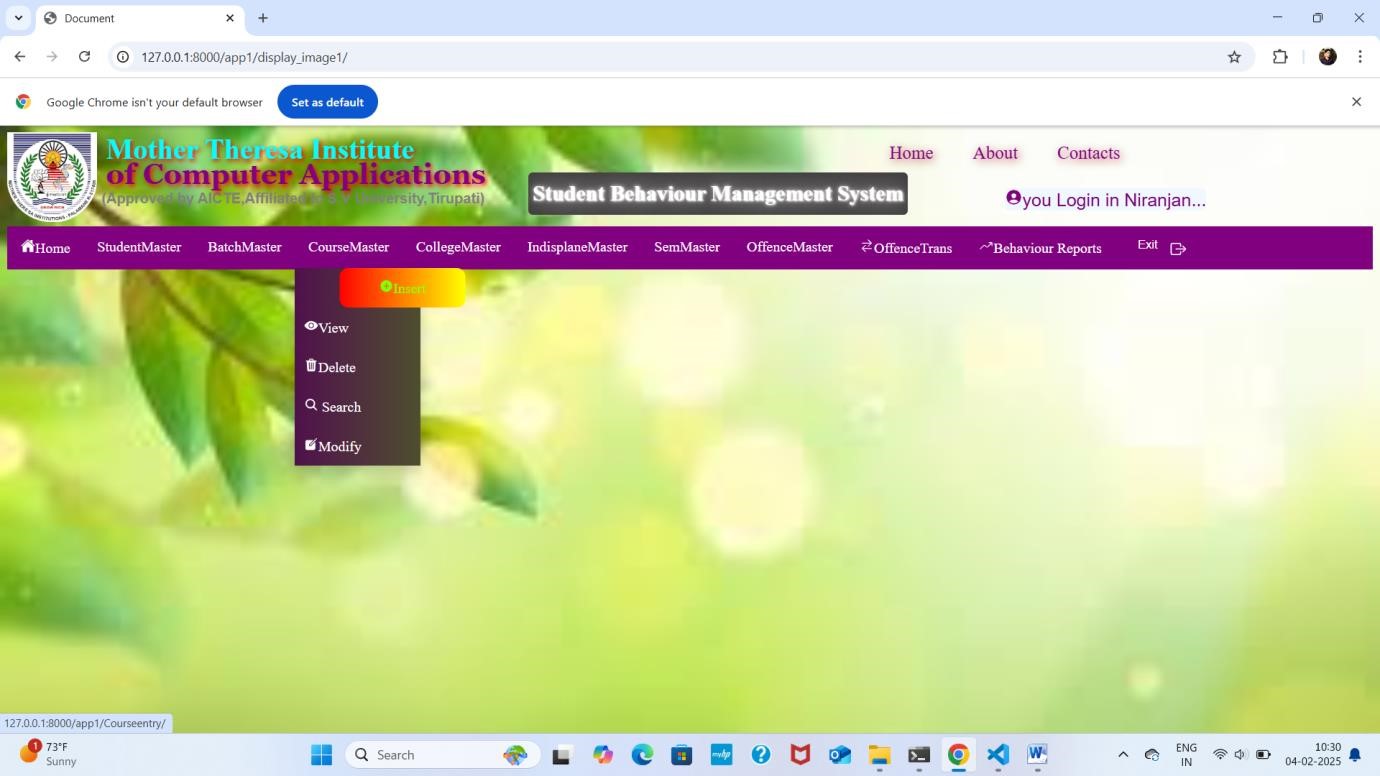
#### Student master:-



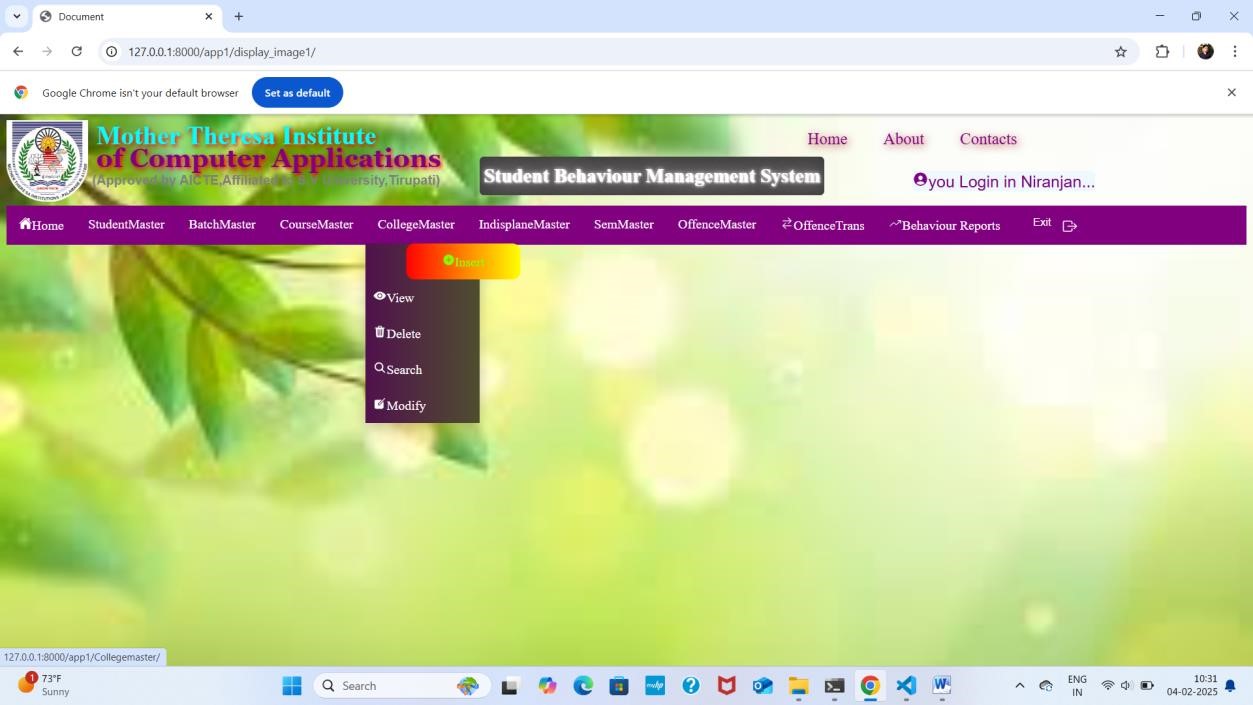
#### Batch Master:-



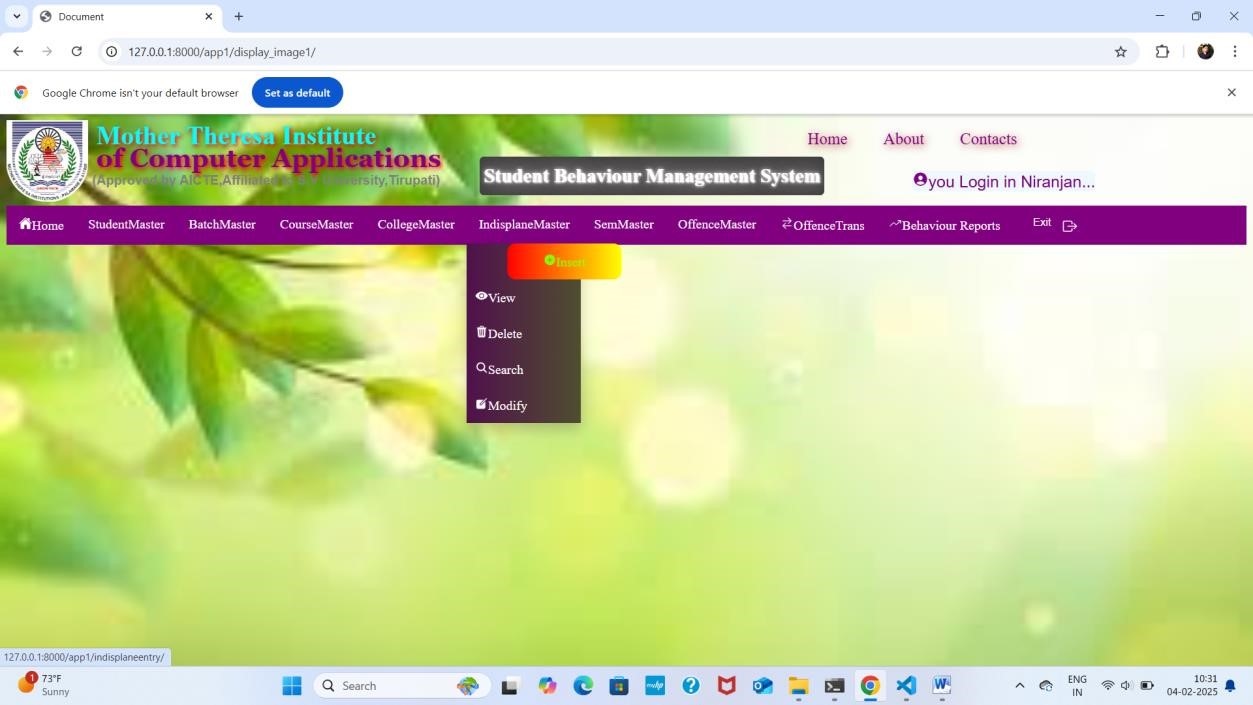
#### Course Master:-



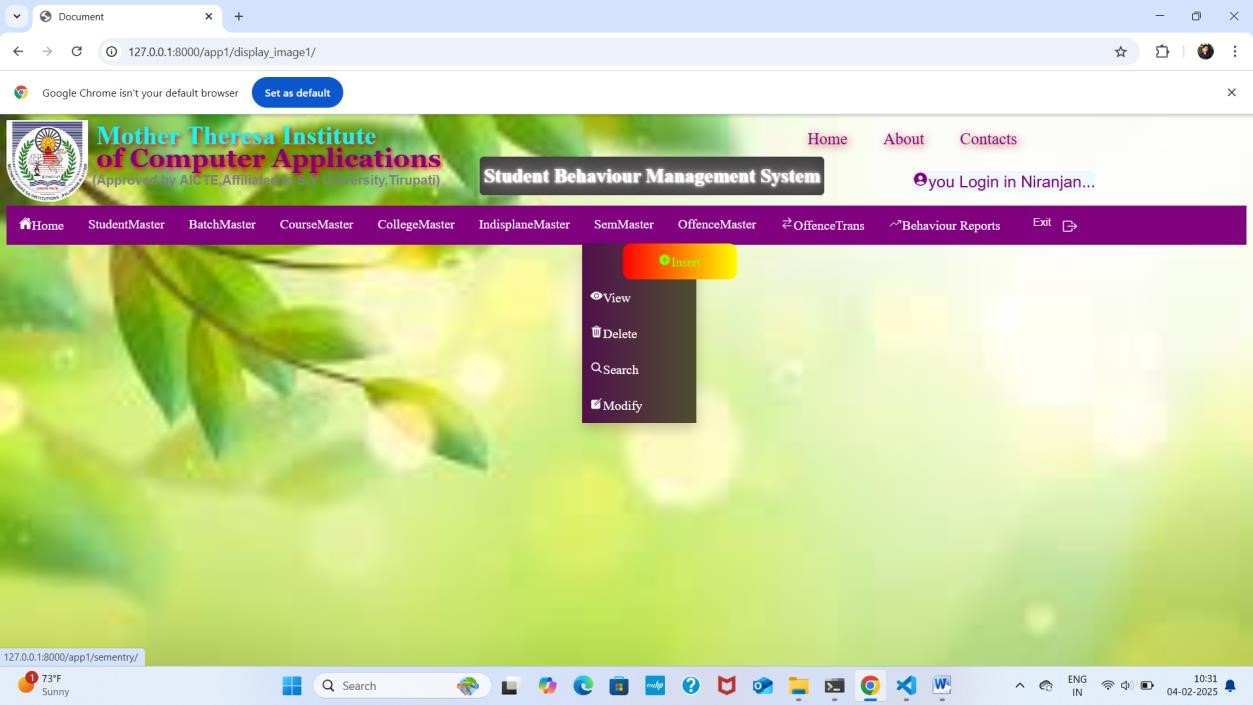
#### College Master:-



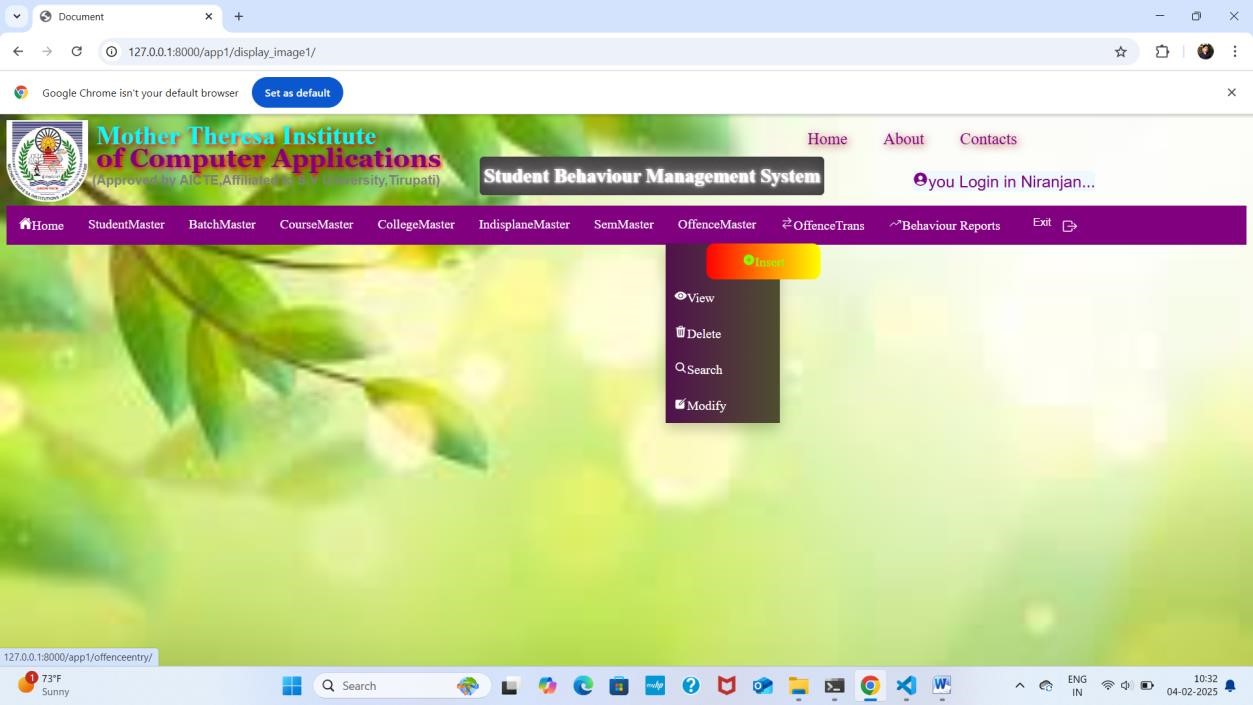
#### Indispline Master:-



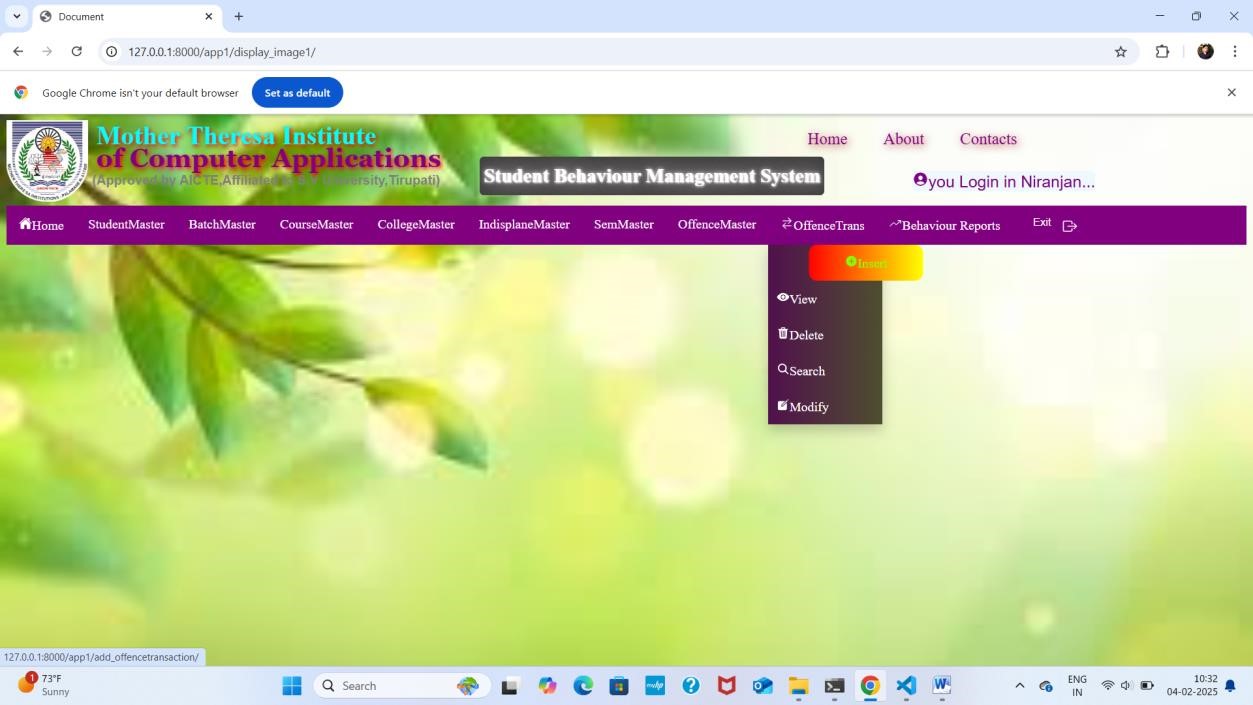
#### Sem Master:-



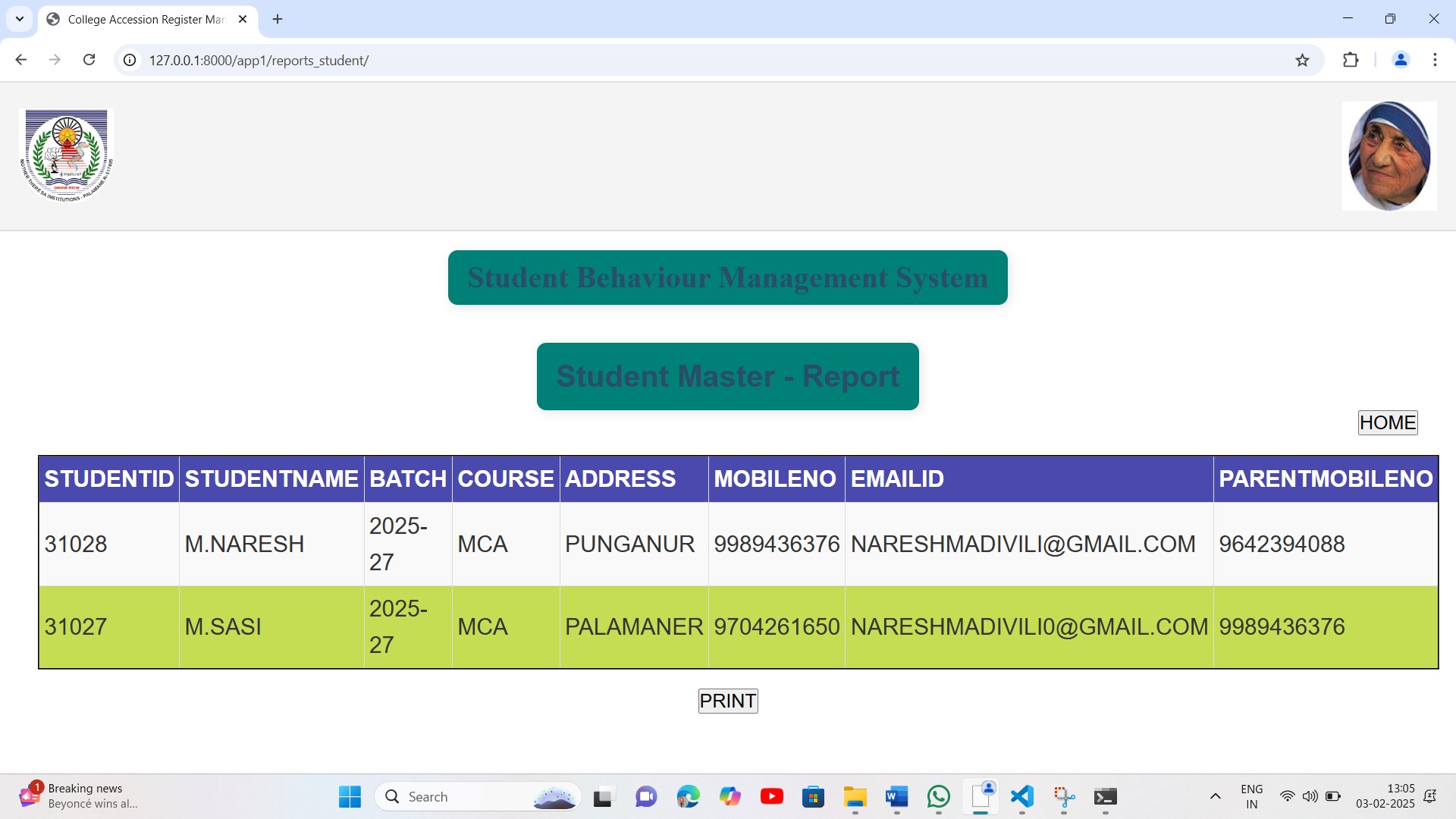
#### Offence Master:-



#### Transaction Master:-



#### Reports:-



### 10.Coding

#### Login Page Views.py

def login(request): if request.method == 'POST':

username = request.POST['username'] password = request.POST['password']

user = authenticate(request, username=username, password=password) if user is not None:

auth\_login(request, user) messages.success(request, f'Login successful! {username}') return redirect('display\_image1') else:

messages.error(request, f'Invalid username or password. Please try again.') return render(request, "login.html") **Urls.py**

path('login/',views.login, name='login'),

#### Login.html

{% load crispy\_forms\_tags %}

{% load static %} <style> body{

background: url("{% static 'img/back1.jpg' %}")no-repeat; background-size: cover;

} .form{ color:white; font-size:larger; margin-top:85px; margin-left:500px; height:433px; width: 380px; border: 1px solid black; backdrop-filter: blur(20px); border: none; border-radius: 20px; box-shadow: 0px 0px 15px 0px black;

} h2{ text-align: center; text-shadow: 2px 2px 5px brown; color: white; }

.inner{ margin-top:80px; margin-left:150px;

} .btn{ background-color:purple; /\* Green \*/ border: none; color: white; margin-top:20px; margin-left:15px; border-radius: 20px; width: 90%; padding: 10px 20px; text-align: center; text-decoration: none; display: inline-block; font-size: 16px;

}

.donot{ font-size: 20px; text-shadow: 2px 2px 5px brown;

} .donot a{ text-decoration: none; color: brown; text-shadow: 2px 2px 5px brown; margin-left: 20px; } input[type=text], select { width:70%; padding: 12px 20px; margin: 8px 0; display: inline-block; border: 1px solid #ccc; border-radius:20px; box-sizing: border-box; } input[type=password], select { width:71%; padding: 12px 20px; margin: 8px 0; display: inline-block; border: 1px solid #ccc; border-radius:20px; box-sizing: border-box; } label{ text-shadow: 2px 2px 5px brown; font-size:larger;

} input[type=checkbox]{ margin-top:30px;

} .forget{ text-decoration: none; color: brown; text-shadow: 2px 2px 5px brown; margin-left:80px;

} .p{ margin-left:350px; color: purple; font-size:larger; text-shadow: 2px 2px 10px #FF0000;

} .image{ height: 300px; width:300px; margin-left:250px; }

#message{

position: fixed; color: purple; margin-top: -420px; margin-left: 350px; text-shadow: 0px 0px 10px #FF0000; font-size: larger; } .person{ color: black;

position: fixed; margin-top: 18px; margin-left: -30px;

} .lock{ color: black; position: fixed; margin-top: 18px; margin-left: -30px;

}

</style>

{% if messages %}

<ul>

{% for message in messages %}

<h2 id="message">{{ message }}</h2>

{% endfor %}

</ul>

{% endif %}

<div class="form">

<h2>LoginForm</h2>

<form method="POST">

<div class="inner">

{% csrf\_token %}

</div>

<div>

<label for="username">Username:</label>

<input type="text" name="username" id="username" required placeholder="Username"><ion-icon name="person" class="person"></ion-icon>

</div>

<div>

<label for="password">Password:</label>

<input type="password" name="password" id="password" required placeholder="Password"><ion-icon name="lock-closed" class="lock"></ion-icon>

</div>

<input type="checkbox">Remember Me<a href="{% url 'forgot\_password' %}" class="forget">Forgot Password</a>

<button type="submit" class="btn" onclick="myFunction()">Login</button>

<script> function myFunction() { confirm("Login Successfully!!");

}

</script>

</form>

<p class="donot">Don't have an account? <a href="{% url 'register'

%}">RegisterHere..</a></p> <!-- Adjust registration link -->

</div>

#### Home Page:- Views.py

from django.shortcuts import render # Create your views here. def display\_image(request):

return render(request,"master.html") def display\_image1(request): return render(request,"master1.html") **Urls.py**

path('display\_image/',views.display\_image,name='display\_image'), path('display\_image1/',views.display\_image1,name='display\_image1'),

#### Home.html

{% load static %}

<html>

<head>

<title>Document</title>

<style> body{

background: url("{% static 'img/back1.jpg' %}")no-repeat; background-size: cover;

} p{ font-size: larger;

font-family: Georgia, 'Times New Roman', Times, serif;

} img{ height: 100px; width: 100px;

} .para{ position: fixed; text-align:left; margin-top:-100px; margin-left: 110px; color: aqua; text-shadow: 2px 2px 5px red;

} .para1{ position: fixed; margin-top: -60px; margin-left: 580px; color:white; background-color:rgb(62, 60, 60); padding: 10px 5px;

text-shadow: 1px 1px 2px white, 0 0 25px white, 0 0 5px white; border-radius: 5px;

} .text1{ color: purple; margin-top: -10px; font-size:xx-large;

} .text{

font-family:Arial, Helvetica, sans-serifs; color: purple; text-align:right; margin-left: 1110px; margin-top: -90px; font-size: 20px; background-color: aliceblue; width: fit-content; border-radius: 5px;

} ul { list-style-type: none; margin: 0; padding: 0; overflow: hidden; background-color:purple;

} .drop{ position: fixed; color:white; margin-top:3px; padding: 10px 20px; border:none; background-color: purple; border-radius:10px;

} .drop1{ position: fixed; color:white; margin-left:5px; margin-top:3px; padding: 10px 20px; border:none; background-color: purple; border-radius:10px; }

.drop1:hover{ background-color:rgb(16, 199, 68); color:purple;

box-shadow: 0px 0px 50px 0px rgb(16,199,68);

} .drop:hover{ background-color:rgb(16, 199, 68); color:purple;

box-shadow: 0px 0px 50px 0px rgb(16,199,68);

} li { float:left;

}

li a, .dropbtn { display: inline-block; color: white; text-align: center; padding: 14px 15px; text-decoration: none; z-index: var(--i); transition: 0.5s;

}

li a:hover, .dropdown:hover.dropbtn { background-image: linear-gradient(to right, rgb(106, 255, 0) , rgb(59, 62, 54,1)); }

li.dropdown { display: inline-block;

}

.dropdown-content { display: none; position: absolute;

background-image: linear-gradient(to right, rgb(80, 16, 76) , rgb(78, 78, 47)); min-width: 140px;

box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2); z-index: 1;

}

.dropdown-content a { color:black; padding: 12px 10px; text-decoration: none; display: block; text-align: left;

}

.dropdown-content a:hover {background-image: linear-gradient(to right, red , yellow); transform: translateX(50px); border-radius: 10px; text-align: center; color:chartreuse;

}

.dropdown:hover .dropdown-content { display: block;

} header{ position: fixed; top: 0; left: 0;

width: 100%; padding: 20px 950px; display: flex;

justify-content: space-between; align-items:center; z-index:99;

}

.navigation a{ position: relative; font-size:20px; color:purple;

text-shadow: 2px 2px 10px brown; text-decoration: none; font-weight: 500; margin-left: 40px;

}

.navigation a::after{ content: ''; position: absolute; left: 0;

bottom: -6px; width:100%; height:2px; background:brown; border-radius: 5px; transform: scaleX(0); transition: transform 0.5s; } .navigation a:hover:after{ transform: scaleX(1); } .print{ position: fixed; margin-top: 14px; margin-left:0px; }

.exit{ font-size:20px; color:white; margin-left: 60px; margin-top: 15px; } .img-container{ height: 100px; width: 100px; }

.Affiliation{ font-family:sans-serif; color: gray; margin-top: -35px; margin-left: 105px;

}

</style>

</head>

<body>

<header>

<nav class="navigation">

<a href="#Home">Home</a>

<a href="#About">About</a>

<a href="#Contacts">Contacts</a>

</nav>

</header>

<img src="{% static 'img/Logo2.png' %}" class="img-container">

<h1 class="para">Mother Theresa Institute <br><p class="text1">of Computer

Applications</p></h1>

<h4 class="Affiliation">(Approved by AICTE,Affiliated to S.V University,Tirupati)</h4>

<h2 class="para1">Student Behaviour Management System</h2>

<div class="nav">

<ul>

<li class="dropdown">

<a href="{% url 'display\_image1' %}" class="dropbtn"><ion-icon name="home"></ionicon>Home</a>

<div class="dropdown-content">

</div>

</li>

<li class="dropdown">

<a href="javascript:void(0)" class="dropbtn">StudentMaster</a>

<div class="dropdown-content">

<a href="{% url 'Studententry' %}"><ion-icon name="add-circle"></ion-icon>Insert</a>

<a href="{% url 'displaystudent' %}"><ion-icon name="eye"></ion-icon>View</a>

<a href="{% url 'delete\_Student' %}"><ion-icon name="trash"></ion-icon>Delete</a>

<a href="{% url 'search\_student' %}"><ion-icon name="search"></ion-icon>Search</a>

<a href="{% url 'student\_all\_details' %}"><ion-icon name="create"></ion-icon>Modify</a>

</div>

</li>

<li class="dropdown">

<a href="javascript:void(0)" class="dropbtn">BatchMaster</a>

<div class="dropdown-content">

<a href="{% url 'Batchentry' %}"><ion-icon name="add-circle"></ion-icon>Insert</a>

<a href="{% url 'displaybatch' %}"><ion-icon name="eye"></ion-icon>View</a>

<a href="{% url 'delete\_batch' %}"><ion-icon name="trash"></ion-icon>Delete</a>

<a href="{% url 'search\_batch' %}"><ion-icon name="search"></ion-icon>Search</a>

<a href="{% url 'batch\_all\_details' %}"><ion-icon name="create"></ion-icon>Modify</a>

</div>

</li>

<li class="dropdown">

<a href="javascript:void(0)" class="dropbtn">CourseMaster</a>

<div class="dropdown-content">

<a href="{% url 'Courseentry' %}"><ion-icon name="add-circle"></ion-icon>Insert</a>

<a href="{% url 'displaycourse' %}"><ion-icon name="eye"></ion-icon>View</a>

<a href="{% url 'delete\_course' %}"><ion-icon name="trash"></ion-icon>Delete</a>

<a href="{% url 'search\_course' %}"><ion-icon name="search"></ion-icon> Search</a>

<a href="{% url 'course\_all\_details' %}"><ion-icon name="create"></ion-icon>Modify</a>

</div>

</li>

<li class="dropdown">

<a href="javascript:void(0)" class="dropbtn">CollegeMaster</a>

<div class="dropdown-content">

<a href="{% url 'Collegemaster' %}"><ion-icon name="add-circle"></ion-icon>Insert</a>

<a href="{% url 'displaycollege' %}"><ion-icon name="eye"></ion-icon>View</a>

<a href="{% url 'delete\_college' %}"><ion-icon name="trash"></ion-icon>Delete</a>

<a href="{% url 'search\_college' %}"><ion-icon name="search"></ion-icon>Search</a>

<a href="{% url 'college\_all\_details' %}"><ion-icon name="create"></ion-icon>Modify</a>

</div>

</li>

<li class="dropdown">

<a href="javascript:void(0)" class="dropbtn">IndisplaneMaster</a>

<div class="dropdown-content">

<a href="{% url 'indisplaneentry' %}"><ion-icon name="add-circle"></ion-icon>Insert</a>

<a href="{% url 'displayIndisplane' %}"><ion-icon name="eye"></ion-icon>View</a>

<a href="{% url 'delete\_Indisplane' %}"><ion-icon name="trash"></ion-icon>Delete</a>

<a href="{% url 'search\_Indisplane' %}"><ion-icon name="search"></ion-icon>Search</a>

<a href="{% url 'indisplane\_all\_details' %}"><ion-icon name="create"></ionicon>Modify</a>

</div>

</li>

<li class="dropdown">

<a href="javascript:void(0)" class="dropbtn">SemMaster</a>

<div class="dropdown-content">

<a href="{% url 'sementry' %}"><ion-icon name="add-circle"></ion-icon>Insert</a>

<a href="{% url 'displaySem' %}"><ion-icon name="eye"></ion-icon>View</a>

<a href="{% url 'delete\_sem' %}"><ion-icon name="trash"></ion-icon>Delete</a>

<a href="{% url 'search\_sem' %}"><ion-icon name="search"></ion-icon>Search</a>

<a href="{% url 'sem\_all\_details' %}"><ion-icon name="create"></ion-icon>Modify</a>

</div>

</li>

<li class="dropdown">

<a href="javascript:void(0)" class="dropbtn">OffenceMaster</a>

<div class="dropdown-content">

<a href="{% url 'offenceentry' %}"><ion-icon name="add-circle"></ion-icon>Insert</a>

<a href="{% url 'displayoffence' %}"><ion-icon name="eye"></ion-icon>View</a>

<a href="{% url 'delete\_offence' %}"><ion-icon name="trash"></ion-icon>Delete</a>

<a href="{% url 'search\_offence' %}"><ion-icon name="search"></ion-icon>Search</a>

<a href="{% url 'offence\_all\_details' %}"><ion-icon name="create"></ion-icon>Modify</a>

</div>

</li>

<li class="dropdown">

<a href="javascript:void(0)" class="dropbtn"><ion-icon name="swap-horizontal"></ionicon>OffenceTrans</a>

<div class="dropdown-content">

<a href="{% url 'add\_offencetransaction' %}"><ion-icon name="add-circle"></ionicon>Insert</a>

<a href="{% url 'offencetransaction\_list' %}"><ion-icon name="eye"></ion-icon>View</a>

<a href="{% url 'delete\_Transaction' %}"><ion-icon name="trash"></ion-icon>Delete</a>

<a href="#search"><ion-icon name="search"></ion-icon>Search</a>

<a href="#Modify"><ion-icon name="create"></ion-icon>Modify</a>

</div>

</li>

<li class="dropdown">

<a href="javascript:void(0)" class="dropbtn"><ion-icon name="trending-up-outline"></ionicon>Behaviour Reports</a>

<div class="dropdown-content">

<a href="{% url 'reports\_student' %}">Student Master</a>

<a href="{% url 'reports\_batch' %}">Batch Master</a>

<a href="{% url 'reports\_course' %}">Course Master</a>

<a href="{% url 'reports\_college' %}">College master</a>

<a href="{% url 'reports\_indisplane' %}">Indisplane Master</a>

<a href="{% url 'reports\_sem' %}">Sem Master</a>

<a href="{% url 'reports\_offence' %}">Offence Master</a>

<a href="#Modify">OffenceTrans Master</a>

</div>

</li>

<a href="{% url 'logout\_view' %}"><input type="submit" value="Exit" class="drop1"><ionicon name="exit-outline" class="exit"></ion-icon></a>

</ul>

</div>

<p class="text"><ion-icon name="person-circle"></ion-icon>you Login in

{{user.username}}...</p>

<script type="module" src="https://unpkg.com/ionicons@7.1.0/dist/ionicons/ionicons.esm.js"></script>

{% block content%}

{% endblock %}

</body>

</html>

### Student Master

#### Models.py

class StudentMaster(models.Model):

StudentId = models.IntegerField()

StudentName = models.CharField(max\_length=15) Batch = models.CharField(max\_length=8) course = models.CharField(max\_length=10)

Address = models.CharField(max\_length=20)

MobileNo = models.IntegerField()

EmailId = models.EmailField(max\_length=30) parentMobileNo = models.IntegerField() def \_\_str\_\_(self):

return self.StudentName

**Forms.py** from . models import StudentMaster class StudentForm(forms.ModelForm):

class Meta:

model = StudentMasterfields = "\_\_all\_\_"

#### Views.py

def Studententry(request):

context = {}

context['form'] = StudentForm()

return render(request,"Studententry.html",context) def process\_student(request): if request.method == 'POST':

StudentId\_inp = request.POST.get('StudentId')

StudentName\_inp = request.POST.get('StudentName') Batch\_inp = request.POST.get('Batch') course\_inp = request.POST.get('course')

Address\_inp = request.POST.get('Address') MobileNo\_inp = request.POST.get('MobileNo') EmailId\_inp = request.POST.get('EmailId')

parentMobileNo\_inp = request.POST.get('parentMobileNo')

ob = StudentMaster(StudentId = StudentId\_inp,StudentName = StudentName\_inp,Batch =

Batch\_inp,course =course\_inp,Address

=Address\_inp,MobileNo=MobileNo\_inp,EmailId=EmailId\_inp,parentMobileNo=parentMobi leNo\_inp) ob.save()

return render(request,"return1.html") else:

return HttpResponse("Inavlid request method")

def displaystudent(request):

student = StudentMaster.objects.all().values() temp = loader.get\_template('displaystudent.html') context = {

'data':student,

}

return HttpResponse(temp.render(context,request))

#### StudentEntry.html

{% extends "master1.html" %} {% block content %}

{% load crispy\_forms\_tags %}

{% load static %}

<style> .form{ color: purple; font-size:larger; margin-top:72px; margin-left:450px; height:420px; width: 380px; border: 1px solid black; backdrop-filter: blur(20px); border: none; border-radius: 20px; box-shadow: 0px 0px 15px 0px black;

} .inner{ position: fixed; margin-top:-20px; margin-left:120px; } p{ text-align: center; justify-content: center;

} .btn{ background-color:purple; /\* Green \*/ border: none;

color: white; margin-top:30px; margin-left:-50px; border-radius: 20px; width: 150%; padding: 10px 2px; text-align: center; text-decoration: none; font-size: 16px;

}

</style>

<form method="post" action="{% url 'process\_student' %}">

<div class="form">

<p>StudentMaster</p>

<div class="inner">

{% csrf\_token %}

{{form|crispy}}

<input type="submit" value="SUBMIT" class="btn">

</div>

</div>

</form> {% endblock %} **DisplayStudent.html**

{% extends "master1.html" %}

{% block content %}

<style> .table {

font-family: Arial, Helvetica, sans-serif; border-collapse: collapse; width: 100%;

}

.table td, .table th { border: 1px solid #ddd; padding: 8px;

}

.table tr:nth-child(even){background-color: #f2f2f2;}

.table tr:hover {background-color: #ddd;}

.table th { padding-top: 12px; padding-bottom: 12px; text-align: left; background-color:rgb(29, 29, 65); color: white; } .table-container{ margin-top: 90px;

} .head{ padding-top: 20px; padding-bottom: 20px; padding-left: 50px; text-align: left; background-color:rgb(29, 29, 65); color: white; } .btn1{ color: white; background: green; text-decoration: none; padding: 10px 15px; margin-left: 980px;

border-radius: 5px;

} .btn2{ color: white; margin-left: -320px; background:red; padding: 10px 15px; border-radius: 5px; text-decoration: none;

}

</style>

<div class="content">

<div class="table-container">

<table border='' style='border-collapse:collapse;' class="table">

<p class="head">Student View

<a href="{% url 'Studententry' %}" class="btn1"><ion-icon name="add-circle"></ionicon>Add New Student</a>

<a href="{% url 'delete\_Student' %}" class="btn2"><ion-icon name="trash"></ionicon>Delete</a>

</p>

<tr><th>StudentId</th>

<th>StudentName</th><th>Batch</th><th>course</th><th>Address</th><th>MobileNo</t h><th>EmailId</th><th>parentMobileNo</th></tr>

{% for x in data %}

<tr>

<td>{{x.StudentId}}</td>

<td>{{x.StudentName}}</td>

<td>{{x.Batch}}</td>

<td>{{x.course}}</td>

<td>{{x.Address}}</td>

<td>{{x.MobileNo}}</td>

<td>{{x.EmailId}}</td>

<td>{{x.parentMobileNo}}</td>

</tr>

{% endfor %}

</table>

</div>

</div>

{% endblock %}

### Delete Student Masters

#### Views.py

def delete1(request,pk):

StudentMaster.objects.filter(id=pk).delete() return render(request,"retrieve2.html") def delete\_Student(request): allstudent=StudentMaster.objects.all().values() temp=loader.get\_template('delete\_student.html') context={

'data':allstudent,

}

return HttpResponse(temp.render(context,request)) **Urls.py**

path('delete1/<pk>',views.delete1,name='delete1'), path('delete\_Student/',views.delete\_Student,name='delete\_Student'), **delete\_student.html**

{% extends "master1.html" %}

{% block content %}

<style> .table {

font-family: Arial, Helvetica, sans-serif; border-collapse: collapse; width: 100%;

}

.table td, .table th { border: 1px solid #ddd; padding: 8px;

}

.table tr:nth-child(even){background-color: #f2f2f2;}

.table tr:hover {background-color: #ddd;}

.table th { padding-top: 12px; padding-bottom: 12px; text-align: left; background-color:rgb(29, 29, 65); color: white; } .table-container{ margin-top: 90px;

} .head{ padding-top: 20px; padding-bottom: 20px; padding-left: 50px; text-align: left; background-color:rgb(29, 29, 65); color: white; } .btn1{ color: white; background: green; text-decoration: none; padding: 10px 15px;

margin-left: 950px; border-radius: 5px;

} .btn2{ color: white; margin-left: -320px; background:red; padding: 10px 15px; border-radius: 5px; text-decoration: none;

} .btn{

background:rgb(16, 84, 193); border: none; padding: 5px 10px; border-radius: 3px;

}

</style>

<div class="content">

<div class="table-container">

<table border='' style='border-collapse:collapse;' class="table">

<p class="head">Student Delete

<a href="{% url 'Studententry' %}" class="btn1"><ion-icon name="add-circle"></ionicon>Add New Student</a>

</p>

<tr> <th>StudentId</th>

<th>StudentName</th><th>Batch</th><th>course</th><th>Address</th><th>MobileNo</t h><th>EmailId</th><th>parentMobileNo</th><th>Delete</th></tr>

{% for x in data %}

<tr>

<td>{{x.StudentId}}</td>

<td>{{x.StudentName}}</td>

<td>{{x.Batch}}</td>

<td>{{x.course}}</td>

<td>{{x.Address}}</td>

<td>{{x.MobileNo}}</td>

<td>{{x.EmailId}}</td>

<td>{{x.parentMobileNo}}</td>

<td><a href="{% url 'delete1' pk=x.id %}"><button type="button" style="margin-left:5px;" class="btn" onclick="myFunction()">delete</button></a></td>

</tr>

{% endfor %}

</table> <script>

function myFunction() { alert("Are you sure");

}

</script>

{% endblock %}

### Search Student Master

#### Views.py

def search\_student(request): if request.method == 'GET':

form =SearchForm(request.GET)

if form.is\_valid(): query = form.cleaned\_data.get('query') results = StudentMaster.objects.filter(StudentId=query) else: results = None else:

form = SearchForm() results = None

return render(request, 'search\_student.html', {'form': form, 'results': results}) **search\_student.html**

{% extends "master1.html" %}

{% block content %}

<style> .table {

font-family: Arial, Helvetica, sans-serif; border-collapse: collapse; width: 100%;

}

.table td, .table th { border: 1px solid #ddd; padding: 8px;

}

.table tr:nth-child(even){background-color: #f2f2f2;}

.table tr:hover {background-color: #ddd;}

.table th { padding-top: 12px; padding-bottom: 12px; text-align: left;

background-color:rgb(29, 29, 65); color: white;

}

.table-container{ margin-top: 90px;

} .head{ padding-top: 20px; padding-bottom: 20px; padding-left: 50px; text-align: left; background-color:rgb(29, 29, 65); color: white; margin-top: 100px;

} .btn2{ color: white; margin-left: 1020px; background:red; padding: 10px 15px; border-radius: 5px; text-decoration: none;

} .search{ margin-top: -70px; margin-left:900px; position: absolute; }

.searchicon{ margin-top: -70px; margin-left:1100px; background:red; padding: 10px 15px; border-radius: 5px; border: none; text-decoration: none; color: white; } input{ margin-top: -40px; position: fixed; padding: 10px 15px; border-radius: 5px; position: absolute;

} .arrange{ margin-top: -70px; position: absolute;

}

</style>

<table border=1 class="table">

<p class="head">Student Search

<a href="{% url 'display\_image1' %}" class="btn2"><ion-icon name="home-outline"></ionicon> Home</a>

<div class="search">

<form method="get" class="arrange">

{{ form.as\_p }}

</div>

<button type="submit" class="searchicon"><ion-icon name="search-outline"></ion-icon> Search</button>

{% if results %}

</form>

</div>

</p>

<tr> <th>StudentId</th>

<th>StudentName</th><th>Batch</th><th>course</th><th>Address</th><th>MobileNo</t h><th>EmailId</th><th>parentMobileNo</th></tr>

{% for item in results %}

<tr>

<td>{{ item.StudentId }}</td>

<td>{{ item.StudentName }}</td>

<td>{{ item.Batch }}</td>

<td>{{ item.course }}</td>

<td>{{ item.Address}}</td>

<td>{{ item.MobileNo}}</td>

<td>{{ item.EmailId}}</td>

<td>{{ item.parentMobileNo}}</td>

</tr>

{% endfor %}

</table>

{% endif %}

<form action="{% url 'display\_image1' %}">

</form>

{% endblock %}

### Modify Student Master

#### Views.py

from django.shortcuts import render, get\_object\_or\_404, redirect def edit\_student(request, student\_id): student = get\_object\_or\_404(StudentMaster, id=student\_id) if request.method == 'POST': form = StudentForm(request.POST, instance=student) # Populate form with existing student data

if form.is\_valid(): # Check if the form data is valid form.save() # Save the updated student record

return redirect('student\_detail', student\_id=student.id) # Redirect to the student's detail page else:

form = StudentForm(instance=student) # Pre-populate the form with existing student data return render(request, 'edit\_student.html', {'form': form, 'student': student}) def student\_detail(request,student\_id):

student = get\_object\_or\_404(StudentMaster,pk=student\_id) return render(request,'student\_detail.html', {'student':student}) def student\_all\_details(request):

ob=StudentMaster.objects.all().values() context={

'data':ob }

temp=loader.get\_template('student\_all\_details.html') return HttpResponse(temp.render(context,request))

#### Edit\_student.html

{% extends "master1.html" %} {% block content %}

{% load crispy\_forms\_tags %}

{% load static %}

<style> .design{ margin-top: 60px; margin-left: 550px; height:430px; width:280px; align-content: center; color: purple; backdrop-filter: blur(20px); border-radius: 20px;

} .btn\_edit{

background-color:purple; /\* Green \*/ border: none; color: white; padding: 5px 25px; text-align: center; text-decoration: none; display: inline-block; font-size: 16px; margin-left: 50px; width:70%;

border-radius: 10px; margin-top: 10px;

}

</style>

<div class="design">

<h2>Edit student</h2>

<form method="post">

{% csrf\_token %}

{{form|crispy}}

<button type="submit" class="btn\_edit">Save Changes</button>

</form>

</div> {% endblock %} **student\_detail.html**

{% extends "master1.html" %} {% block content %}

{% load crispy\_forms\_tags %}

{% load static %}

<style> .design\_data{ margin-left: 480px; margin-top: 80px; height: 380px; width: 450px; color: purple; backdrop-filter: blur(20px); border-radius: 10px;

}

.btn\_edit{ background-color:purple; /\* Green \*/ border: none;

color: white; padding: 5px 25px; text-align: center; text-decoration: none; display: inline-block; font-size: 16px; margin-left: 50px; width:70%; border-radius: 10px;

}

</style>

<div class="design\_data">

<p>StudentId: {{ student.StudentId }}</p>

<p>StudentName: {{ student.StudentName }}</p>

<p>Batch: {{ student.Batch }}</p>

<p>course: {{ student.course }}</p>

<p>Address: {{ student.Address }}</p>

<p>MobileNo: {{ student.MobileNo }}</p>

<p>EmailId: {{ student.EmailId}}</p>

<p>parentMobileNo: {{ student.parentMobileNo }}</p>

<a href="{% url 'edit\_student' student.id %}" class="btn\_edit">Modify</a>

#### </div> {% endblock %} student\_all\_details.html

{% extends "master1.html" %} {% block content %}

{% load crispy\_forms\_tags %}

{% load static %}

<style> .table { font-family: Arial, Helvetica, sans-serif; border-collapse: collapse; width: 100%;

}

.table td, .table th { border: 1px solid #ddd; padding: 8px;

}

.table tr:nth-child(even){background-color: #f2f2f2;}

.table tr:hover {background-color: #ddd;}

.table th { padding-top: 12px; padding-bottom: 12px; text-align: left; background-color:rgb(29, 29, 65); color: white; } .table-container{ margin-top: 90px;

} .head{ padding-top: 20px; padding-bottom: 20px; padding-left: 50px; text-align: left; background-color:rgb(29, 29, 65); color: white; margin-top: 150px;

} .btn1{ color: white; background: green; text-decoration: none; padding: 10px 15px; margin-left: 980px; border-radius: 5px;

} .btn2{ color: white; margin-left: -320px; background:red; padding: 10px 15px; border-radius: 5px; text-decoration: none;

}

.inner\_table{ font-family: Arial, Helvetica, sans-serif; border-collapse: collapse; width: 100%;

}

.inner\_table td, .inner\_table th { border: 1px solid #ddd; padding: 8px;

}

.inner\_table tr:nth-child(even){background-color: #f2f2f2;}

.inner\_table tr:hover {background-color: #ddd;}

.inner\_table th { padding-top: 12px; padding-bottom: 12px; text-align: left; background-color:rgb(29, 29, 65); color: white; }

.btn\_edit{ background-color: blue; color: white; padding: 5px 15px; border: none; text-decoration: none; border-radius:5px;

}

</style>

<div class="data">

<p class="head">Student Modify

<a href="{% url 'Studententry' %}" class="btn1"><ion-icon name="add-circle"></ionicon>Add New Student</a>

<a href="{% url 'delete\_Student' %}" class="btn2"><ion-icon name="trash"></ionicon>Delete</a>

</p>

<table border=1 class="inner\_table">

<tr><th>ID</th><th>StudentId</th>

<th>StudentName</th><th>Batch</th><th>course</th><th>Address</th><th>MobileNo</t h><th>EmailId</th><th>parentMobileNo</th><th>Action</th></tr>

{% for i in data %}

<tr>

<td>{{i.id}}</td>

<td>{{i.StudentId}}</td>

<td>{{i.StudentName}}</td>

<td>{{i.Batch}}</td>

<td>{{i.course}}</td>

<td>{{i.Address}}</td>

<td>{{i.MobileNo}}</td>

<td>{{i.EmailId}}</td>

<td>{{i.parentMobileNo}}</td>

<td>

<a href="{% url 'edit\_student' i.id %}" class="btn\_edit">Edit</a>

</td>

</tr>

{% endfor %}

</table>

</div>

</table>

{% endblock %}

### Batch Master

#### Models.py

#Batchmaster insert class BatchMaster(models.Model):

BatchNo = models.CharField(max\_length=10) BatchId = models.IntegerField() def \_\_str\_\_(self): return self.BatchNo

#### Forms.py

#batchmaster insert from . models import BatchMaster class BatchForm(forms.ModelForm): class Meta: model = BatchMaster fields = "\_\_all\_\_" **views.py** def Batchentry(request): context = {} context['form'] = BatchForm() return render(request,"Batchentry.html",context) def student\_valid(request): if request.method == 'POST':

student\_detail = StudentForm(request.POST) if student\_detail.is\_valid(): student\_detail.save() return HttpResponse(" ")

else:

return render(request,'Batchentry.html',{'form':student\_detail}) else:

form = StudentForm(None) return render(request,'Batchentry.html',{'form':form}) def process\_batch(request): if request.method == 'POST':

BatchNo\_inp = request.POST.get('BatchNo') BatchId\_inp = request.POST.get('BatchId') ob = BatchMaster(BatchNo=BatchNo\_inp,BatchId=BatchId\_inp) ob.save() return render(request,"return2.html") else:

return HttpResponse("Invalid request method") def displaybatch(request): batch = BatchMaster.objects.all().values() temp = loader.get\_template('displaybatch.html') context = {

'data':batch, }

return HttpResponse(temp.render(context,request)) **Urls.py**

path('Batchentry/',views.Batchentry,name='Batchentry'), path('process\_batch/',views.process\_batch,name='process\_batch'), path('displaybatch/',views.displaybatch,name='displaybatch'),

#### Batchentry.html

{% extends "master1.html" %}

{% block content %}

{% load crispy\_forms\_tags %}

{% load static %}

<style> .form{ color: purple; font-size:larger; margin-top:72px; margin-left:450px; height:420px; width: 380px; border: 1px solid black; backdrop-filter: blur(20px); border: none; border-radius: 20px; box-shadow: 0px 0px 15px 0px black;

} .inner{ position: fixed; margin-top:80px; margin-left:120px; } p{ text-align: center; justify-content: center;

} .btn{ background-color:purple; /\* Green \*/ border: none; color: white; margin-top:100px; margin-left:-50px; border-radius: 20px;

width: 150%; padding: 10px 2px; text-decoration: none; font-size: 16px;

}

</style>

<form method="post" action="{% url 'process\_batch' %}">

<div class="form">

<p>BatchMaster</p>

<div class="inner">

{% csrf\_token %}

{{form|crispy}}

<input type="submit" value="SUBMIT" class="btn">

</div>

</div>

</form> {% endblock %} **displaybatch.html**

{% extends "master1.html" %}

{% block content %}

<style> .table { font-family: Arial, Helvetica, sans-serif; border-collapse: collapse; width: 100%;

}

.table td, .table th { border: 1px solid #ddd; padding: 8px;

}

.table tr:nth-child(even){background-color: #f2f2f2;}

.table tr:hover {background-color: #ddd;}

.table th { padding-top: 12px; padding-bottom: 12px; text-align: left; background-color:rgb(29, 29, 65); color: white; } .table-container{ margin-top: 90px; } .head{ padding-top: 20px; padding-bottom: 20px; padding-left: 50px; text-align: left; background-color:rgb(29, 29, 65); color: white; } .btn1{ color: white; background: green; text-decoration: none; padding: 10px 15px; margin-left: 980px; border-radius: 5px;

} .btn2{ color: white; margin-left: -320px; background:red; padding: 10px 15px; border-radius: 5px; text-decoration: none;

}

</style>

<div class="content">

<div class="table-container">

<table border='' style='border-collapse:collapse;' class="table">

<p class="head">Batch View

<a href="{% url 'Batchentry' %}" class="btn1"><ion-icon name="add-circle"></ionicon>Add New Batch</a>

<a href="{% url 'delete\_batch' %}" class="btn2"><ion-icon name="trash"></ionicon>Delete</a>

</p>

<tr> <th>BatchNo</th> <th>BatchId</th></tr>

{% for x in data %}

<tr>

<td>{{x.BatchNo}}</td>

<td>{{x.BatchId}}</td>

</tr>

{% endfor %}

</table>

{% endblock %}

### Course Master

#### Models.py

class CourseMaster(models.Model):

CourseName = models.CharField(max\_length=10) CourseId = models.IntegerField() def \_\_str\_\_(self):

return self.CourseName

#### Forms.py

from . models import CourseMaster class CourseForm(forms.ModelForm): class Meta:

model = CourseMaster fields = "\_\_all\_\_"

#### Views.py

def Courseentry(request):

context = {} context['form'] = CourseForm() return render(request,"Courseentry.html",context) def process\_course(request): if request.method == 'POST':

CourseName\_inp = request.POST.get('CourseName') CourseId\_inp= request.POST.get('CourseId') ob = CourseMaster(CourseName = CourseName\_inp,CourseId=CourseId\_inp) ob.save() return render(request,"return3.html") else:

return HttpResponse("Inavlid request method") def displaycourse(request):

course = CourseMaster.objects.all().values() temp = loader.get\_template('displaycourse.html') context = { 'data':course,

}

return HttpResponse(temp.render(context,request)) **Urls.py**

path('Courseentry/',views.Courseentry,name='Courseentry'), path('displaycourse/',views.displaycourse,name='displaycourse'),

#### Courseentry.html

{% extends "master1.html" %}

{% block content %}

{% load crispy\_forms\_tags %}

{% load static %}

<style> .form{ color: purple; font-size:larger; margin-top:72px; margin-left:450px; height:420px; width: 380px; border: 1px solid black; backdrop-filter: blur(20px); border: none; border-radius: 20px; box-shadow: 0px 0px 15px 0px black;

} .inner{ position: fixed; margin-top:80px; margin-left:120px; } p{ text-align: center; justify-content: center;

} .btn{ background-color:purple; /\* Green \*/ border: none; color: white; margin-top:100px; margin-left:-50px; border-radius: 20px;

width: 150%; padding: 10px 2px; text-decoration: none; font-size: 16px;

}

</style>

<form method="post" action="{% url 'process\_course' %}">

<div class="form">

<p>CourseMaster</p>

<div class="inner">

{% csrf\_token %}

{{form|crispy}}

<input type="submit" value="SUBMIT" class="btn">

</div>

</div>

</form> {% endblock %} **displaycourse.html**

{% extends "master1.html" %}

{% block content %}

<style> .table { font-family: Arial, Helvetica, sans-serif; border-collapse: collapse; width: 100%;

}

.table td, .table th { border: 1px solid #ddd; padding: 8px;

}

.table tr:nth-child(even){background-color: #f2f2f2;}

.table tr:hover {background-color: #ddd;}

.table th { padding-top: 12px; padding-bottom: 12px; text-align: left; background-color:rgb(29, 29, 65); color: white; } .table-container{ margin-top: 90px; } .head{ padding-top: 20px; padding-bottom: 20px; padding-left: 50px; text-align: left; background-color:rgb(29, 29, 65); color: white; } .btn1{ color: white; background: green; text-decoration: none; padding: 10px 15px; margin-left: 980px; border-radius: 5px;

} .btn2{ color: white; margin-left: -320px; background:red; padding: 10px 15px; border-radius: 5px; text-decoration: none;

}

</style>

<div class="content">

<div class="table-container">

<table border='' style='border-collapse:collapse;' class="table">

<p class="head">Course View

<a href="{% url 'Courseentry' %}" class="btn1"><ion-icon name="add-circle"></ionicon>Add New Course</a>

<a href="{% url 'delete\_course' %}" class="btn2"><ion-icon name="trash"></ionicon>Delete</a>

</p>

<tr> <th>CourseName </th><th>CourseId</th></tr>

{% for x in data %}

<tr>

<td>{{x.CourseName}}</td>

<td>{{x.CourseId}}</td>

</tr>

{% endfor %}

</table>

</div>

</div>

{% endblock %}

### College Master

#### Models.py

class CollegeMaster(models.Model):

CollegeCode = models.IntegerField()

CollegeName = models.CharField(max\_length=80) def \_\_str\_\_(self): return self.CollegeName

#### Forms.py

from . models import CollegeMaster class CollegeForm(forms.ModelForm): class Meta:

model = CollegeMaster fields = "\_\_all\_\_"

#### Views.py

def Collegemaster(request):

context = {} context['form'] = CollegeForm() return render(request,"Collegeentry.html",context) def process\_college(request): if request.method == 'POST':

CollegeCode\_inp = request.POST.get('CollegeCode') CollegeName\_inp = request.POST.get('CollegeName') ob = CollegeMaster(CollegeCode=CollegeCode\_inp,CollegeName=CollegeName\_inp) ob.save() return render(request,"return4.html") else:

return HttpResponse("Invalid request method") def displaycollege(request):

college = CollegeMaster.objects.all().values() temp = loader.get\_template('displaycollege.html') context = {

'data':college,

}

return HttpResponse(temp.render(context,request)) **Urls.py**

path('Collegemaster/',views.Collegemaster,name='Collegemaster'), path('process\_college/',views.process\_college,name='process\_college'), path('displaycollege/',views.displaycollege,name='displaycollege'),

#### Collegeentry.html

{% extends "master1.html" %} {% block content %}

{% load crispy\_forms\_tags %}

{% load static %}

<style> .form{ color: purple; font-size:larger; margin-top:72px; margin-left:450px; height:420px; width: 380px; border: 1px solid black; backdrop-filter: blur(20px); border: none; border-radius: 20px; box-shadow: 0px 0px 15px 0px black;

} .inner{ position: fixed; margin-top:80px; margin-left:120px; } p{ text-align: center; justify-content: center;

}

.btn{

background-color:purple; /\* Green \*/ border: none; color: white; margin-top:100px; margin-left:-50px; border-radius: 20px; width: 150%; padding: 10px 2px; text-decoration: none; font-size: 16px;

}

</style>

<form method="post" action="{% url 'process\_college' %}">

<div class="form">

<p>CollegeMaster</p>

<div class="inner">

{% csrf\_token %}

{{form|crispy}}

<input type="submit" value="SUBMIT" class="btn">

</div>

</div> </form> {% endblock %}

**11.Feature Enhancements**

1. **User & Role Management**

o **Role-Based Access Control (RBAC):**

* + - Assign different roles (Admin, Teacher, Counselor, Student) with specific permissions.
    - Example: Teachers can record behavior incidents, Counselors can review and intervene, and Admins can manage users and reports.
  + **Authentication & Security:**
    - Implement two-factor authentication (2FA) for Admin and Teacher accounts.
    - Password reset link expiration (e.g., within 30 minutes).

1. **Behavior Tracking & Incident Management** o **Incident Logging:**
   * + Enable teachers to log behavioral incidents with date, time, and severity levels.
     + Categorization of incidents (e.g., Minor, Major, Severe) with predefined resolution steps.
   * **Real-time Behavior Status:**
     + Display a student’s behavioral record with history and corrective actions taken.
     + Allow real-time updates on behavior improvements or repeated offenses.
   * **Automated Alerts:**
     + Notify parents and counselors immediately for serious incidents.
     + Alert students about warnings and disciplinary actions.
2. **Student Performance & Engagement Monitoring** o **Behavior Analytics Dashboard:**
   * + Provide real-time insights into student behavior trends.
     + Identify high-risk students needing early intervention.
   * **Behavioral Score & Progress Tracking:**
     + Assign behavior scores based on incidents and improvements.
     + Generate progress reports for parents, teachers, and counselors.
3. **Disciplinary Actions & Rewards System** o **Customizable Disciplinary Actions:**
   * + Set up predefined disciplinary measures like detention, counseling sessions, and suspensions.
     + Provide restorative justice options such as apology letters and community service.
   * **Positive Reinforcement System:**
     + Reward students for good behavior with badges, certificates, or privileges.
     + Display top-behaved students and issue recognition certificates.
4. **Reports & Analytics** o **Behavioral Reports:**

▪ Generate reports on common disciplinary issues, student-wise behavior records, and resolution statistics.

* + **Counseling & Intervention Reports:**
    - Provide insights into students who have undergone interventions and their progress over time.

1. **Notifications & Alerts** o **Automated Email/SMS Alerts:**

▪ Alerts for parents, teachers, and students regarding incidents, resolutions, and scheduled counseling sessions.

* + **Event & Announcement System:**
    - Display upcoming behavioral workshops, motivational sessions, and policy changes.

1. **Integration & Security** o **Parental Access & Communication:**

▪ Provide a parent portal for viewing student behavior reports and communicating with teachers.

* + **AI-Based Predictive Analysis:**
    - Implement AI to analyze behavior patterns and suggest intervention strategies.

### Results and Discussions

The **Student Behavior Management System (SBMS)** has significantly improved student discipline, engagement, and overall learning environments. The key benefits observed are:

1. **Improved Behavior Tracking**
   * The automation of behavior logging and tracking has minimized manual efforts for teachers and administrators. o Real-time updates on student behavior status have streamlined disciplinary actions.
2. **Increased Transparency & Accountability** o Parents and counselors have real-time access to students' behavioral records. o Incident history and resolutions provide clarity on past actions and improvements.
3. **Enhanced Communication**
   * Automated notifications for incidents, interventions, and progress updates ensure timely communication between stakeholders.
   * Teachers, parents, and counselors can collaborate effectively for better student outcomes.
4. **Better Student Development & Intervention**
   * Early detection of behavioral patterns enables timely interventions to prevent escalations. o Counseling reports provide valuable insights into student progress and necessary improvements.
5. **Data-Driven Decision Making**
   * Behavioral analytics allow administrators to identify trends, common disciplinary issues, and areas for improvement. o Reports on high-risk students help schools develop targeted behavioral improvement programs.
6. **Paperless & Digital Record Management** o The system has minimized the need for paper-based record-keeping. o Secure digital records ensure data integrity and easy access for authorized personnel.

### 12.Conclusion

The **Student Behavior Management System (SBMS)** has transformed the way educational institutions track and manage student behavior, fostering a more structured and supportive learning environment. By automating behavior tracking, interventions, and reporting, the system has reduced administrative workload while enhancing transparency and accountability.

Furthermore, the integration of automated notifications, digital records, and behavior analytics has enabled data-driven decision-making, improving student engagement and discipline. The system’s predictive capabilities allow early intervention, ensuring that students receive the support they need in a timely manner.

Moving forward, future enhancements such as AI-driven behavior predictions, mobile applications, and deeper parental engagement features will further improve the system’s effectiveness and user experience. Overall, SBMS has proven to be a valuable tool in fostering a positive, disciplined, and student-friendly academic environment.

**13.BIBLIOGRAPHY**

1. **Alberto, P. A., & Troutman, A. C.** (2013). *Applied behavior analysis for teachers* (9th ed.). Pearson Education.
   * This book provides an overview of applied behavior analysis (ABA) techniques, which are crucial for designing and implementing behavior management systems in classrooms.
2. **Emmer, E. T., & Sabornie, E. J.** (2015). *Handbook of classroom management* (2nd ed.). Routledge.
   * This handbook offers extensive guidance on various classroom management techniques, including behavior management strategies, providing practical approaches for educators.
3. **Jones, V. F., & Jones, L. S.** (2012). *Comprehensive classroom management:*

*Creating communities of support and solving problems* (10th ed.). Pearson. o Focuses on developing proactive behavior management strategies in schools, fostering supportive learning environments.

1. **Simonsen, B., Fairbanks, S., Briesch, A., Myers, D., & Sugai, G.** (2008). *Evidencebased practices in classroom management: Considerations for research to practice*. Education and Treatment of Children, 31(3), 351-380. o Explores the implementation of evidence-based practices and approaches in managing student behavior.

**Journal Articles:**

1. **Baker, M. L., & Proctor, S. L.** (2000). "An analysis of student behavior management techniques in the elementary school classroom." *Journal of Educational Psychology*, 92(3), 598-608.
   * This article reviews and evaluates various behavior management techniques and their effectiveness in elementary school settings.
2. **McIntosh, K., Chard, D. J., & Boland, J. R.** (2007). "The impact of proactive behavior management on student behavior: A study in an urban middle school." *Journal of Positive Behavior Interventions*, 9(3), 137-146.
   * Provides evidence from an intervention study focused on using proactive behavior management techniques in middle schools.

**Websites:**

[https://www.pbis.org](https://www.pbis.org/) [https://www.ed.gov](https://www.ed.gov/)

[https://www.niet.org](https://www.niet.org/)