DATABASE SYSTEMS

Faculty Management System

TEAM :

CS22B1030-K BALA SAI MANVITHA
CS22B1049-D CHAITANYA ABHINAV
CS22B1078-P KEERTHI REKHA
CS22B2030-V SRI MANASWINI
CS22B2044-M CHINNA RAYUDU

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to articulate the detailed requirements for the development of a comprehensive database. In response to the dynamic nature of academic institutions and the critical need for efficient faculty allocation, this document outlines the intricate requirements for the development of a Faculty Management System. Designed to optimize the scheduling and utilization of faculty members, this system aims to enhance operational agility and promote a conducive learning environment within the institution.

1.2 Intended Audience and Reading Suggestions

This project is a prototype for the Institute Students and it is restricted within the institute premises.This project is useful for all student organisations or groups for displaying information and keeping students know when the faculty is available.The intended audience typically includes Administrators and Academic heads(Deans),Faculty Member,Support Staff,Students,etc.
1.Product name, scope and purpose
2.Functional requirements(use case diagram)
3 Detailed requirement specifications.
4.Hardware, software specifications
5.Proposed ER model ,ER schema

1.3 Project Scope

The project scope for the Faculty Management System encompasses the development of a comprehensive platform enabling faculty members and TA's to input their availability for teaching and academic duties while providing administrative staff with tools for scheduling and allocation based on this availability and also detailing about their projects.

1.4 References

- Fundamentals of Database Systems by Ramez Elmasri, Shamkant B. Navathe.
- https://www.geeksforgeeks.org/data-models-in-dbms/
- Others: Stack Overflow, Javatpoint.

2 Overall Description

2.1 Product Perspective

Faculty members often face challenges in maintaining their availability records, resulting in potential scheduling conflicts and miscommunication. This proposed solution seeks to bridge this gap by providing a sophisticated software platform tailored to address the specific needs of faculty availability management. By offering a user-friendly interface and robust functionalities, the system aims to streamline the process of recording faculty availability, ensuring accuracy, and facilitating seamless communication between faculty members and students.

2.2 Product Features

Key features of the system include:

- Manage faculty member information (name, department, contact no, email).
- Manage TA information (email, id).
- Manage project information (name, id, domain).
- Manage cabin information (number, availability status, location).
- Manage role information (role name,role id,email).
- Allocate cabins to faculty.
- Allocate cabins to TA's.
- Manage faculty project involvement.
- Assign TA's to faculty.
- Assign roles for particular faculty.
- Track faculty availability for meetings or office hours.

2.3 User Classes and Characteristics

User classes include administrators, faculty members, and support staff. Each class has specific roles and responsibilities within the system.

2.4 Operating Environment

The system will operate on modern web browsers and integrate with backend databases such as MySQL.

2.5 Design and Implementation Constraints

The system design should prioritize scalability, security, and user-friendliness. Integration with existing systems may require API development and database management.

3 System Features

3.1 Functional Requirements

3.1.1 User Management

- Users will be able to register as either faculty members or teaching assistants.
- Students should also be able to log in to the system using their credentials and have viewing access.
- Faculty members will be able to update their profile information, including contact details and availability preferences.

3.1.2 Availability Management

- Faculty members will be able to input their availability at any point in time.
- The availability input shall include yes-or-no options.Faculty shall be able to set availability options, such as YES if they are available in the cabin or NO if they are not available in the cabin at that particular point in time.
- Faculty will be able to update their availability as needed.

3.1.3 Schedule Management

- Students and administrators will be able to view the availability of faculty members.
- Administrators shall be able to schedule classes, meetings, or other activities based on available staff.

3.2 Nonfunctional Requirements

3.2.1 Performance Requirements:

- **Response Time:**
  - System should respond to user actions within a defined time frame (e.g., < 2 seconds for page loads).
- **Scalability:**
  - Ability to handle increasing numbers of users and data without significant degradation in performance.

3.2.2 Safety Requirements

- **Data Integrity:**
  - Ensure that user data remains accurate and consistent throughout the system's operations.
- **Disaster Recovery:**
  - Implement measures to recover data and resume system functionality in case of unexpected failures.
- **Backup Mechanisms:**
  - Regularly backup system data to prevent loss due to accidental deletion or corruption.

3.2.3 Security Requirements

- **Authentication and Authorization:**
  - Secure user authentication and role-based access control to prevent unauthorized access to sensitive information.
- **Data Encryption:**
  - Encrypt sensitive data during transmission and storage to protect against unauthorized interception or access.

3.2.4 Software Quality Attributes

- **Reliability:**
  - Ensure the system operates consistently and reliably under normal and adverse conditions.
- **Usability:**
  - Provide an intuitive and user-friendly interface that minimizes the learning curve for users.
- **Maintainability:**
  - Design the system with clear code structure and documentation to facilitate future updates and enhancements.
- **Portability:**
  - Ensure the system can be easily deployed and operated across different platforms andÂ environments.

4 External Interface

4.1 User Interfaces

- **Web and Mobile:**
  - Easy-to-use interfaces accessible via web browsers and mobile devices.
- **Role-based Access:**
  - Customized interfaces based on user roles for faculty and administrators.
- **Dashboard:**
  - Quick-view dashboard showing upcoming events and notifications.

4.2 Software Interfaces

- **Database Integration:**
  - Seamless interaction with a database system for efficient data management.
- **External Systems:**
  - Integration with other software like student information and payroll systems.

- **API Access:**
  - Ability to interact with the system's features through APIs for custom integrations.

4.3 Hardware Interfaces:

- **Device Compatibility:**
  - Support for various devices such as computers, laptops, tablets, and smartphones.
- **Network Connectivity:**
  - Operability over wired and wireless networks using standard protocols.

5 External Interfaces Requirements

5.1 User Interfaces

- The system should have a user-friendly web-based interface accessible from various devices with modern web browsers.
- Integration with a database management system (e.g., MySQL) for storing and retrieving faculty-related data.

5.2 Software Interfaces

- Integration with the institute's existing authentication system for user login and access control.
- The interface should comply with accessibility guidelines (e.g., WCAG) to ensure usability for users with disabilities.

6 Appendices

6.1 Glossary

List of terms and definitions used throughout the document.

- SRS - Software Requirements Specification
- FMS - Faculty Management System
- API - Application Programming Interface

- ER - Entity-Relationship
- ERD - Entity-Relationship Diagram
- DBMS - Database Management System
- TA - Teaching Assistant
- PIC - Professor In Charge
- WCAG - Web Content Accessibility Guidelines

6.2 Analysis Models

Include relevant analysis models such as Entity-Relationship Diagrams (ERD), Physical Data Model, Conceptual Data Model, and Logical Data Model specific to the Faculty Management System.

6.2.1 Conceptual Data Model

- **Purpose :**
  - The conceptual data model provides a high-level view of the business requirements without going into technical details.
- **Content :**
  - It includes the main concepts (entities) required to store information.
  - Relationships between these entities are defined.
  - No specific details about each piece of information are included.
- **Representation :**
  - Conceptual data models are often depicted using a data structure diagram (DSD) or a simplified entity-relationship diagram (ERD).
- **Use Case :**
  - Helps define the scope of a business solution.
  - Understandable by management and users, not just technical staff.

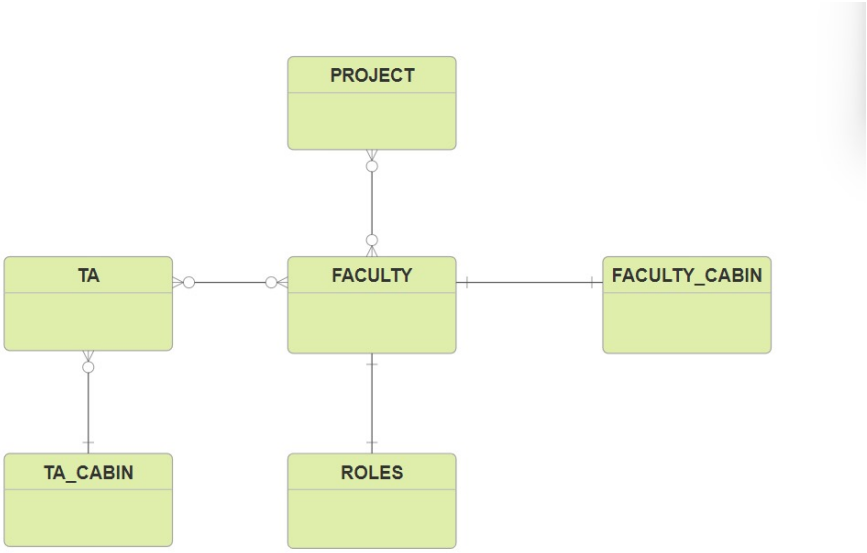

Figure 1: Conceptual Data Model.

6.2.2 Logical Data Model

- **Purpose :**
  - The logical data model goes beyond the conceptual model and provides more detail.
- **Content :**
  - Includes entities, relationships, and details on entities, attributes.
  - Defines primary keys (unique identifiers) and foreign keys (relationships between entities).
- **Representation :**
  - A Logical Data Model (LDM) is a conceptual representation of an organization's data assets, focusing on the structure, relationships, and constraints of the data independent of any specific technology or implementation details.

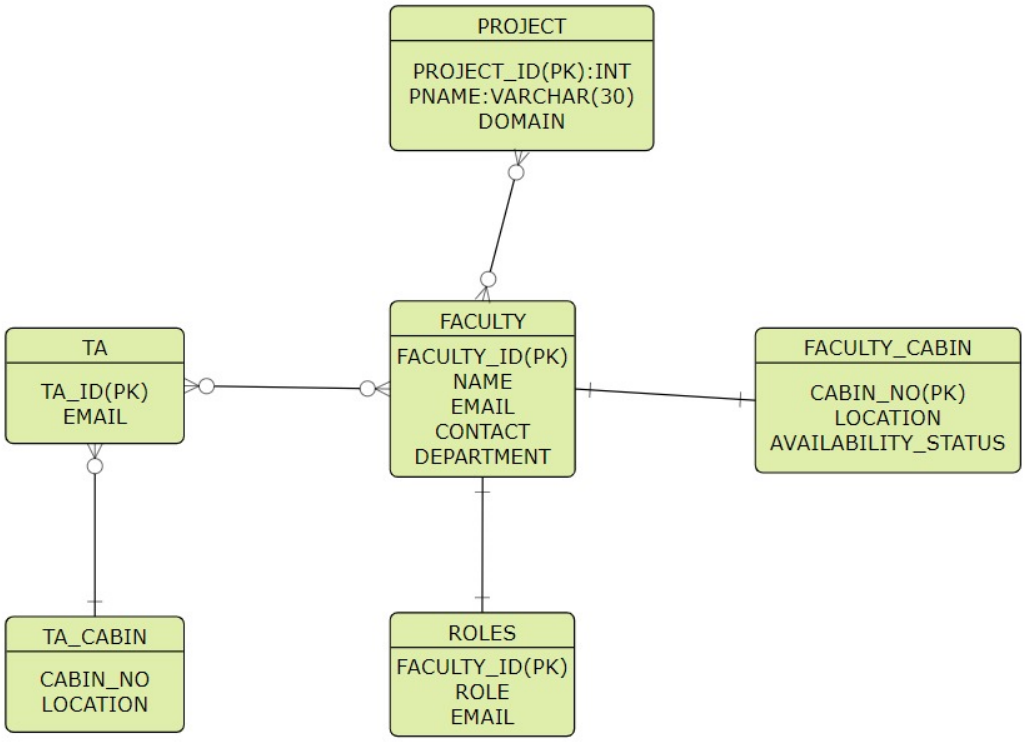

Figure 2: Logical Data Model.

- **Use Case :**
  - Used for designing the database structure.
  - Guides developers in creating the actual database.

6.2.3 Physical Data Model

- **Purpose :**
  - The physical data model is used for actual implementation.
- **Content :**
  - Specifies data types, naming conventions, and other technology-specific features.
  - Maps the logical model to a specific database provider (e.g., Oracle, SQL Server, MySQL).

- **Representation :**
  - A physical data model (PDM) is a representation of a database design at a level of detail that is suitable for implementing the database on a specific database management system (DBMS).

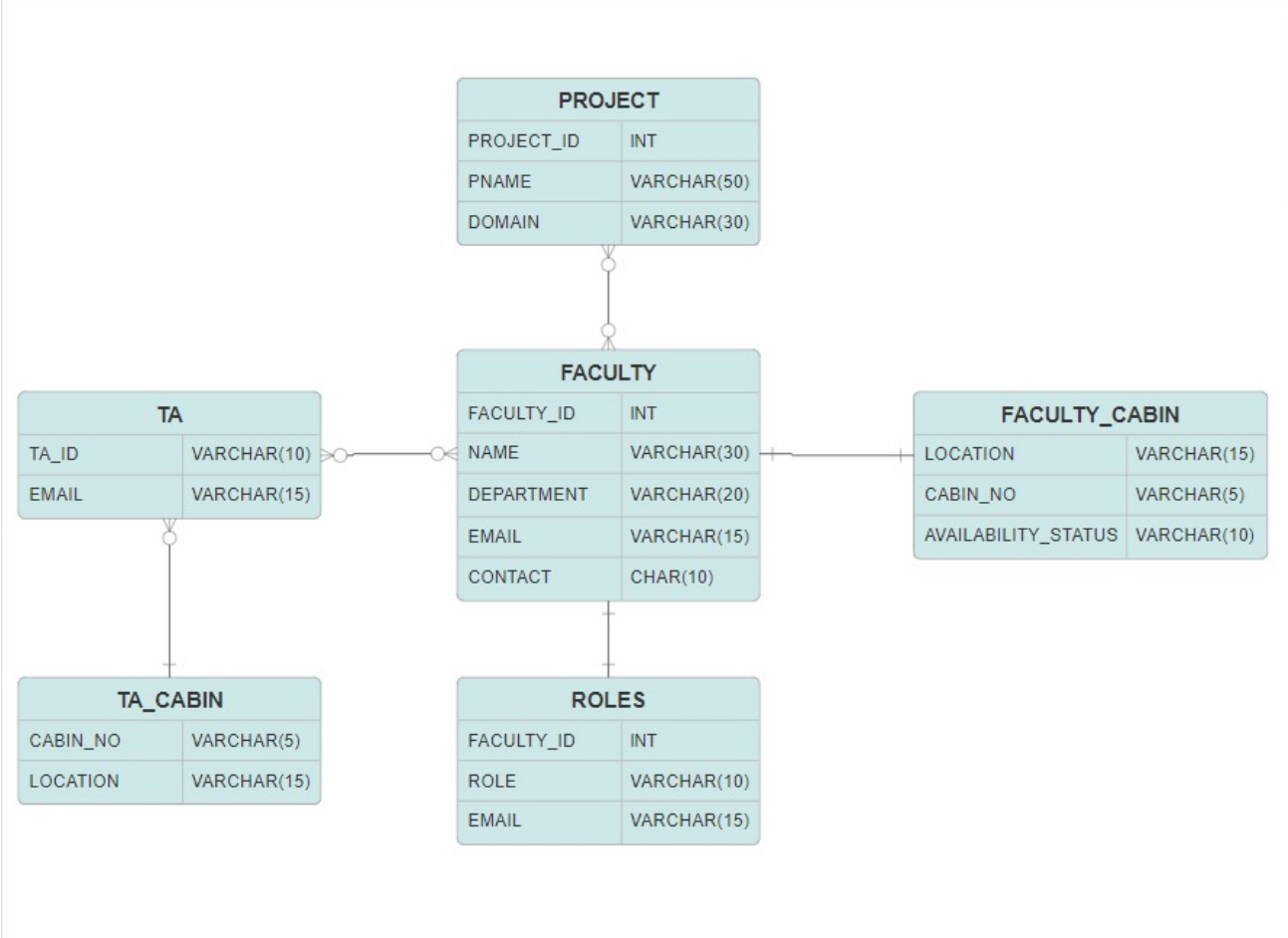

Figure 3: Physical Data Model.

- **Use Case :**
  - Guides database administrators in building the physical database.
  - Considers storage, indexing, and performance aspects.

6.2.4 Entity-Relationship Diagrams (ERD)

- An Entity-Relationship (ER) diagram is a powerful tool used in database management systems (DBMS) to visually represent the structure of a database. It consists of
  - **Entities :**
    - Entities represent real-world objects, concepts, or things that we want to store information about in our database.
    - In an ER diagram, entities are typically depicted as rectangles.
    - In our database, Entities are Faculty, Faculty_Cabin, TA, TA_Cabin,Roles and Project.
  - **Attributes :**
    - Attributes describe the properties or characteristics of an entity.
    - Key attributes are essential and often serve as primary keys for uniquely identifying each entity.
    - Composite attributes are composed of multiple sub-attributes.
    - Multivalued attributes can have more than one value.
    - Derived attributes are calculated from other attributes
      - **For Example**, attributes for Faculty entity in our database are like Faculty id,name,contact number, email id, department which are the general features related to any faculty.
  - **Relationships :**
    - Relationships define how entities are related to each other.
    - They represent associations between entities.
    - Common relationship types include:
      - **One-to-One:** When one instance of an entity is associated with only one instance of another entity.
      - In our database, each faculty is associated with a cabin and each cabin is allotted to a single faculty member. Hence entities faculty and faculty_cabin are in 1:1 relationship.Similarly, each faculty is associated with a role and each role such as PIC's, dean's can be given to a single faculty member.
      - **One-to-Many:** Where one instance of an entity is related to multiple instances of another entity.

        In case of Teaching Assistants(TA), many TA's can be given a single cabin.Hence TA and TA_Cabin are in N:1 relationship.
      - **Many-to-Many:** When multiple instances of one entity are associated with multiple instances of another entity.

- In our database, each faculty can assist any number of TA's and each TA can work under multiple faculty members.So faculty and TA are in M:N relationship.Similarly, each faculty can work on multiple projects and each project can be handled by multiple faculty.
- **Total Participation:**Every entity occurrence in the entity set must participate in the relationship. It is represented by a double line joining entity and relationship.
- **Partial Participation:** Not all entity occurrences are required to participate in the relationship.It is represented by a single line joining entity and relationship.

* As every faculty is allocated a cabin, faculty has total participation in allocation relationship, while every cabin need not be allocated to a faculty or TA. Hence cabins have partial participation.Also, every TA need not have a cabin allocated.So, TA has a partial participation in allocation relationship.
* In Assists relationship, every faculty need not assist a TA,but every TA works under a faculty.So faculty is partial and TA is total. Similarly, in works on relationship every faculty need not work on a project,but a project doesn't exist without any faculty working on it.Hence faculty is partial and project is total.Also in manages relationship, every role is assigned to some faculty whereas every faculty cannot be assigned to that special role.Therefore, roles have total and faculty has partial participation.Relationships are typically represented by diamonds or rhombuses in the ER diagram.

– **Purpose:**

* ER diagrams help us design the logical structure of a database.
* They provide a high-level view of how entities are connected and how data flows.
* These diagrams aid in communication between developers, designers, and stakeholders during the database design process.

– ER diagrams serve as a blueprint for designing databases, making them easier to understand and maintain. They play a crucial role in creating efficient and well-organized database systems.

- **Representation :**

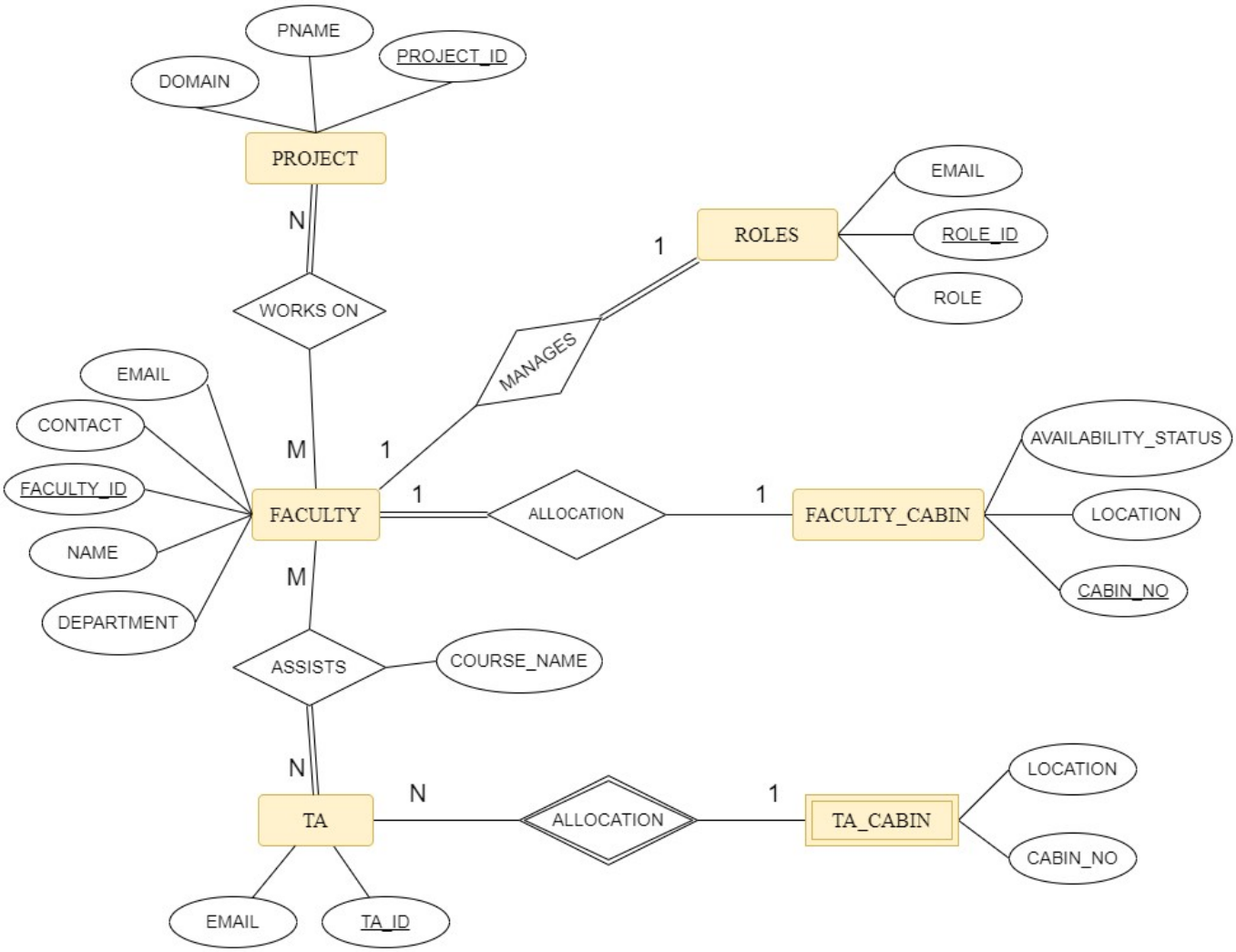

Figure 4: ERD.

6.2.5 Schema

- **Purpose :**
  - The schema description outlines the structure and organization of a database without delving into implementation specifics or technical details.
- **Content :**
  - Entities: Defines the main components or objects within the database.
  - Relationships: Specifies the connections and interactions between these entities.
  - Attributes: Describes the characteristics or properties of each entity.
- **Use Case :**
  - Facilitates understanding of the data model's design and organization.
  - Serves as a communication tool between stakeholders, including non-technical personnel, to discuss and validate data requirements.
- **Translating an ER diagram into Relational Database(Schema) :**
  Translating an Entity-Relationship (ER) diagram into a relational database schema involves converting entities, relationships, attributes, and their respective cardinalities into tables, columns, and constraints.
  Each entity becomes a table, each attribute becomes a column, and relationships are represented using foreign keys. Cardinalities dictate the relationships between tables. This process helps organize and structure data effectively within a relational database management system (RDBMS), ensuring data integrity and efficient querying.
  In our database, entities like Faculty and TA are in M:N relationship. For such relationships, a new entity is created with the name as its relationship's name. That new entity has primary keys of both entities (Faculty and TA) as its attributes.Also, additional attributes corresponding to that relationship are added.Similarly, Faculty and Project are in M:N relationship and hence a new entity called works_on is created with both the primary keys as attributes.
  For entities in one to many relationships, like TA and TA_Cabin (N:1 relationship) the primary key of TA_Cabin acts as foreign key in TA entity and hence we add it as an attribute in TA entity.
  For entities like Faculty and Faculty_Cabin(1:1 relationship), the primary key of Faculty_Cabin acts as foreign key in the Faculty entity because of the partial participation of the Faculty_Cabin entity.
  Now, the relationship between entities is represented by lines joining foreign keys and corresponding primary keys, where the line points towards primary keys.

- **Representation :**
  - Schemas are typically represented using a schema diagram , showcasing the entities, relationships, and attributes in a visual format.

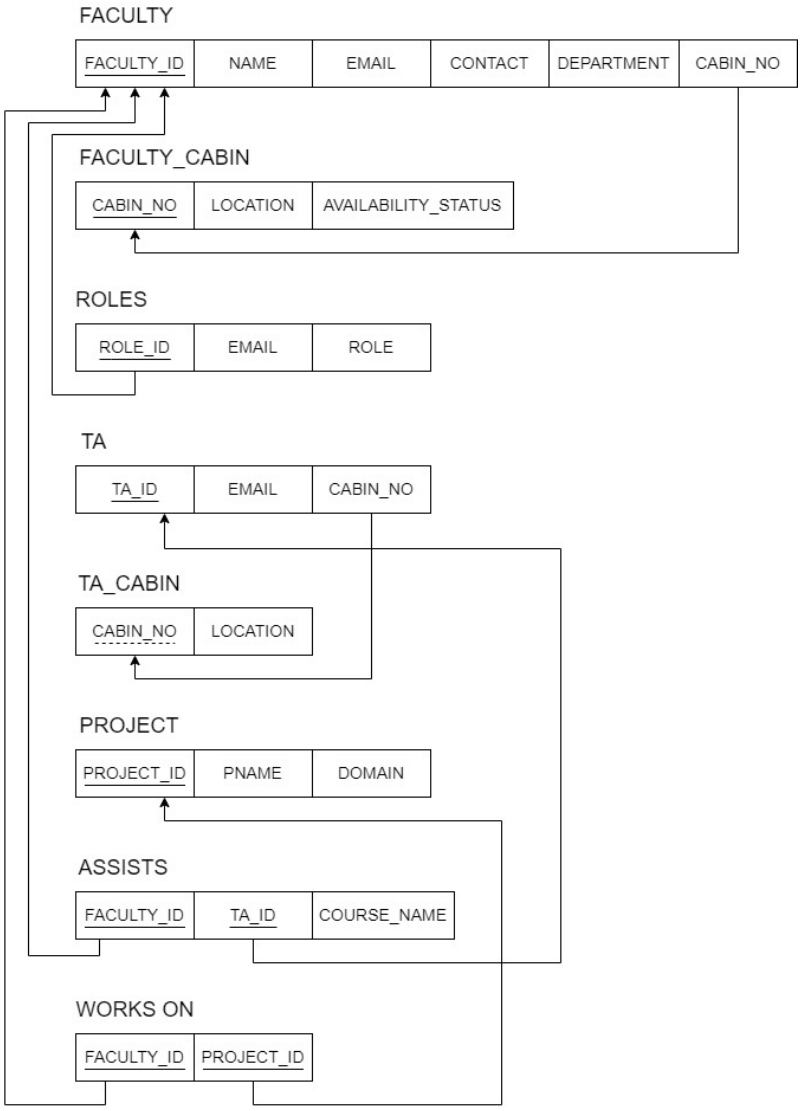

Figure 5: Schema.

7 Implementation

7.1 Database

The database is created in MySql and here is the description of the tables or entities.

```
mysql> use project1
Database changed
mysql> show tables;
+--------------------+
| Tables_in_project1 |
+--------------------+
| assists            |
| availability       |
| faculty            |
| project            |
| ta                 |
| works_on           |
+--------------------+
6 rows in set (0.01 sec)

mysql> desc faculty;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| faculty_id | int         | NO   | PRI | NULL    |       |
| name       | varchar(30) | YES  |     | NULL    |       |
| contact    | bigint      | YES  |     | NULL    |       |
| email      | varchar(30) | YES  |     | NULL    |       |
| department | varchar(30) | YES  |     | NULL    |       |
| cabin_no   | int         | YES  |     | NULL    |       |
| location   | varchar(30) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
7 rows in set (0.01 sec)

mysql> desc availability;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| faculty_id   | int         | YES  |     | NULL    |       |
| availability | varchar(10) | YES  |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Figure 6: Database 1.

```
mysql> desc project;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| project_id | int         | NO   | PRI | NULL    |       |
| pname      | varchar(30) | YES  |     | NULL    |       |
| domain     | varchar(30) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)

mysql> desc assists;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| faculty_id  | int         | YES  |     | NULL    |       |
| ta_id       | int         | YES  |     | NULL    |       |
| course_name | varchar(30) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> desc ta;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| ta_id       | int         | NO   | PRI | NULL    |       |
| email       | varchar(30) | YES  |     | NULL    |       |
| cabin_no    | int         | YES  |     | NULL    |       |
| location    | varchar(30) | YES  |     | NULL    |       |
| course_name | varchar(30) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> desc works_on;
+------------+------+------+-----+---------+-------+
| Field      | Type | Null | Key | Default | Extra |
+------------+------+------+-----+---------+-------+
| project_id | int  | YES  |     | NULL    |       |
| faculty_id | int  | YES  |     | NULL    |       |
+------------+------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Figure 7: Database 2.

7.2 Libraries and Frameworks

- We're using Flask, a popular Python framework for building web applications. Flask provides the basic structure for our backend code.
- Flask provides the core structure for our application, handling functionalities like routing user requests, rendering dynamic webpages, and interacting with the database.
- We're also using libraries like mysql.connector to connect to our MySQL database. This allows us to store and retrieve data from the database.

7.3 User Login and Authentication

- Our website uses a login system to restrict access to authorized users.
- We've implemented user authentication using Flask-Login, a Flask extension.
- When a user logs in, the username and password are checked against a predefined dictionary of users (users in this code).
- If the credentials match, a session is created, and the user is marked as logged in. This session is maintained using a secret key ('abhi' in this code).
- The login_required decorator ensures that only logged-in users can access certain pages of our website.

7.4 Database Interaction

- We're using a MySQL database to store various data like faculty information, projects, TAs, and their associations.
- The code includes functions to insert new data (insert_faculty_data, etc.) and retrieve existing data (faculty_details, etc.) from the database.
- We're using prepared statements to prevent SQL injection vulnerabilities. This helps protect our database from malicious attacks.

7.5 Routes and Templates

- The @app.route decorator defines different routes for our website. These routes correspond to different URLs.
- For example, the /login route handles the login functionality, while the /addproject route handles adding a new project.
- Flask uses templates to dynamically generate HTML content. The code uses the render_template function to render these templates and pass data to them.

7.6 Security Considerations

- It's important to remember that this is a basic example, and real-world applications would have additional security measures in place.

- For instance, we're using a simple dictionary to store user credentials in this example. In a production system, you would likely use a more secure method like hashing passwords.

7.7 Technology Stack

The following core technologies were used to build the backend:

- Programming Language: Python (a popular and versatile language)
- Web Framework: Flask (lightweight and flexible for web development)
- Database: MySQL (widely used and reliable relational database)
- Database Connector: mysql.connector library (enables secure communication with the MySQL database)
- Template Engine: Jinja2 (default templating engine for Flask)
- Flask-Login: (extension for user login and authentication management)

7.8 Data Flow

- **User Interaction :** Users interact with the website's frontend (HTML, CSS, JavaScript) by submitting forms or making requests.
- **Frontend-Backend Communication :** The frontend sends these requests (e.g., login credentials, project data) to the backend through API endpoints or form submissions.
- **Backend Processing :**
  - Flask routes the requests to appropriate functions based on the URL and HTTP method (GET, POST, etc.).
  - User login credentials are validated against hashed passwords stored in the database.
  - For data manipulation requests (adding projects, updating faculty information), the backend interacts with the database using prepared SQL statements.
  - The backend logic processes the requests, performs necessary actions (e.g., inserting data, updating records), and interacts with the database.
- **Response Generation :** The backend generates a response based on the request. This might involve retrieving data from the database and formatting it for the frontend (e.g., JSON for APIs, HTML for webpages).

- **Response Delivery :** The backend sends the response back to the frontend.
- **Frontend Update :** The frontend receives the response and updates the user interface accordingly (e.g., displaying success messages, populating tables with data).

7.9 Connecting Frontend and Backend

- **Forms :** HTML forms with appropriate HTTP methods (POST for data submission) are used to send data from the frontend to specific backend routes.

7.10 Conclusion

This report has comprehensively explained the methodology, implementation details, and technology stack employed to develop the website's backend. The backend serves as the foundation, ensuring secure user authentication, efficient database interaction, and overall application logic. By leveraging Flask and other well-established technologies, we've built a robust and scalable backend that can support the website's functionalities. Future enhancements could involve integrating additional security measures and expanding functionalities as needed.

7.11 Source Code

```
from flask import Flask, render_template, request, flash, redirect, url_for
from flask_login import login_user, logout_user, login_required, current_user
from flask_login import UserMixin, LoginManager
import mysql.connector

app = Flask(__name__)
app.secret_key = 'abhi'

login_manager = LoginManager()
login_manager.init_app(app)

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="Abhinav@2005",
    database="project1"
)

mycursor = mydb.cursor()

class User(UserMixin):
    def _init_(self, id):
        self.id = id

    @property
    def is_authenticated(self):
        return True

    @property
    def is_active(self):
        return True

    @property
    def is_anonymous(self):
        return False

    def get_id(self):
        return str(self.id)

users = {'admin': {'password': 'password'}, 'user1': {'password': '123456'}}
```

```
@login_manager.user_loader
def load_user(user_id):
    user_data = users.get(user_id)
    if user_data:
        return User(user_id)
    return None

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        if username in users and users[username]['password'] == password:
            user = User(username)
            login_user(user)
            flash('Logged in successfully!', 'success')
            return redirect(url_for('index'))
        else:
            flash('Invalid username or password', 'danger')
            return redirect(url_for('login'))
    else:
        return render_template('login.html')


@app.route('/logout')
@login_required
def logout():
    logout_user()
    flash('Logged out successfully!', 'success')
    return redirect(url_for('index'))

def insert_project_data(project_id, pname, domain):
    try:
        insert_sql = "INSERT INTO project (project_id, pname, domain) VALUES (%s, %s, %s)"
        insert_val = (project_id, pname, domain)
        mycursor.execute(insert_sql, insert_val)
        mydb.commit()
        return "Project data inserted successfully!"
    except mysql.connector.IntegrityError:
        return "Project with ID {} already exists.".format(project_id)
```

```
def insert_availability(FACULTY_ID,availability_status):
    try:
        insert_sql = "INSERT INTO availability (FACULTY_ID,availability_status) VALUES (%s,%s)"
        insert_val = (FACULTY_ID,availability_status)
        mycursor.execute(insert_sql, insert_val)
        mydb.commit()
        return "Availability data inserted successfully!"
    except mysql.connector.IntegrityError:
        return "Availability data for Faculty ID {} already exists.".format(FACULTY_ID)

def insert_faculty_data(FACULTY_ID, NAME, CONTACT, EMAIL, DEPARTMENT, cabin_no, location):
    try:
        insert_sql = "INSERT INTO faculty (FACULTY_ID, NAME, CONTACT, EMAIL, DEPARTMENT,
                      cabin_no, location) VALUES (%s, %s, %s, %s, %s, %s, %s)"
        insert_val = (FACULTY_ID, NAME, CONTACT, EMAIL, DEPARTMENT,cabin_no, location)
        mycursor.execute(insert_sql, insert_val)
        mydb.commit()
        return "Faculty data inserted successfully!"
    except mysql.connector.IntegrityError:
        return "Faculty with ID {} already exists.".format(FACULTY_ID)

def insert_TA_data(ta_id, email,cabin_no, location,course_name):
    try:
        insert_sql = "INSERT INTO ta (ta_id,email,cabin_no, location,course_name)
                      VALUES (%s, %s, %s, %s, %s)"
        insert_val = (ta_id,email,cabin_no, location,course_name)
        mycursor.execute(insert_sql, insert_val)
        mydb.commit()
        return "TA data inserted successfully!"
    except mysql.connector.IntegrityError:
        return "TA with ID {} already exists.".format(ta_id)

def insert_assists_data(faculty_id,ta_id,course_name):
    try:
        insert_sql = "INSERT INTO assists (faculty_id,ta_id,course_name) VALUES (%s, %s, %s)"
        insert_val = (faculty_id,ta_id,course_name)
        mycursor.execute(insert_sql, insert_val)
        mydb.commit()
        return "Assists data inserted successfully!"
    except mysql.connector.IntegrityError:
        return "Assists data already exists for Faculty ID {} and
                TA ID {}.".format(faculty_id, ta_id)
```

```
def faculty_details(FACULTY_ID):
    mycursor.execute("SELECT * FROM faculty WHERE FACULTY_ID = %s", (FACULTY_ID,))
    result = mycursor.fetchone()
    return result

def TA_details(ta_id):
    mycursor.execute("SELECT * FROM ta WHERE ta_id = %s", (ta_id,))
    result = mycursor.fetchone()
    return result

def faculty_project_details(FACULTY_ID):
    mycursor.execute("SELECT w.faculty_id,p.project_id,p.pname,p.domain FROM faculty f,
    project p,works_on w WHERE f.FACULTY_ID=w.FACULTY_ID and w.project_id=p.project_id AND
    f.FACULTY_ID= %s", (FACULTY_ID,))
    result = mycursor.fetchone()
    return result

def faculty_cabin_details(FACULTY_ID):
    mycursor.execute("SELECT a.faculty_id,f.cabin_no,f.location,a.availability_status
    FROM faculty f,availability a
    WHERE f.FACULTY_ID=a.FACULTY_ID and f.FACULTY_ID= %s", (FACULTY_ID,))
    result = mycursor.fetchone()
    return result

def work_on(Project_id,FACULTY_ID):
    insert_sql = "INSERT INTO works_on (Project_id,FACULTY_ID) VALUES (%s, %s)"
    insert_val = (Project_id,FACULTY_ID)
    mycursor.execute(insert_sql, insert_val)
    mydb.commit()

def modify_availability_status(FACULTY_ID, availability_status):
    mycursor.execute("SELECT * FROM availability WHERE FACULTY_ID = %s", (FACULTY_ID,))
    existing_faculty = mycursor.fetchone()

    if existing_faculty:
        update_sql = "UPDATE availability SET availability_status = %s WHERE FACULTY_ID = %s"
        update_val = (availability_status, FACULTY_ID)
        mycursor.execute(update_sql, update_val)
        mydb.commit()
        return "Availability status modified successfully"
    else:
        return "Faculty ID does not exist"
```

```
@app.route('/')
def index():
    return render_template('index.html', current_user=current_user)

@app.route('/addproject', methods=['POST', 'GET'])
@login_required
def submit():
    if request.method == 'POST':
        FACULTY_ID = request.form['FACULTY_ID']
        project_id = request.form['project_id']
        pname = request.form['pname']
        domain = request.form['domain']
        message = insert_project_data(project_id, pname, domain)
        message1 = work_on(project_id,FACULTY_ID)
        return message,message1
    else:
        return render_template('addproject.html', current_user=current_user)

@app.route('/addfaculty', methods=['POST', 'GET'])
@login_required
def facultyit():
    if request.method == 'POST':
        FACULTY_ID = request.form['FACULTY_ID']
        NAME = request.form['NAME']
        CONTACT = request.form['CONTACT']
        EMAIL = request.form['EMAIL']
        DEPARTMENT = request.form['DEPARTMENT']
        cabin_no = request.form['cabin_no']
        location = request.form['location']
        availability_status='-'
        message = insert_faculty_data(FACULTY_ID, NAME, CONTACT, EMAIL, DEPARTMENT,cabin_no,
                  location)
        message1 = insert_availability(FACULTY_ID,availability_status)
        return message,message1
    else:
        return render_template('addfaculty.html', current_user=current_user)

@app.route('/addTA', methods=['POST', 'GET'])
@login_required
def TA():
    if request.method == 'POST':
        faculty_id = request.form['faculty_id']
        ta_id = request.form['ta_id']
```

```
        email = request.form['email']
        cabin_no = request.form['cabin_no']
        location = request.form['location']
        course_name = request.form['course_name']
        message = insert_TA_data(ta_id,email,cabin_no, location,course_name)
        message1 = insert_assists_data(faculty_id,ta_id,course_name)
        return message,message1
    else:
        return render_template('addTA.html', current_user=current_user)

@app.route('/triggers', methods=['GET', 'POST'])
def triggers():
    if request.method == 'POST':
        FACULTY_ID = request.form['FACULTY_ID']
        faculty_data = faculty_details(FACULTY_ID)
        if faculty_data:
            return render_template('triggers.html', faculty_data=faculty_data)
        else:
            flash('Faculty with ID {} not found.'.format(FACULTY_ID), 'danger')
            return render_template('triggers.html')
    else:
        return render_template('triggers.html', current_user=current_user)

@app.route('/fetchdetails', methods=['GET', 'POST'])
def triggers1():
    if request.method == 'POST':
        FACULTY_ID = request.form['FACULTY_ID']
        faculty_data = faculty_project_details(FACULTY_ID)
        if faculty_data:
            return render_template('triggers1.html', faculty_data=faculty_data)
        else:
            flash('Faculty with ID {} not found.'.format(FACULTY_ID), 'danger')
            return render_template('triggers1.html')
    else:
        return render_template('triggers1.html', current_user=current_user)

@app.route('/cabin', methods=['GET', 'POST'])
def triggers2():
    if request.method == 'POST':
        FACULTY_ID = request.form['FACULTY_ID']
        faculty_data = faculty_cabin_details(FACULTY_ID)
        if faculty_data:
            return render_template('triggers2.html', faculty_data=faculty_data)
```

```
        else:
            flash('Faculty with ID {} not found.'.format(FACULTY_ID), 'danger')
            return render_template('triggers2.html')
    else:
        return render_template('triggers2.html', current_user=current_user)

@app.route('/availability_satus', methods=['GET', 'POST'])
@login_required
def triggers3():
    if request.method == 'POST':
        FACULTY_ID = request.form['FACULTY_ID']
        availability_status = request.form['availability_status']
        faculty_data = modify_availability_status(FACULTY_ID, availability_status)
        return render_template('triggers3.html', faculty_data=faculty_data)
    else:
        return render_template('triggers3.html', current_user=current_user)

@app.route('/fetchTAdetails', methods=['GET', 'POST'])
def triggers4():
    if request.method == 'POST':
        ta_id = request.form['ta_id']
        faculty_data = TA_details(ta_id)
        if faculty_data:
            return render_template('triggers4.html', faculty_data=faculty_data)
        else:
            flash('TA with ID {} not found.'.format(ta_id), 'danger')
            return render_template('triggers4.html')
    else:
        return render_template('triggers4.html', current_user=current_user)

if __name__ == '__main__':
    app.run(debug=True)
```

8 Frontend

8.1 Home page

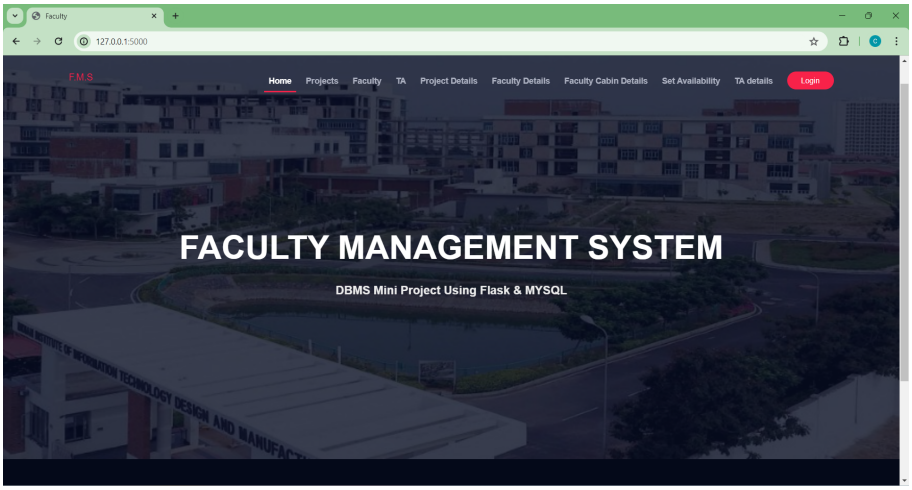

Figure 8: Home 1.

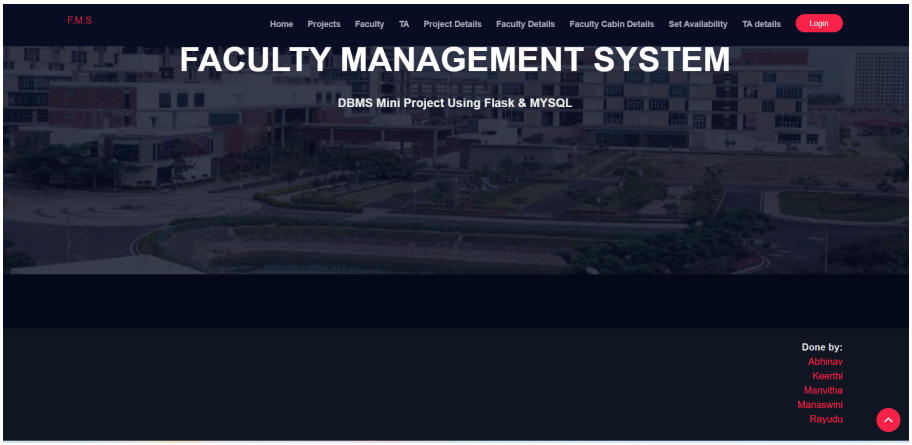

Figure 9: Home 2.

8.2 Login Interface

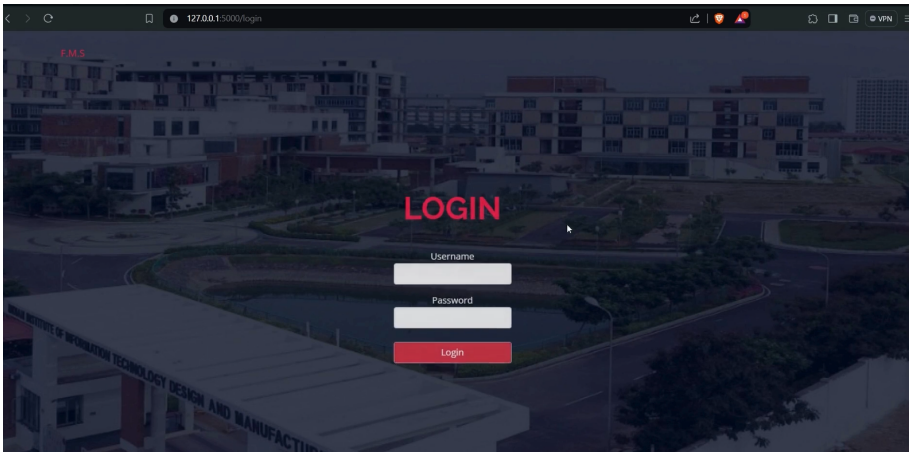

Figure 10: Login.

8.3 Faculty Data Entry

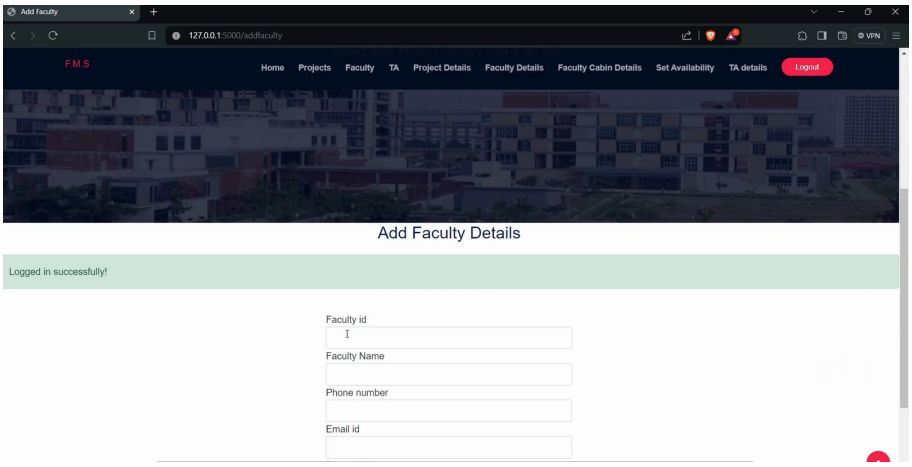

Figure 11: Faculty Data Entry.

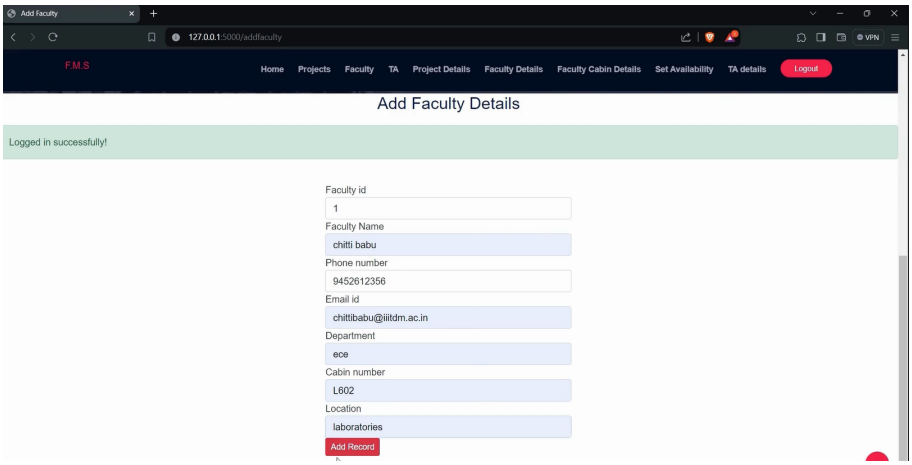

Figure 12: Faculty Data Entry.

```
MYSQL :
1)
INSERT INTO FACULTY (FACULTY_ID, NAME, CONTACT, EMAIL, DEPARTMENT, cabin_no, location)
VALUES (%s, %s, %s, %s, %s, %s, %s)
2)
INSERT INTO availability (FACULTY_ID,availability_status)
VALUES (%s,%s)
```

8.4 Project Data Entry

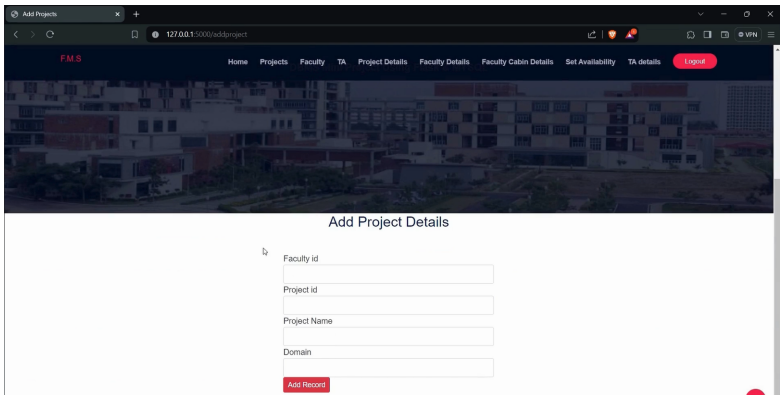

Figure 13: Project Data Entry.

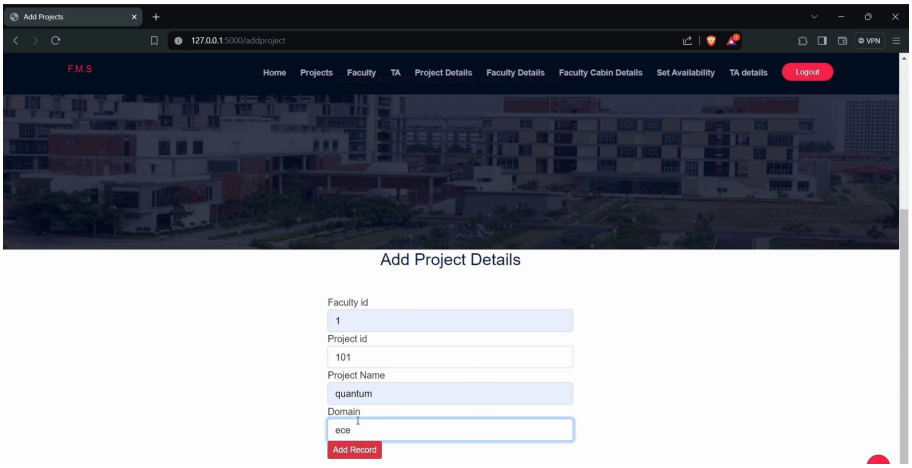

Figure 14: Project Data Entry.

```
MYSQL :
1)
INSERT INTO project (project_id, pname, domain)
VALUES (%s, %s, %s)
2)
INSERT INTO works_on (Project_id,FACULTY_ID)
VALUES (%s, %s)
```

8.5 TA Data Entry

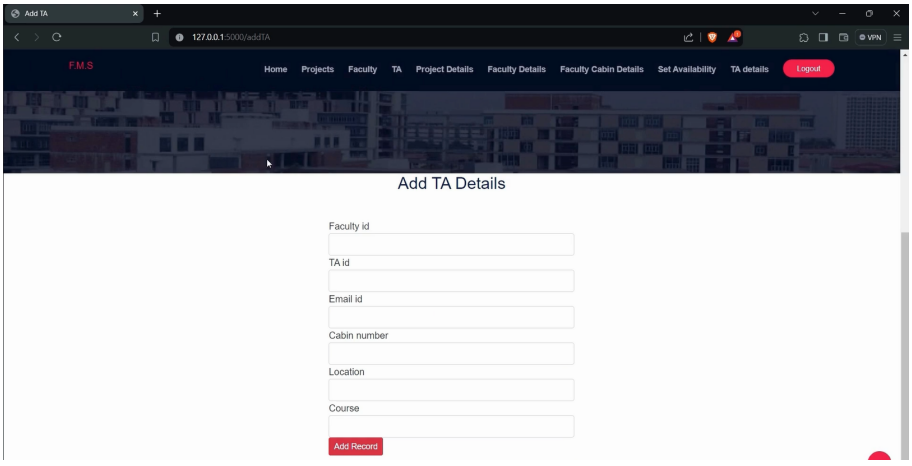

Figure 15: TA Data Entry.

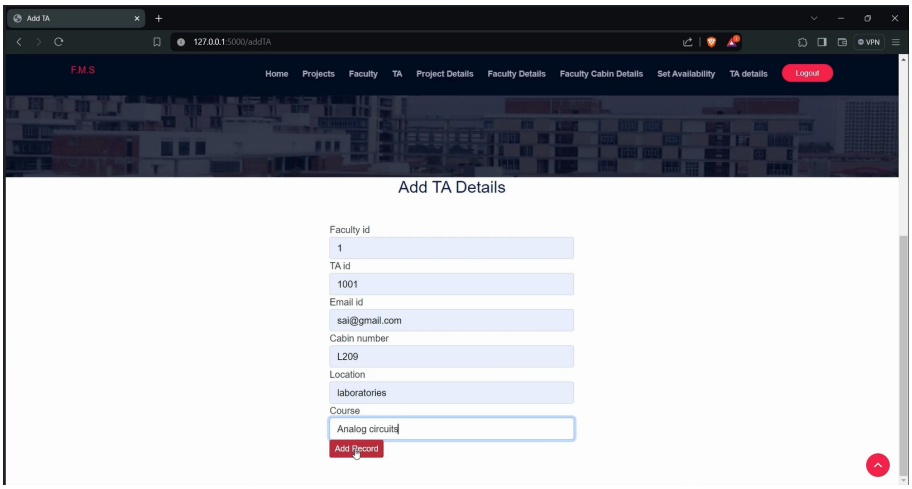

Figure 16: TA Data Entry.

```
MYSQL :
1)
INSERT INTO ta (ta_id,email,cabin_no, location,course_name)
VALUES (%s, %s, %s, %s, %s)
2)
INSERT INTO assists (faculty_id,ta_id,course_name)
VALUES (%s, %s, %s)
```

8.6 Faculty Records

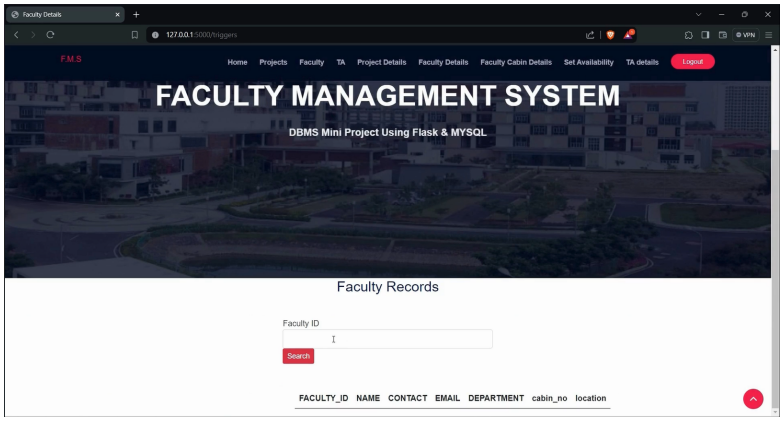

Figure 17: Faculty Records.

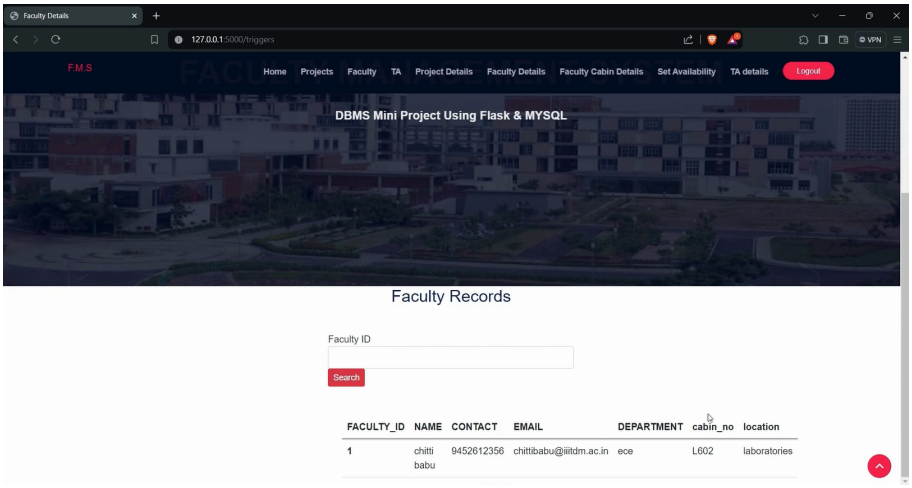

Figure 18: Faculty Records.

```
MYSQL :
SELECT *
FROM faculty
WHERE FACULTY_ID = %s
```

8.7 Faculty Project Records

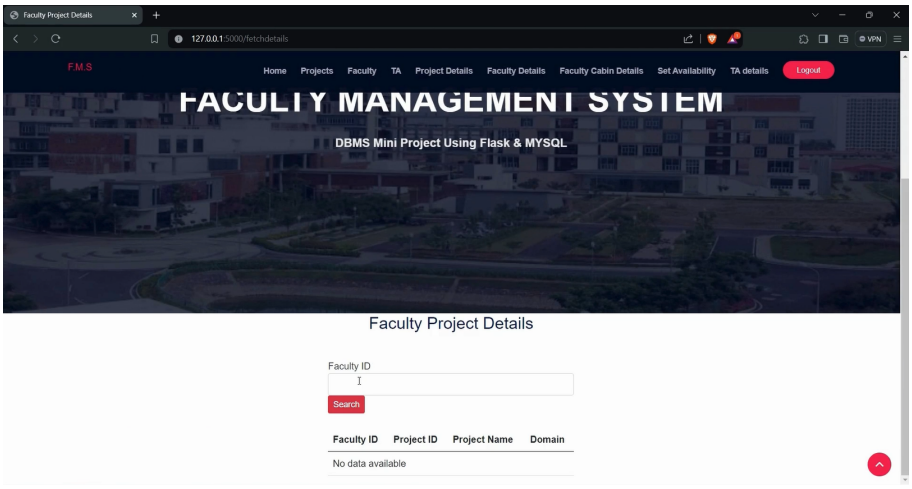

Figure 19: Faculty Project Records.

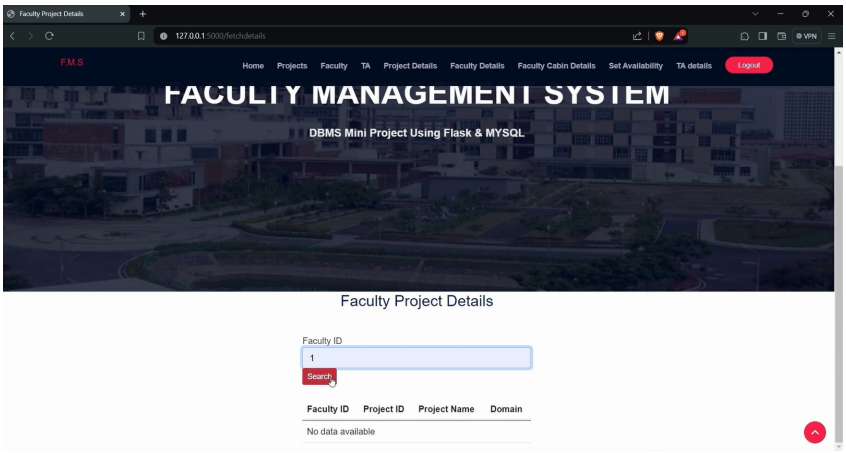

Figure 20: Faculty Project Records.

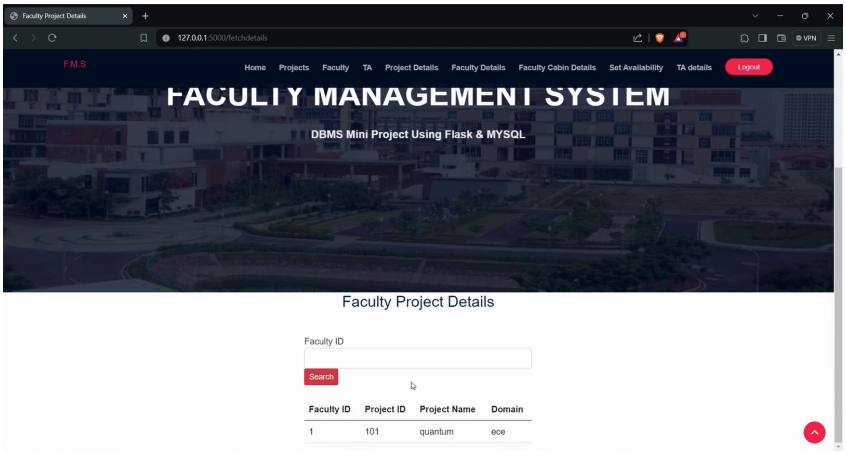

Figure 21: Faculty Project Records.

```
MYSQL :
SELECT w.faculty_id,p.project_id,p.pname,p.domain
FROM faculty f,project p,works_on w
WHERE f.FACULTY_ID=w.FACULTY_ID and w.project_id=p.project_id AND f.FACULTY_ID= %s
```

8.8 Faculty Cabin Details

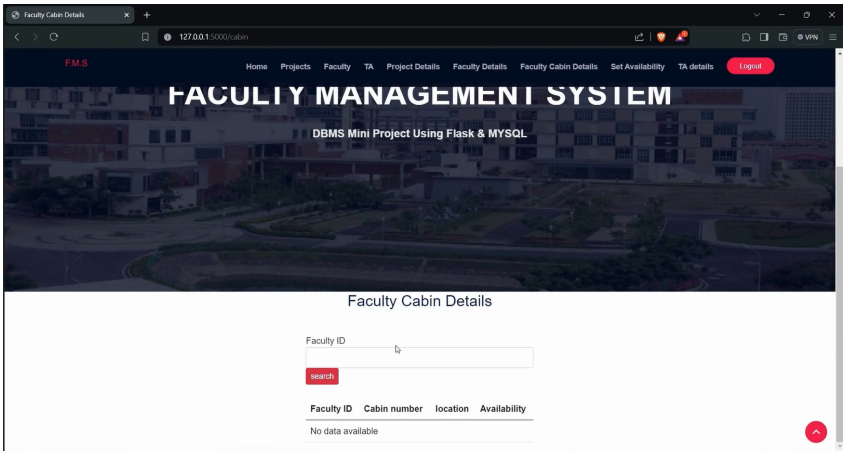

Figure 22: Faculty Cabin Details.

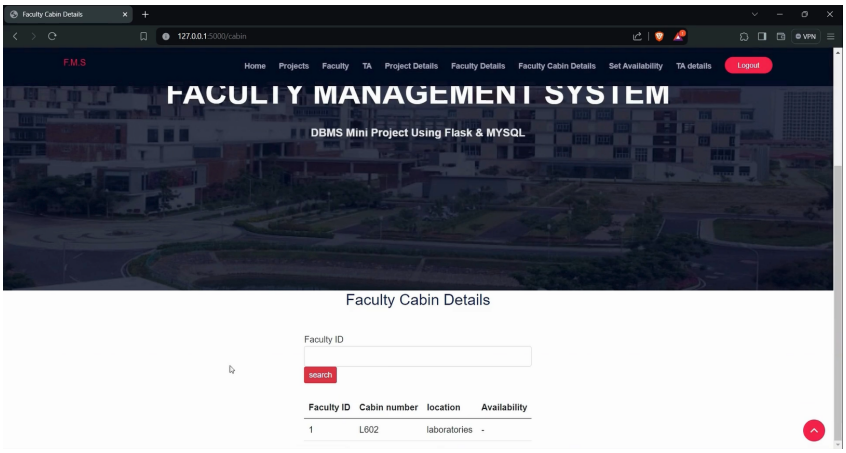

Figure 23: Faculty Cabin Details.

```
MYSQL :
SELECT a.faculty_id,f.cabin_no,f.location,a.availability_status
FROM faculty f,availability a
WHERE f.FACULTY_ID=a.FACULTY_ID and f.FACULTY_ID= %s
```

8.9 Set Availability

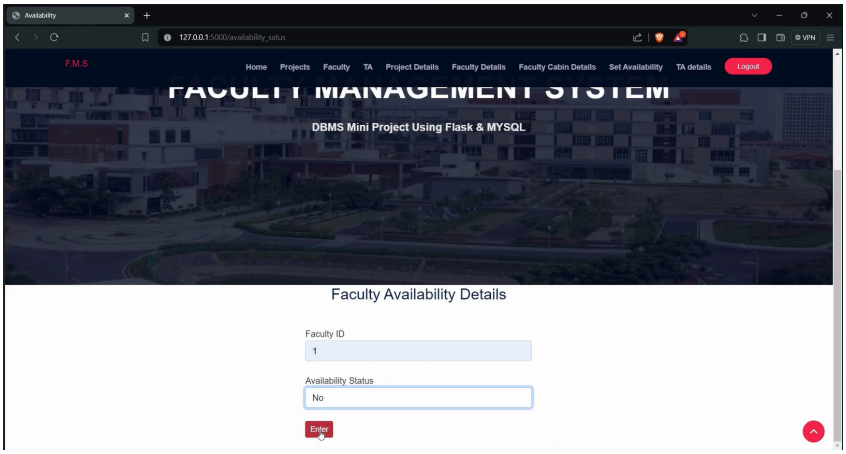

Figure 24: Setting Availability as No.

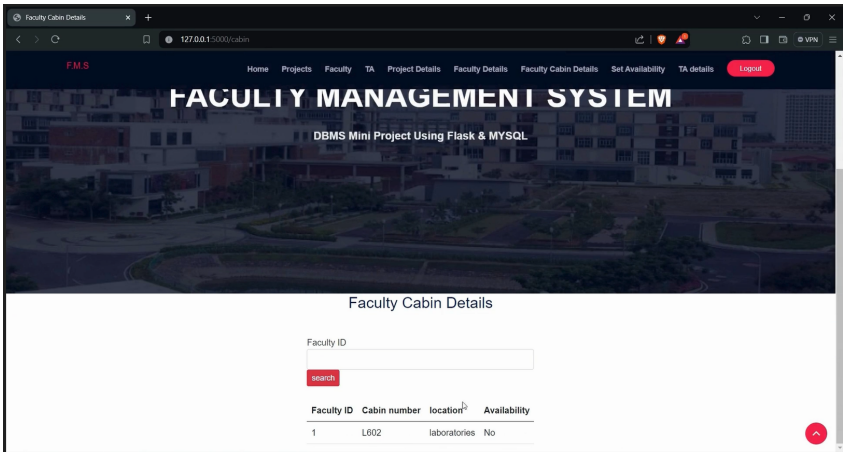

Figure 25: Faculty Cabin Details.

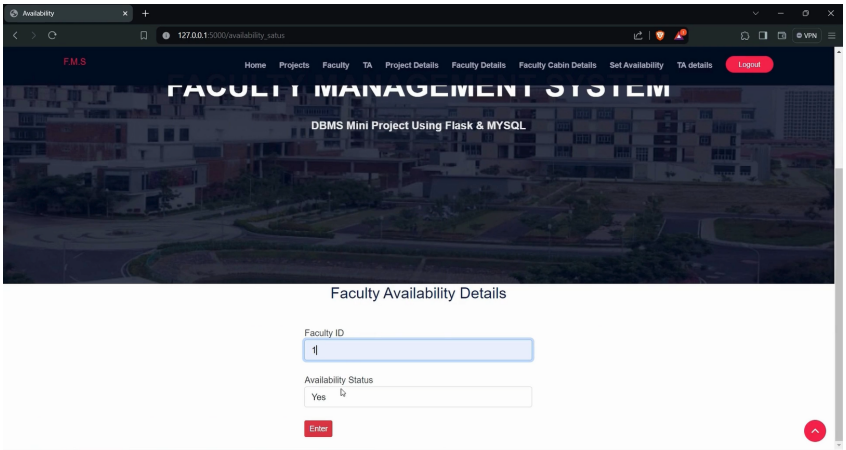

Figure 26: Setting Availability as Yes.

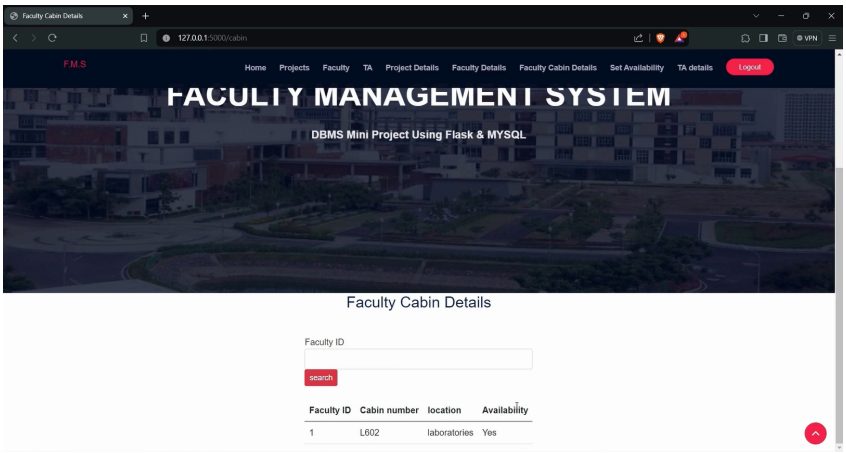

Figure 27: Faculty Cabin Details.

MYSQL :

```
UPDATE availability
SET availability_status = %s
WHERE FACULTY_ID = %s
```

8.10 TA Details

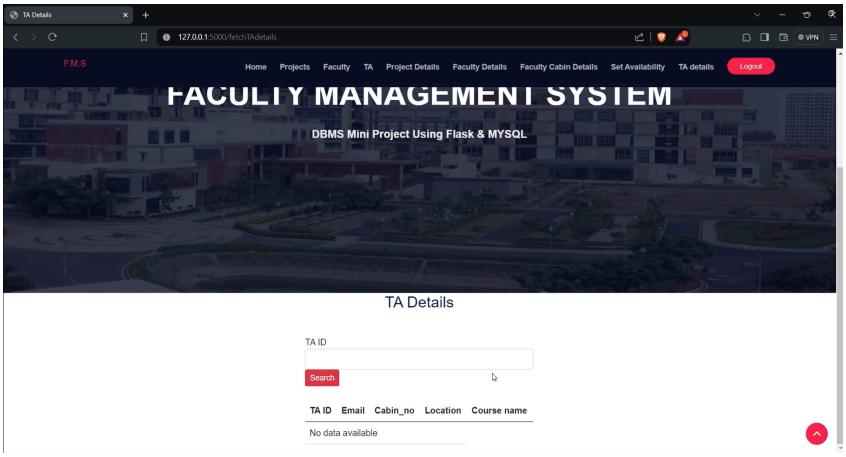

Figure 28: TA Details.

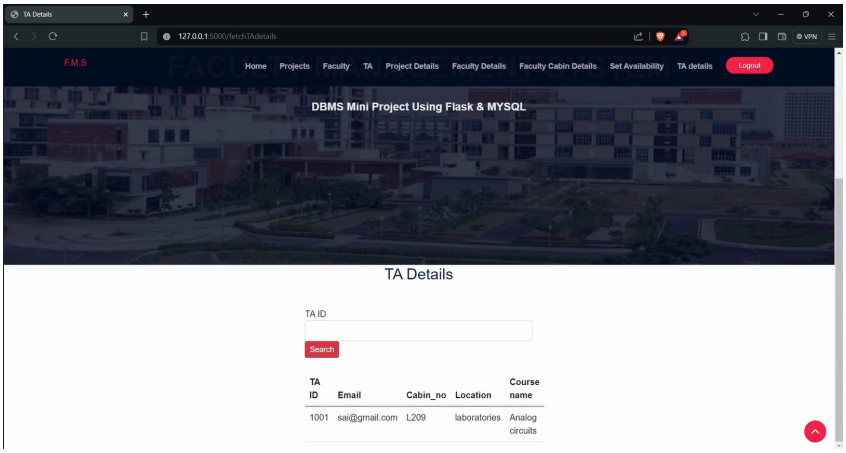

Figure 29: TA Details.

```
MYSQL :
SELECT *
FROM ta
WHERE ta_id = %s
```