UNIVERSIDAD AUTONOMA DEL ESTADO DE MEXICO
CENTRO UNIVERSITARIO UAEM ATLACOMULCO

Licenciatura en Ingeniería en Computación

Paradigmas de la Programación

Equipo:

Emmanuel Nieto Garcia

ICO 28

```python
from flask import Flask, render_template, request, redirect, url_for
import os
import database as db

template_dir = os.path.dirname(os.path.abspath(os.path.dirname(__file__)))
template_dir = r"C:\Users\emman\Downloads\FloresAmarillas\src\templates"
app = Flask(__name__, template_folder=template_dir)
```

from flask import : Imports the necessary Flask modules for creating routes, rendering HTML templates, handling HTTP requests, and managing URL redirects.

import os: Imports the os library for interacting with the operating system.

import database as db: Imports a module named database (assumed to handle database connections) as db

template_dir = ...: Sets up the directory for HTML templates.

app = Flask(...): Initializes a Flask application instance with the specified template directory (template_folder=template_dir), so it knows where to look for HTML templates

```python
#Rutas de la aplicación
@app.route('/')
def home():
    cursor = db.database.cursor()
    cursor.execute("SELECT * FROM users")
    myresult = cursor.fetchall()
    #Convertir los datos a diccionario
    insertObject = []
    columnNames = [column[0] for column in cursor.description]
    for record in myresult:
        insertObject.append(dict(zip(columnNames, record)))
    cursor.close()
    return render_template('index.html', data=insertObject)
```

@app.route('/'): Defines the route for the homepage (/).

cursor = db.database.cursor(): Creates a database cursor to execute SQL queries.

cursor.execute(...): Executes a query to select all records from the users table.

myresult = cursor.fetchall(): Fetches all records returned by the query.

insertObject = ...: Converts the database records into a list of dictionaries (each dictionary represents a row in the table).

return render_template(...): Renders the index.html template, passing insertObject (user data) to it

```python
#Ruta para guardar usuarios en la bdd
@app.route('/user', methods=['POST'])
def addUser():
    username = request.form['username']
    name = request.form['name']
    password = request.form['password']

    if username and name and password:
        cursor = db.database.cursor()
        sql = "INSERT INTO users (username, name, password) VALUES (%s, %s
        data = (username, name, password)
        cursor.execute(sql, data)
        db.database.commit()
    return redirect(url_for('home'))
```

@app.route('/user', methods=['POST']): Defines a route to handle user data submissions via the POST method.

request.form['...']: Retrieves username, name, and password values from the submitted form.

if username and name and password: Checks that all required fields are filled.

cursor.execute(...): Inserts the new user data into the users table.

db.database.commit(): Saves changes to the database.

redirect(url_for('home')): Redirects the user back to the homepage after adding a user.

```python
@app.route('/delete/<string:id>')
def delete(id):
    cursor = db.database.cursor()
    sql = "DELETE FROM users WHERE id=%s"
    data = (id,)
    cursor.execute(sql, data)
    db.database.commit()
    return redirect(url_for('home'))
```

@app.route('/delete/<string:id>'): Defines a route to delete a user based on a unique ID.

sql = "DELETE FROM users WHERE id=%s": SQL statement to delete a record where the id matches.

return redirect(url_for('home')): Redirects back to the homepage after deletion.

```python
@app.route('/edit/<string:id>', methods=['POST'])
def edit(id):
    username = request.form['username']
    name = request.form['name']
    password = request.form['password']

    if username and name and password:
        cursor = db.database.cursor()
        sql = "UPDATE users SET username = %s, name = %s, password = %s
        data = (username, name, password, id)
        cursor.execute(sql, data)
        db.database.commit()
    return redirect(url_for('home'))

if __name__ == '__main__':
    app.run(debug=True, port=4000)
```

@app.route('/edit/<string:id>', methods=['POST']): Defines a route to update a user's information based on the provided id.
sql = "UPDATE users SET ... WHERE id = %s": SQL statement to update fields (username, name, password) for a user with the matching id.

return redirect(url_for('home')): Redirects back to the homepage after the update.

if __name__ == '__main__':: Checks if the script is being run directly.

app.run(...): Starts the Flask application in debug mode on port 4000, so errors are displayed in the browser for easier debugging

```python
import mysql.connector

database = mysql.connector.connect(
    host='localhost',
    user='root',
    password='',
    database='base'
)
```

host='localhost': Specifies the server where the database is hosted. localhost means it's on the same machine as the code.

user='root': The username to access the database. The default username for MySQL is usually root.

password='': The password for the specified user. Here it's empty, but you should provide a password for security if one is set.

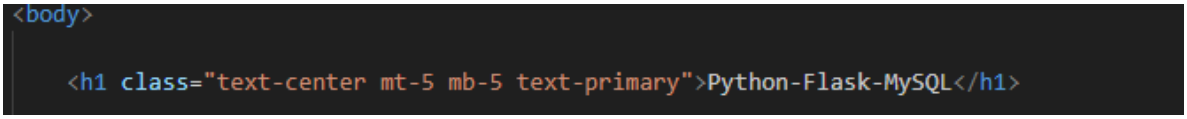
database='base': The name of the database you want to connect to. This should be a database that already exists on the MySQL server

The <head> section includes:

Character encoding and compatibility settings.

The viewport settings for responsive design.

Bootstrap CSS and JavaScript for styling and interactivity



The main title of the page is styled using Bootstrap classes for centering and color

A Bootstrap card is used to create a styled container.

A form is created with fields for username, name, and password.

The form submits a POST request to the /user endpoint when the "Save" button is clicked

```
<table class="table table-bordered">
    <thead>
        <tr>
            <th scope="col">#</th>
            <th scope="col">Username</th>
            <th scope="col">Name</th>
            <th scope="col">Password</th>
            <th scope="col">Edit</th>
            <th scope="col">Delete</th>
        </tr>
    </thead>

<tbody>
```

The table displays existing users with columns for ID, username, name, password, and actions (Edit/Delete).

The {% for d in data %} loop dynamically generates rows for each user passed to the template, using Jinja2 syntax.

```
<div class="modal fade" id="modal{{d.id}}" tabindex="-1" aria-labelledby="exampleModalLabe
    <div class="modal-dialog">
      <div class="modal-content">
        <div class="modal-header">
          <h1 class="modal-title fs-5" id="exampleModalLabel">{{d.username}}</h1>
          <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Clos
        </div>
        <div class="modal-body">
            <form action="/edit/{{d.id}}" method="post">
                <label>Username</label>
                <input type="text" class="form-control mb-3" name="username" value="{{d.us
                <label>Name</label>
                <input type="text" class="form-control mb-3" name="name" value="{{d.name}}
                <label>Password</label>
                <input type="text" class="form-control mb-3" name="password" value="{{d.pa

        </div>
        <div class="modal-footer">
          <button type="submit" class="btn btn-primary">Save changes</button>
        </div>
        </form>
      </div>
    </div>
  </div>
```

Each user has an associated modal for editing their details.

The modal is triggered by the Edit button and contains a form pre-filled with the user's current data.

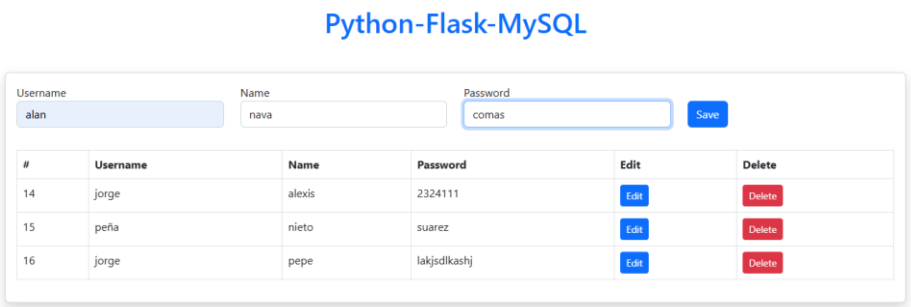
Submitting the form sends a POST request to /edit/{{d.id}} to update the user.

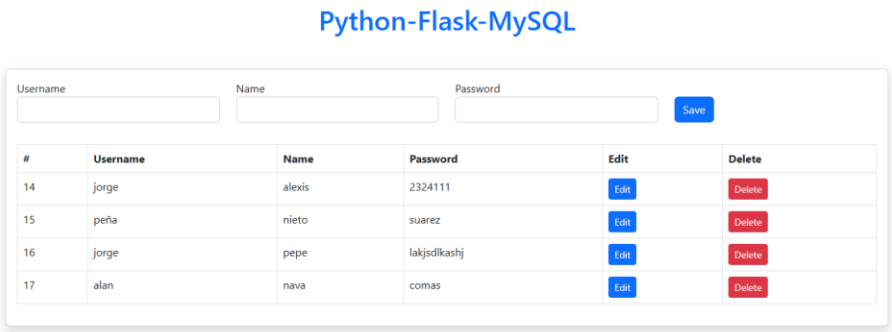
The table and container are closed, along with the body and HTML tags.

The template ends the loop for displaying users and finalizes the document structure

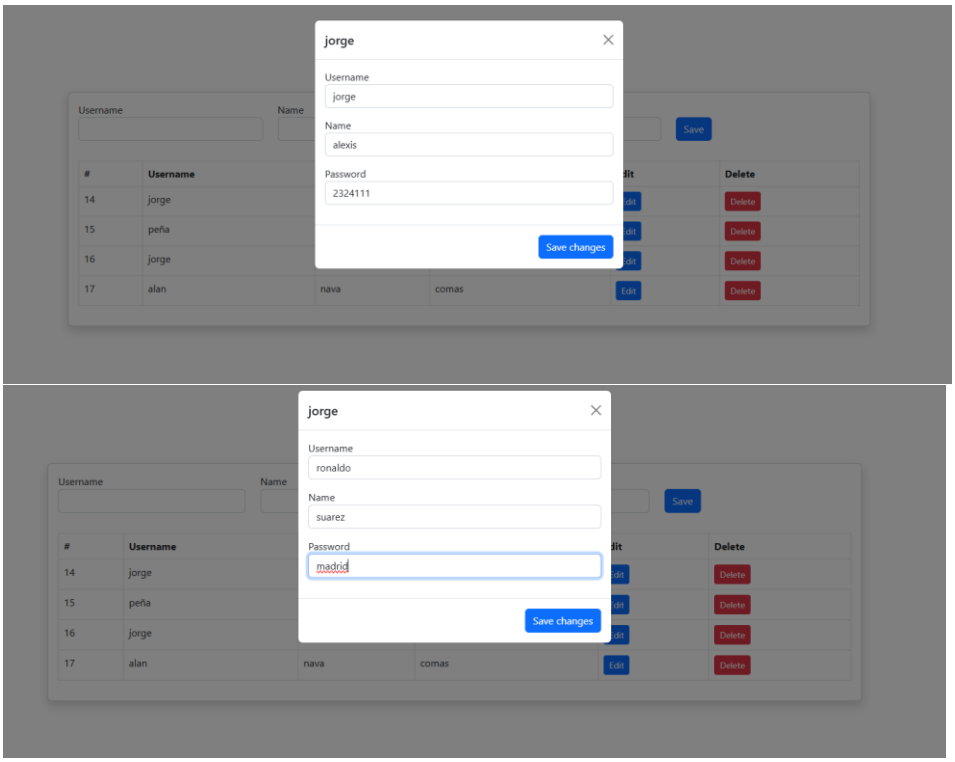
Functionality

At the beginning, we can add user data to the table, and it will be displayed there.

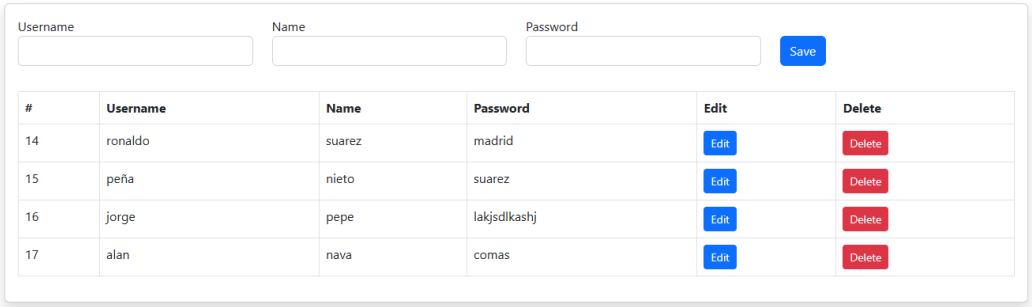


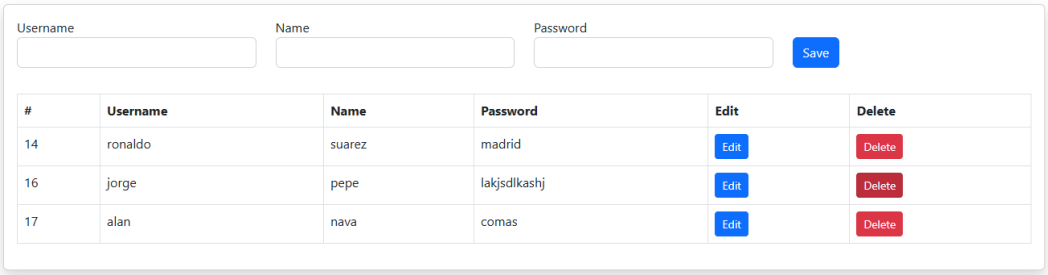


We can also edit it



And we can delete it.





The records in the table can be both deleted and edited.

Repositorio:

https://github.com/emmanig/Flask-con-Mysql1.git