Proyecto: Web App de Help Desk con Flask, MariaDB, Bootstrap y jQuery

Objetivos

- Diseñar un modelo de base de datos de 3 tablas para un sistema de Help Desk.
- Implementar una Web App full stack utilizando Flask, MariaDB, HTML, Bootstrap y jQuery.
- Aplicar conceptos básicos de autenticación, autorización y seguridad en aplicaciones web.
- Integrar la capa de presentación (Bootstrap + jQuery) con la capa de servicios (Flask) y la base de datos.
- Documentar el sistema (manuales, diagramas, capturas de pantalla) y publicar el proyecto en un repositorio de GitHub.

1. Descripción general del sistema

Desarrollaremos un **Sistema de Soporte / Help Desk** con:

Tablas principales (3)

1. **users**
2. **tickets**
3. **ticket_comments**

Roles

- **Admin:** administrar usuarios, puede ver todos los tickets.
- **Agent:** atender tickets, cambiar estado, añadir comentarios.
- **User (cliente interno):** crear tickets, ver sus tickets y comentarios.

Funcionalidades mínimas

- Registro inicial de un usuario administrador (puede ser creado por SQL).
- Inicio/cierre de sesión.
- Crear ticket de soporte.
- Ver listado de tickets (filtrado según el rol).
- Ver detalle del ticket y su historial de comentarios.
- Añadir comentarios a un ticket.
- Cambiar estado y prioridad del ticket.
- Panel de administración de usuarios (solo admin).
- Vistas responsivas con Bootstrap.
- Algo de interacción con jQuery (por ejemplo, cambiar estado vía formulario sin recargar toda la página o mejoras de UX).

Arquitectura (en palabras):

> Navegador (HTML + Bootstrap + jQuery) <-->Flask (Python: rutas, lógica, seguridad)<-->MariaDB (datos persistentes)

2. Diseño de la base de datos (MariaDB)

2.1. Modelo lógico

Tabla users

- id (INT, PK, AUTO_INCREMENT)
- name (VARCHAR(100))
- email (VARCHAR(150), UNIQUE)
- password_hash (VARCHAR(255))
- role (ENUM(‘ADMIN’, ‘AGENT’, ‘USER’))
- created_at (DATETIME)

Tabla tickets

- id (INT, PK, AUTO_INCREMENT)
- title (VARCHAR(200))
- description (TEXT)
- status (ENUM('OPEN', 'IN_PROGRESS', 'RESOLVED'))
- priority (ENUM('LOW', 'MEDIUM', 'HIGH'))
- created_at (DATETIME)
- updated_at (DATETIME)
- created_by (INT, FK → users.id)
- assigned_to (INT, FK → users.id, NULL permitido)

Tabla ticket_comments

- id (INT, PK, AUTO_INCREMENT)
- ticket_id (INT, FK → tickets.id)
- user_id (INT, FK → users.id)
- comment (TEXT)
- created_at (DATETIME)

2.2. Script SQL de creación (MariaDB)

```
CREATE DATABASE helpdesk_db CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
USE helpdesk_db;
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(150) NOT NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    role ENUM('ADMIN', 'AGENT', 'USER') NOT NULL DEFAULT 'USER',
    created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP
);



CREATE TABLE tickets (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
```

```
    description TEXT NOT NULL,
    status ENUM('OPEN', 'IN_PROGRESS', 'RESOLVED') NOT NULL DEFAULT 'OPEN',
    priority ENUM('LOW', 'MEDIUM', 'HIGH') NOT NULL DEFAULT 'MEDIUM',
    created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    created_by INT NOT NULL,
    assigned_to INT NULL,
    CONSTRAINT fk_tickets_created_by FOREIGN KEY (created_by) REFERENCES users(id),
    CONSTRAINT fk_tickets_assigned_to FOREIGN KEY (assigned_to) REFERENCES users(id)
);

CREATE TABLE ticket_comments (
    id INT AUTO_INCREMENT PRIMARY KEY,
    ticket_id INT NOT NULL,
    user_id INT NOT NULL,
    comment TEXT NOT NULL,
    created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_comments_ticket FOREIGN KEY (ticket_id) REFERENCES tickets(id),
    CONSTRAINT fk_comments_user FOREIGN KEY (user_id) REFERENCES users(id)
);
```

Nota: Crear manualmente un admin al principio:

```
INSERT INTO users (name, email, password_hash, role)
VALUES ('Admin', 'admin@example.com', 'PASSWORD_HASH_AQUI', 'ADMIN');
```

El **password_hash** lo generaremos con Flask y luego lo pegamos aquí (más adelante lo vemos).

3. Preparar el entorno

3.1. Requerimientos

- Python 3.x
- MariaDB server
- Git
- Navegador web moderno
- Editor de código (VS Code, PyCharm, etc.)

3.2. Crear entorno virtual e instalar librerías

En el folder o la carpeta de tu proyecto:

```
python -m venv venv
source venv/bin/activate   # en macOS / Linux
venv\Scripts\activate      # en Windows

pip install flask pymysql python-dotenv Werkzeug
```

Crea un archivo requirements.txt:

```
Flask
pymysql
python-dotenv
Werkzeug
```

4. Estructura del proyecto

Ejemplo:

```
helpdesk_app/
├─ app.py
├─ config.py
├─ requirements.txt
├─ .env                # variables de entorno (opcional, recomendado)
├─ templates/
│   ├─ base.html
│   ├─ login.html
│   ├─ dashboard.html
│   ├─ tickets_list.html
│   ├─ ticket_detail.html
│   ├─ ticket_new.html
│   ├─ users_list.html
├─ static/
│   ├─ css/
│   │   └─ custom.css
│   └─ js/
│       └─ main.js
└─ docs/
    ├─ manual_usuario.md
    ├─ manual_tecnico.md
    ├─ er_diagram.png
    └─ screenshots/
        ├─ login.png
        ├─ dashboard.png
        └─ ...
```

** **IMPORTANTE REGLA PARA ENTREGA DEL PROYECTO:** toda la documentación (manuales, diagramas, capturas de pantalla) debe estar en el folder o la carpeta docs/ y subir a GitHub.

5. Configuración de Flask y conexión a MariaDB

5.1. Archivo config.py

```
import os
from dotenv import load_dotenv

load_dotenv()

class Config:
    SECRET_KEY = os.getenv("SECRET_KEY", "dev-secret-key-change-this")
    DB_HOST = os.getenv("DB_HOST", "localhost")
    DB_USER = os.getenv("DB_USER", "helpdesk_user")
    DB_PASSWORD = os.getenv("DB_PASSWORD", "helpdesk_password")
    DB_NAME = os.getenv("DB_NAME", "helpdesk_db")
```

5.2. Archivo .env (no se requiere subir a GitHub, pero es muy importante en el proyecto)

```
SECRET_KEY=pon_aqui_una_clave_larga_y_segura
DB_HOST=localhost
DB_USER=helpdesk_user
DB_PASSWORD=helpdesk_password
DB_NAME=helpdesk_db
```

5.3. Archivo app.py (estructura básica)

```
from flask import Flask, render_template, request, redirect, url_for, flash, session
import pymysql
from werkzeug.security import generate_password_hash, check_password_hash
from config import Config

app = Flask(__name__)
app.config.from_object(Config)


def get_db_connection():
```

```
    return pymysql.connect(
        host=app.config["DB_HOST"],
        user=app.config["DB_USER"],
        password=app.config["DB_PASSWORD"],
        database=app.config["DB_NAME"],
        cursorclass=pymysql.cursors.DictCursor
    )


@app.route("/")
def index():
    # If user is logged in, redirect to dashboard
    if "user_id" in session:
        return redirect(url_for("dashboard"))
    return redirect(url_for("login"))


@app.route("/dashboard")
def dashboard():
    if "user_id" not in session:
        return redirect(url_for("login"))
    return render_template("dashboard.html")


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

6. Autenticación y roles

6.1. Helpers: login_required y role_required

Añade en app.py:

```
from functools import wraps

def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
```

```
        if "user_id" not in session:
            flash("You must be logged in to access this page.", "warning")
            return redirect(url_for("login"))
        return f(*args, **kwargs)
    return decorated_function


def role_required(*roles):
    def decorator(f):
        @wraps(f)
        def decorated_function(*args, **kwargs):
            if "user_role" not in session or session["user_role"] not in roles:
                flash("You do not have permission to access this page.", "danger")
                return redirect(url_for("dashboard"))
            return f(*args, **kwargs)
        return decorated_function
    return decorator
```

6.2. Ruta de login

Template templates/login.html (Bootstrap + jQuery mínimo)

```
{% extends "base.html" %}
{% block content %}
<div class="container mt-5">
  <div class="row justify-content-center">
    <div class="col-md-4">
      <h3 class="text-center mb-4">Help Desk Login</h3>
      <form method="post" action="{{ url_for('login') }}">
        <div class="mb-3">
          <label for="email" class="form-label">Email</label>
          <input id="email" name="email" type="email" class="form-control" required />
        </div>
        <div class="mb-3">
          <label for="password" class="form-label">Password</label>
          <input id="password" name="password" type="password" class="form-control" required
/>
        </div>
        <button type="submit" class="btn btn-primary w-100">Login</button>
      </form>
```

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

Base template templates/base.html (Bootstrap + navbar básico)

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Help Desk</title>
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet" />
  <link rel="stylesheet" href="{{ url_for('static', filename='css/custom.css') }}">
</head>
<body>
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class="container-fluid">
      <a class="navbar-brand" href="{{ url_for('dashboard') }}">Help Desk</a>
      <div class="collapse navbar-collapse">
        <ul class="navbar-nav ms-auto">
          {% if session.get("user_id") %}
            <li class="nav-item">
              <a class="nav-link" href="{{ url_for('tickets_list') }}">Tickets</a>
            </li>
            {% if session.get("user_role") == "ADMIN" %}
            <li class="nav-item">
              <a class="nav-link" href="{{ url_for('users_list') }}">Users</a>
            </li>
            {% endif %}
            <li class="nav-item">
              <a class="nav-link" href="{{ url_for('logout') }}">Logout</a>
            </li>
          {% endif %}
        </ul>
      </div>
    </div>
```

```
  </nav>

  <div class="container mt-3">
    {% with messages = get_flashed_messages(with_categories=true) %}
      {% if messages %}
        {% for category, msg in messages %}
          <div class="alert alert-{{ category }} alert-dismissible fade show" role="alert">
            {{ msg }}
            <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
          </div>
        {% endfor %}
      {% endif %}
    {% endwith %}
  </div>

  {% block content %}{% endblock %}

  <!-- jQuery & Bootstrap JS -->
  <script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
  <script src="{{ url_for('static', filename='js/main.js') }}"></script>
</body>
</html>
```

Lógica de login/logout en app.py

```
@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        email = request.form.get("email")
        password = request.form.get("password")

        conn = get_db_connection()
        with conn.cursor() as cursor:
            cursor.execute("SELECT * FROM users WHERE email = %s", (email,))
            user = cursor.fetchone()
        conn.close()

        if user and check_password_hash(user["password_hash"], password):
```

```
            # Store user info in session
            session["user_id"] = user["id"]
            session["user_name"] = user["name"]
            session["user_role"] = user["role"]
            flash("Welcome, {}!".format(user["name"]), "success")
            return redirect(url_for("dashboard"))
        else:
            flash("Invalid email or password.", "danger")

    return render_template("login.html")


@app.route("/logout")
@login_required
def logout():
    session.clear()
    flash("You have been logged out.", "info")
    return redirect(url_for("login"))
```

Actividad sugerida luego de crear la base de datos y tener Python configurado. Crear un script Python rápido para generar un password_hash y lo usen para el admin inicial.

Ejemplo:

```
from werkzeug.security import generate_password_hash
print(generate_password_hash("admin123"))
```

Copiar ese hash en el INSERT SQL del admin (está en el SQL script del inicio).

7. Gestión de tickets

7.1. Lista de tickets

Ruta en app.py

```
@app.route("/tickets")
@login_required
def tickets_list():
    user_id = session["user_id"]
    user_role = session["user_role"]
```

```
    conn = get_db_connection()
    with conn.cursor() as cursor:
        if user_role == "ADMIN":
            cursor.execute("""
                SELECT t.*, u.name AS created_by_name, a.name AS assigned_to_name
                FROM tickets t
                JOIN users u ON t.created_by = u.id
                LEFT JOIN users a ON t.assigned_to = a.id
                ORDER BY t.created_at DESC
            """)
        elif user_role == "AGENT":
            cursor.execute("""
                SELECT t.*, u.name AS created_by_name, a.name AS assigned_to_name
                FROM tickets t
                JOIN users u ON t.created_by = u.id
                LEFT JOIN users a ON t.assigned_to = a.id
                WHERE t.assigned_to = %s OR t.assigned_to IS NULL
                ORDER BY t.created_at DESC
            """, (user_id,))
        else:  # USER
            cursor.execute("""
                SELECT t.*, u.name AS created_by_name, a.name AS assigned_to_name
                FROM tickets t
                JOIN users u ON t.created_by = u.id
                LEFT JOIN users a ON t.assigned_to = a.id
                WHERE t.created_by = %s
                ORDER BY t.created_at DESC
            """, (user_id,))

        tickets = cursor.fetchall()
    conn.close()

    return render_template("tickets_list.html", tickets=tickets)
```

Template templates/tickets_list.html

```
{% extends "base.html" %}
{% block content %}
<div class="container mt-4">
```

```
  <div class="d-flex justify-content-between align-items-center mb-3">
    <h3>Tickets</h3>
    <a href="{{ url_for('ticket_new') }}" class="btn btn-success">New Ticket</a>
  </div>

  <table class="table table-striped table-hover">
    <thead>
      <tr>
        <th>#</th>
        <th>Title</th>
        <th>Status</th>
        <th>Priority</th>
        <th>Created by</th>
        <th>Assigned to</th>
        <th>Created at</th>
      </tr>
    </thead>
    <tbody>
      {% for t in tickets %}
      <tr>
        <td><a href="{{ url_for('ticket_detail', ticket_id=t.id) }}">{{ t.id }}</a></td>
        <td>{{ t.title }}</td>
        <td><span class="badge bg-secondary">{{ t.status }}</span></td>
        <td><span class="badge bg-info">{{ t.priority }}</span></td>
        <td>{{ t.created_by_name }}</td>
        <td>{{ t.assigned_to_name or '-' }}</td>
        <td>{{ t.created_at }}</td>
      </tr>
      {% endfor %}
    </tbody>
  </table>
</div>
{% endblock %}
```

7.2. Crear un nuevo ticket

Ruta en app.py

```
@app.route("/tickets/new", methods=["GET", "POST"])
@login_required
```

```
def ticket_new():
    if request.method == "POST":
        title = request.form.get("title")
        description = request.form.get("description")
        priority = request.form.get("priority")
        created_by = session["user_id"]

        if not title or not description:
            flash("Title and description are required.", "warning")
            return redirect(url_for("ticket_new"))

        conn = get_db_connection()
        with conn.cursor() as cursor:
            cursor.execute("""
                INSERT INTO tickets (title, description, priority, created_by)
                VALUES (%s, %s, %s, %s)
            """, (title, description, priority, created_by))
        conn.commit()
        conn.close()

        flash("Ticket created successfully.", "success")
        return redirect(url_for("tickets_list"))

    return render_template("ticket_new.html")
```

Template templates/ticket_new.html

```
{% extends "base.html" %}
{% block content %}
<div class="container mt-4">
  <h3>New Ticket</h3>
  <form method="post">
    <div class="mb-3">
      <label class="form-label">Title</label>
      <input name="title" type="text" class="form-control" required />
    </div>
    <div class="mb-3">
      <label class="form-label">Description</label>
      <textarea name="description" class="form-control" rows="5" required></textarea>
    </div>
```

```
    <div class="mb-3">
      <label class="form-label">Priority</label>
      <select name="priority" class="form-select">
        <option value="LOW">LOW</option>
        <option value="MEDIUM" selected>MEDIUM</option>
        <option value="HIGH">HIGH</option>
      </select>
    </div>
    <button type="submit" class="btn btn-primary">Create</button>
    <a href="{{ url_for('tickets_list') }}" class="btn btn-secondary">Cancel</a>
  </form>
</div>
{% endblock %}
```

8. Detalle del ticket y comentarios

8.1. Ver detalle + cambiar estado/asignar

```
@app.route("/tickets/<int:ticket_id>", methods=["GET", "POST"])
@login_required
def ticket_detail(ticket_id):
    conn = get_db_connection()
    with conn.cursor() as cursor:
        cursor.execute("""
            SELECT t.*, u.name AS created_by_name, a.name AS assigned_to_name
            FROM tickets t
            JOIN users u ON t.created_by = u.id
            LEFT JOIN users a ON t.assigned_to = a.id
            WHERE t.id = %s
        """, (ticket_id,))
        ticket = cursor.fetchone()

        cursor.execute("""
            SELECT c.*, u.name AS user_name
            FROM ticket_comments c
            JOIN users u ON c.user_id = u.id
            WHERE c.ticket_id = %s
            ORDER BY c.created_at ASC
        """, (ticket_id,))
```

```
        comments = cursor.fetchall()

        cursor.execute("SELECT id, name FROM users WHERE role IN ('ADMIN', 'AGENT')")
        agents = cursor.fetchall()
    conn.close()

    if not ticket:
        flash("Ticket not found.", "danger")
        return redirect(url_for("tickets_list"))

    return render_template("ticket_detail.html",
                           ticket=ticket,
                           comments=comments,
                           agents=agents)
```

8.2. Template ticket_detail.html (con jQuery para mejorar UX si quieres)

```
{% extends "base.html" %}
{% block content %}
<div class="container mt-4">
  <div class="d-flex justify-content-between align-items-center mb-3">
    <h3>Ticket #{{ ticket.id }} - {{ ticket.title }}</h3>
    <a href="{{ url_for('tickets_list') }}" class="btn btn-secondary">Back</a>
  </div>

  <div class="card mb-3">
    <div class="card-body">
      <p><strong>Status:</strong> <span class="badge bg-secondary">{{ ticket.status
}}</span></p>
      <p><strong>Priority:</strong> <span class="badge bg-info">{{ ticket.priority
}}</span></p>
      <p><strong>Created by:</strong> {{ ticket.created_by_name }}</p>
      <p><strong>Assigned to:</strong> {{ ticket.assigned_to_name or "Unassigned" }}</p>
      <p><strong>Description:</strong></p>
      <p>{{ ticket.description }}</p>
    </div>
  </div>

  <!-- Form to update status and assign -->
  {% if session.get("user_role") in ["ADMIN", "AGENT"] %}
```

```
  <div class="card mb-3">
    <div class="card-body">
      <form method="post" action="{{ url_for('ticket_update', ticket_id=ticket.id) }}">
        <div class="row">
          <div class="col-md-4">
            <label class="form-label">Status</label>
            <select name="status" class="form-select">
              <option value="OPEN" {% if ticket.status == "OPEN" %}selected{% endif
%}>OPEN</option>
              <option value="IN_PROGRESS" {% if ticket.status == "IN_PROGRESS" %}selected{%
endif %}>IN_PROGRESS</option>
              <option value="RESOLVED" {% if ticket.status == "RESOLVED" %}selected{% endif
%}>RESOLVED</option>
            </select>
          </div>
          <div class="col-md-4">
            <label class="form-label">Assigned to</label>
            <select name="assigned_to" class="form-select">
              <option value="">Unassigned</option>
              {% for a in agents %}
              <option value="{{ a.id }}" {% if ticket.assigned_to == a.id %}selected{% endif
%}>{{ a.name }}</option>
              {% endfor %}
            </select>
          </div>
          <div class="col-md-4 d-flex align-items-end">
            <button type="submit" class="btn btn-primary w-100">Update</button>
          </div>
        </div>
      </form>
    </div>
  </div>
  {% endif %}

  <!-- Comments -->
  <h5>Comments</h5>
  <ul class="list-group mb-3">
    {% for c in comments %}
    <li class="list-group-item">
```

```
      <div class="d-flex justify-content-between">
        <strong>{{ c.user_name }}</strong>
        <small>{{ c.created_at }}</small>
      </div>
      <p class="mb-0">{{ c.comment }}</p>
    </li>
    {% else %}
    <li class="list-group-item"><em>No comments yet.</em></li>
    {% endfor %}
  </ul>

  <!-- Add comment -->
  <div class="card">
    <div class="card-body">
      <form method="post" action="{{ url_for('comment_add', ticket_id=ticket.id) }}">
        <div class="mb-3">
          <label class="form-label">Add Comment</label>
          <textarea name="comment" class="form-control" rows="3" required></textarea>
        </div>
        <button type="submit" class="btn btn-primary">Add Comment</button>
      </form>
    </div>
  </div>
</div>
{% endblock %}
```

8.3. Actualizar ticket y añadir comentarios

```
@app.route("/tickets/<int:ticket_id>/update", methods=["POST"])
@login_required
def ticket_update(ticket_id):
    user_role = session["user_role"]
    if user_role not in ["ADMIN", "AGENT"]:
        flash("You are not allowed to update tickets.", "danger")
        return redirect(url_for("ticket_detail", ticket_id=ticket_id))

    status = request.form.get("status")
    assigned_to = request.form.get("assigned_to") or None

    conn = get_db_connection()
```

```
    with conn.cursor() as cursor:
        cursor.execute("""
            UPDATE tickets
            SET status = %s, assigned_to = %s
            WHERE id = %s
        """, (status, assigned_to, ticket_id))
    conn.commit()
    conn.close()

    flash("Ticket updated.", "success")
    return redirect(url_for("ticket_detail", ticket_id=ticket_id))


@app.route("/tickets/<int:ticket_id>/comments", methods=["POST"])
@login_required
def comment_add(ticket_id):
    comment_text = request.form.get("comment")
    user_id = session["user_id"]

    if not comment_text:
        flash("Comment cannot be empty.", "warning")
        return redirect(url_for("ticket_detail", ticket_id=ticket_id))

    conn = get_db_connection()
    with conn.cursor() as cursor:
        cursor.execute("""
            INSERT INTO ticket_comments (ticket_id, user_id, comment)
            VALUES (%s, %s, %s)
        """, (ticket_id, user_id, comment_text))
    conn.commit()
    conn.close()

    flash("Comment added.", "success")
    return redirect(url_for("ticket_detail", ticket_id=ticket_id))
```

jQuery opcional: podrías hacer que el formulario de comentarios se envíe por AJAX y actualice la lista sin recargar, pero para la primera versión no es obligatorio.

9. Administración de usuarios (solo Admin)

9.1. Listar usuarios

```
@app.route("/users")
@login_required
@role_required("ADMIN")
def users_list():
    conn = get_db_connection()
    with conn.cursor() as cursor:
        cursor.execute("SELECT id, name, email, role, created_at FROM users ORDER BY
created_at DESC")
        users = cursor.fetchall()
    conn.close()
    return render_template("users_list.html", users=users)
```

9.2. Cambiar rol de usuario (simple)

```
@app.route("/users/<int:user_id>/role", methods=["POST"])
@login_required
@role_required("ADMIN")
def user_change_role(user_id):
    new_role = request.form.get("role")
    if new_role not in ["ADMIN", "AGENT", "USER"]:
        flash("Invalid role.", "danger")
        return redirect(url_for("users_list"))

    conn = get_db_connection()
    with conn.cursor() as cursor:
        cursor.execute("UPDATE users SET role = %s WHERE id = %s", (new_role, user_id))
    conn.commit()
    conn.close()

    flash("Role updated.", "success")
    return redirect(url_for("users_list"))
```

9.3. Template users_list.html (extracto)

```
{% extends "base.html" %}
{% block content %}
<div class="container mt-4">
  <h3>Users</h3>
  <table class="table table-striped">
```

```
    <thead>
      <tr>
        <th>#</th><th>Name</th><th>Email</th><th>Role</th><th>Actions</th>
      </tr>
    </thead>
    <tbody>
      {% for u in users %}
      <tr>
        <td>{{ u.id }}</td>
        <td>{{ u.name }}</td>
        <td>{{ u.email }}</td>
        <td>{{ u.role }}</td>
        <td>
          <form method="post" action="{{ url_for('user_change_role', user_id=u.id) }}"
class="d-flex">
            <select name="role" class="form-select form-select-sm me-2">
              <option value="USER" {% if u.role == "USER" %}selected{% endif
%}>USER</option>
              <option value="AGENT" {% if u.role == "AGENT" %}selected{% endif
%}>AGENT</option>
              <option value="ADMIN" {% if u.role == "ADMIN" %}selected{% endif
%}>ADMIN</option>
            </select>
            <button type="submit" class="btn btn-sm btn-primary">Save</button>
          </form>
        </td>
      </tr>
      {% endfor %}
    </tbody>
  </table>
</div>
{% endblock %}
```

10. Seguridad básica que deben aplicar

Puntos mínimos que el proyecto debe cumplir:

1. **Contraseñas hasheadas**

    - Uso de generate_password_hash y check_password_hash.
    - No almacenar contraseñas en texto plano.
2. **Uso de parámetros en SQL**
    - Siempre usar cursor.execute("... WHERE email = %s", (email,)).
    - Nunca concatenar strings con variables directamente en el SQL.
3. **Control de acceso por rol**
    - Rutas protegidas con @login_required.
    - Rutas de administración protegidas con @role_required("ADMIN").
4. **Manejo de errores**
    - No exponer trazas completas a usuarios finales en producción.
    - debug=False en producción.
5. **Protección básica de sesiones**
    - SECRET_KEY robusto.
    - No guardar información sensible (como password) en session.

Sugerencia de mejora (opcional): implementar token CSRF manual o usando una librería adicional (Flask-WTF), si el curso lo permite.

11. Documentación requerida (obligatoria)

En el folder o la carpeta docs/ se debe incluir:

1. **manual_usuario.md**
    - Cómo iniciar sesión.
    - Cómo crear un ticket.
    - Cómo ver y actualizar tickets.
    - Qué ve un Admin vs un Agent vs un User.
2. **manual_tecnico.md**
    - Descripción de la arquitectura (front-end, back-end, base de datos).
    - Diagrama ER de las 3 tablas (er_diagram.png).
    - Instrucciones de instalación:
        - Crear base de datos.
        - Ejecutar script SQL.
        - Configurar .env.

        - Instalar dependencias pip install -r requirements.txt.
        - Ejecutar python app.py.
    - Descripción de rutas principales (tabla de endpoints).
    - Dirección o URL del proyecto en Github
3. **Capturas de pantalla en docs/screenshots/**
    - login.png
    - dashboard.png
    - tickets_list.png
    - ticket_detail.png
    - users_list.png (si el usuario es admin)

12. Publicación en GitHub (requisito final del proyecto)

Cada estudiante debe:

1. Crear un repositorio en GitHub con nombre sugerido:
    - helpdesk-agosto-2025
2. Subir:
    - Todo el código del proyecto.
    - El archivo requirements.txt.
    - La carpeta docs/ con manuales y capturas.
3. Incluir un README.md con:
    - Descripción breve del proyecto.
    - Tecnologías usadas.
    - Pasos de instalación rápidos.
    - Capturas clave (o enlace a docs/screenshots).

Regla: Explicar en el README.md qué variables deben configurarse.

13. Actividad de cierre

Para evidenciar que dominaron los temas, cada estudiante debe:

1. **Extender** el sistema con al menos una mejora, por ejemplo:

- Buscador de tickets por título/estado.
- Filtros por prioridad/fecha.
- **Dashboard con estadísticas simples (cantidad de tickets por estado, tabla).**
    i. `select status, count(*) form tickets group by status;`

2. **Documentar** la mejora en manual_tecnico.md y mostrarla en las capturas.
3. **Hacer un pequeño demo (video)** (5–7 minutos) donde:
    - Muestren el flujo de uso (login → crear ticket → actualizar → comentar).
    - Expliquen brevemente las decisiones de diseño (datos, seguridad, UX).
    - Describan la mejora extra implementada.
    - Para el video pueden utilizar MS Teams, Google Meet, o OBS Studio (https://obsproject.com/) compartir la pantalla y grabar la interacción.