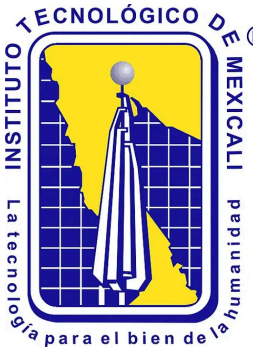

Tecnológico Nacional de México

Instituto Tecnológico de Mexicali

Tema:

Integración de Funcionalidades Avanzadas y Calculadora

Integrantes:

-Quevedo Ramírez Ary Alberto

-Nuñez Villalobos Angel Eduardo

-Campoy Lara Marcos

-Vega Robles Alexis

Carrera:

Ing. En Sistemas Computacionales

Materia:

Programación Web

Docente:

López Castellanos Carlos Alberto

Mexicali, Baja California a 09 de Mayo de 2025

Validación de los Formularios

Para garantizar una experiencia de usuario más fluida y evitar errores en el envío del formulario, se implementaron validaciones en los campos.

Expresiones Regulares

Se incorporaron expresiones regulares para validar campos clave, asegurando que los datos ingresados tengan el formato adecuado. Se aplicaron en los siguientes casos:

- **Correo electrónico**: Se verifica que tenga una estructura válida.
- **Teléfono**: Se exige que contenga únicamente dígitos y tenga la longitud correcta.
- **Enlace**: Se valida que la URL ingresada sea legítima.

Gracias a estas expresiones regulares, se evita que el usuario ingrese información incorrecta desde el principio.

Verificación de Campos Vacíos y Restricciones Numéricas

Se estableció un control sobre los campos obligatorios, asegurando que el usuario no pueda enviar un formulario sin completar la información esencial. Además, se impusieron restricciones en los valores numéricos: el tiempo de preparación debe encontrarse dentro de un rango razonable, permitiendo un mínimo de un minuto y un máximo de 12 horas. De igual manera, el número de personas a servir no puede ser menor a 1, evitando entradas inválidas.

Validación de Imágenes y Formatos Admitidos

Para garantizar la compatibilidad con el sistema, se implementó un filtro que permite únicamente la carga de imágenes en formatos comunes como JPG, PNG y GIF.

Mensajes de Error en Tiempo Real

Para mejorar la experiencia del usuario, se optimizó la visualización de errores para que aparezcan de inmediato cuando el usuario introduce un dato incorrecto. Los mensajes de advertencia en color rojo se ubican justo debajo de los campos afectados, facilitando la corrección antes de intentar enviar el formulario.

Pagina de Tienda y LocalStorage

Carrito

El carrito fue implementado haciendo uso de JavaScript, primeramente se creó una lista con el nombre de products donde se guardó la información de cada producto que se tomara en cuenta para agregar al carrito y para la pagina

Después, se declaró otra lista llamada cart donde se guardaría la información de el carro, haciendo uso de la funcion updateCart(), el cual tiene un .map para crear una plantilla para los objetos que seran agregados a este sin la necesidad de anotar manualmente uno a uno el objeto que va a ser agregado, dandole un nivel mas de dinamismo a el programa

Además, esta función coloca los items de el carrito dentro de el contenedor donde pertenecen y guarda los objetos guardados dentro de el carrito en un localstorage, para que estos se conserven aun si te sales de la página o la recargas

Otra función utilizada es el getTotal la cual actualizara y llevara cuenta de el valor total de los precios de el carrito, tomando en cuenta la cantidad de productos en el carritos

Estas funciones son necesarias para crear la funcion addToCart la cual se dispara al presionar los botones de agregar al carrito debajo de cada carrito

También se implementaron funciones para los botones de incrementar, decrementar y borrar las cuales manipulan el número de productos dentro del carrito o de plano los quitan de la lista

Se tiene una funcion que se dispara apenas se carga el DOM, la cual hará una llamada a una funcion llamada loadCartFromLocalStorage(), Esta funcion se encarga de checar la localstorage por informacion acerca de los objetos del carrito para cargarlos apenas carga el DOM de la página

Calculadora

La calculadora de practicas anteriores fué implementada en la misma página de la tienda, teniendo un botón para ocultar y mostrar dicha calculadora

A la calculadora se le asignan EventListeners para cada botón en donde a través de una estructura de control la cual toma en cuenta el texto de los botones para decidir la función o código a ejecutar o decidir si el texto en la pantalla ha de borrarse o mantenerse

Index

Se utilizó una cantidad pequeña de JavaScript para modificar el Index, haciendo que se pueda pasar entre las imágenes de las distintas categorías de comida a través de un script que manipula las clases de la categoría a la que se le hace click, haciendo que estas reciban la clase de “active”, cambiando el color de la etiqueta y escondiendo los otros grids de imágenes para mostrar solo el que corresponde a la categoría seleccionada

Página de camisetas oficiales

se creó un html para crear una página donde se dan especificaciones sobre las camisetas oficiales y se introdujo una tabla para presentarlas, colocando el hipervínculo en el html de la tienda en la parte de texto que dice “Camisetas oficiales”

el hipervínculo se integró desde el archivo Js de la tienda

Backend:

El Backend de esta aplicación web fué creado haciendo uso de Flask en el lenguaje de Programación Python, con una base de datos creada en MySQL

Existen rutas para cada una de las páginas disponibles en este proyecto, declaradas de la siguiente manera:

```
@app.route("/inicio")
def inicio():
    return render_template("public/index.html")
```

Las especificaciones de la conexión con la base de datos se encuentran en Config.Py, Las cuales son las siguientes:

```
class DevelopmentConfig:
    DEBUG = True
    SECRET_KEY = "qhrf$edjYTJ)*21nsThdK"
    MYSQL_HOST = "127.0.0.1"
    MYSQL_USER = "Python"
    MYSQL_PASSWORD = "123"
    MYSQL_DB = "tiendarecetitas"

config = {"development": DevelopmentConfig}
```

Hay rutas que cuentan con instrucciones mas específicas y parámetros pues estas interactúan directamente con la base de datos de una manera u otra
Antes de esto, cabe destacar que para facilitar las interacciones con la base de datos, se hizo uso de el modelo ORM, Es un modelo que considera las tablas de una BDD como objetos de un lenguaje de programación, Por lo que fué necesario declarar una entidad, la cual es una clase con los atributos de la tabla:

```
class pagina():
       def __init__(self, idReceta, nombre, autor, cantidadpersonas, tiempo, dificultad, ingredientes, tips, preparacion, rutaimg, rutaimgapoyo, idUser, estrellas, votos_totales) -> None:
        self.idReceta = idReceta
        self.nombre = nombre
        self.autor = autor
        self.cantidadpersonas = cantidadpersonas
        self.tiempo = tiempo
        self.dificultad = dificultad
        self.ingredientes = ingredientes
        self.tips = tips
        self.preparacion = preparacion
        self.rutaimg = rutaimg
        self.rutaimgapoyo = rutaimgapoyo
        self.idUser = idUser
        self.estrellas = estrellas
        self.votos_totales = votos_totales
```

```
from flask_login import UserMixin

class User(UserMixin):
     def __init__(self, id, username, password, usertype, fullname="", email="") -> None:
        self.id = id
        self.username = username
        self.password = password
        self.fullname = fullname
        self.email = email
        self.usertype = usertype
```

Y posteriormente, declarar los modelos de cada entidad, Las cuales contienen métodos que interactuan con la BDD

```
from .entities.pagina import pagina

class ModelPagina:
    # Método para registrar nuevos usuarios
    @classmethod
    def agregar(cls, db, pagina):
        try:
            cursor = db.connection.cursor()
            cursor.execute(
                    "call sp_AddReceta(%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)",
                            (pagina.nombre, pagina.autor, pagina.cantidadpersonas, pagina.tiempo, pagina.dificultad,
                             pagina.ingredientes, pagina.tips, pagina.preparacion, pagina.rutaimg, pagina.rutaimgapoyo,
                              pagina.idUser , pagina.estrellas)
            )
            db.connection.commit()
            return True
        except Exception as ex:
            raise Exception(ex)

    @classmethod
    def obtenerrecetas(cls, db):
        cursor = db.connection.cursor()
        cursor.execute("SELECT * FROM recetas ORDER BY idReceta")
        results = cursor.fetchall()
        cursor.close()
        return results

    @classmethod
    def eliminar(cls, db, id):
        try:
            cursor = db.connection.cursor()
            cursor.execute("DELETE FROM recetas WHERE idReceta = %s", (id,))
            db.connection.commit()
            cursor.close()
            return True
        except Exception as ex:
            raise Exception(ex)

    # Para sacar informacion de una pagina en especifico
    @classmethod
    def RecetaPorID(cls,db, id):
        cursor = db.connection.cursor()
        cursor.execute("SELECT * FROM recetas where idReceta = %s", (id,))
        row = cursor.fetchone()
        if row != None:
                PaginaRe = pagina(row[0],row[1], row[2], row[3] ,row[4] ,row[5],row[6], row[7],row[8],row[9],row[10],row[11],row[12],row[13])
            return PaginaRe
        else:
            return None

    @classmethod
    def ActualizarVotos(cls,db,estrellas,id):
        cursor = db.connection.cursor()
         cursor.execute("UPDATE recetas SET estrellas = estrellas+%s WHERE idReceta = %s;", (estrellas,id))
          cursor.execute("UPDATE recetas SET votos_totales = votos_totales+1 WHERE idReceta = %s;", (id))
        db.connection.commit()
        cursor.close
        return True
```

```python
from .entities.users import User


class ModelUsers():
    #Metodo para verificar las credenciales para el inicio de sesion
    @classmethod
    def login(self, db, user):
        try:
            cursor = db.connection.cursor()
            cursor.execute(
                "call sp_verifyIdentity(%s, %s)",
                (user.username, user.password)
            )
            row = cursor.fetchone()
            if row[0] != None:
                user = User(row[0], row[1], row[2], row[4], row[3])
                return user
            else:
                return None
        except Exception as ex:
            raise Exception(ex)
    # Metodo para obtener informacion de un usuario a traves de su ID
    @classmethod
    def get_by_id(cls, db, id):
        try:
            cursor = db.connection.cursor()
            cursor.execute(
                "SELECT * FROM users WHERE id = %s", (id,)
            )
            row = cursor.fetchone()
            if row is not None:
                return User(row[0], row[1], row[2], row[4], row[3])
            else:
                return None
        except Exception as ex:
            raise Exception(ex)
    # Metodo para registrar nuevos usuarios
    @classmethod
    def registrar(self, db, user):
        try:
            cursor = db.connection.cursor()
            cursor.execute(
                "call sp_AddUser(%s, %s, %s, %s, %s)",
                (user.username, user.password,user.fullname, 0, user.email)
            )
            db.connection.commit()
            return True
        except Exception as ex:
            raise Exception(ex)

    @classmethod
    def actualizarnombre(cls,db,nom,id):
        try:
            cursor = db.connection.cursor()
                 cursor.execute("UPDATE users SET fullname = %s  WHERE id =
%s",(nom, id))
            db.connection.commit()
            cursor.close
            return True
        except Exception as ex:
            raise Exception(ex)
```

Como se puede observar, Cada método cuenta con una llamada a “Cursor” , Este Cursor es un objeto que hace referencia a el cursor de la conexión con la BDD y es el que se encarga de hacer todas las interacciones con dicha base de datos, es por eso que se llama el cursor cada que se quiere ejecutar una consulta o un comando con SQL utilizando cursor.execut

En caso de que esta interacción implique una modificación, Ya sea una actualización, creación o borrado de algún valor, es necesario hacer un commit con la base de datos referenciada

La razón de esto es por que es necesario completar la transacción para que los valores cambiados se reflejen en la BDD

Cuando se hacen consultas, Los valores regresados por dicha consulta tendrán la forma de una Matríz Bidimensional, sin embargo, gracias a que creamos una entidad para cada tabla, podemos instanciar objetos para no tener que referenciar las coordenadas de la matríz

```
 @classmethod
    def RecetaPorID(cls,db, id):
        cursor = db.connection.cursor()
        cursor.execute("SELECT * FROM recetas where idReceta = %s", (id,))
        row = cursor.fetchone()
        if row != None:
                PaginaRe = pagina(row[0],row[1], row[2], row[3] ,row[4] ,row[5]
,row[6], row[7],row[8],row[9],row[10],row[11],row[12],row[13])
            return PaginaRe
        else:
            return None
```

-Ejemplo donde se crea y retorna un objeto página a partir de una consulta de BDD

Esto facilita el trabajo cuando se quieren referenciar valores específicos de una página o usuario, mas no es 100% necesario pues simplemente se puede hacer referencia a las coordenadas de la matriz

Para el sistema de Login se importó el modulo de flask_login el cual provee herramientas necesarias para manejar sesiones y referenciar las propiedades de el usuario que se encuentre en la sesión actual.

En los casos en los que sea necesario interactuar con valores ingresados por un usuario en la página, como en el caso de los formularios de Login y Registrar, se hace uso de el metodo “POST”

```
@app.route("/login", methods=["GET", "POST"])
def login():

    if request.method == "POST":
        # Pa checar si es el formulario de registro
        if "RegNombre" in request.form:
            nombre = request.form["RegNombre"]
            passw = request.form["RegPass"]
            fullname = request.form["RegNCom"]
            mail = request.form["RegMail"]

            user = User(0, nombre, passw, 0, fullname, mail)
            ModelUsers.registrar(db, user)
            flash("Registro exitoso", "success")
            return render_template("auth/login.html")
        # Pa checar si es el formulario de login
        elif "username" in request.form:
            user = User(0, request.form['username'],
                        request.form['password'], 0, 0)
            logged_user = ModelUsers.login(db, user)
            if logged_user is not None:
                login_user(logged_user)

                if logged_user.usertype == 1:

                    flash("Bienvenido administrador", "success")
                    return redirect(url_for("admin"))
                else:
                    flash("Inicio de sesion exitoso", "success")
                    return redirect(url_for("inicio"))
            else:
                flash("Acceso rechazado...", "danger")
                return render_template("auth/login.html")
    else:
        return render_template("auth/login.html")
```

En esta ruta se puede ver como es que al darle click a un boton (El cual hace el metodo POST), va a iniciar una estructura de control la cual revisará si el formulario en el que se pulsó el botón cuenta con un campo con el nombre “RegNombre” o “Username”, el primero siendo el username de registro y el otro el username para iniciar sesión

Los comandos request.form obtienen el valor ingresado en el campo referenciado y se asigna a una variable para posteriormente ser utilizada
Pagina principal de admin

Página de administración y usuarios admin

La página **recetas.html**, ubicada en la sección de **templates/admin/** está diseñada para ser utilizada únicamente por usuarios con permisos de administrador. Su función principal es mostrar todas las recetas almacenadas en la base de datos y ofrecer una opción sencilla para eliminarlas. Presenta los datos en una tabla clara, donde cada fila contiene el título, el autor y un botón para borrar la receta correspondiente.

Ruta `admin`

```python
@app.route("/admin")
@admin_required
@login_required
def admin():
    recetas = ModelPagina.obtenerrecetas(db)
    return render_template("admin/recetas.html", recetas=recetas)
```

La ruta **/admin**, protegida con los decoradores **@admin_required** y **@login_required**, permite a los administradores visualizar todas las recetas registradas en el sistema. Recupera los datos mediante **ModelPagina.obtenerrecetas(db)** y los muestra en la página **recetas.html**, ubicada en la carpeta **templates/admin/**.

Esta vista presenta las recetas en formato de tabla, donde cada entrada muestra su título, autor y una opción para eliminar la receta si así se desea. Funciona como el panel principal desde el cual los administradores gestionan el contenido disponible.

Ruta `eliminar_receta`

```python
@app.route("/admin/eliminar/<int:id>", methods=["POST"])
@admin_required
@login_required
def eliminar_receta(id):
    ModelPagina.eliminar(db, id)
    flash("Receta eliminada exitosamente", "success")
    return redirect(url_for("admin"))
```

Se creó la ruta **/admin/recetas/eliminar/<int:id>**, que permite a los administradores eliminar recetas de forma segura mediante una petición POST. Esta ruta utiliza el método **ModelPagina.eliminar(db, id)** para borrar de manera definitiva el registro correspondiente en la base de datos.

Al completar la operación, se utiliza **`flash`** para mostrar un mensaje de éxito al usuario, confirmando que la receta fue eliminada. Luego, se redirige automáticamente de nuevo a la lista de recetas (**`admin_recetas`**) para actualizar la vista sin necesidad de recargar manualmente.

La acción de eliminación está protegida tanto por autenticación (**`@login_required`**) como por verificación de permisos de administrador (**`@admin_required`**), asegurando que solo usuarios autorizados puedan realizar cambios en los datos del sistema.