Proyecto Flask-Login con integración Flask-SqlAlchemy

Nombre de Archivo	Descripción
app.py	Punto de entrada de la aplicación. Crea y ejecuta la app.
app_factory.py	Implementa el patrón de factory app. Crea y configura la instancia Flask.
config.py	Contiene las configuraciones de la aplicación como la BD.
models/base.py	Define la instancia de db, modelo BaseEntity y métodos comunes.
models/entities.py	Modelos de SQLAlchemy que mapean a tablas de la BD.
routes.py	Blueprint con vistas que renderizan templates.
auth.py	Blueprint para autenticación de usuarios.
apis/calificaciones.py	API REST de recursos de calificaciones.
apis/personas.py	API REST de recursos de personas.

app_factory.py

- create_app():
  - Patrón de aplicación factory.
  - Permite crear una app con diferentes configuraciones. Útil para entornos como desarrollo, testing, producción.
  - Evita problemas con imports circulares.
  - Puede crear múltiples apps, por ejemplo una app API y otra app web.
- Flask(**name**):
  - Crea instancia de clase Flask, pasando **name** para encontrar templates, staticos y módulos relativos.
  - **name** será el nombre del módulo actual.
- app.secret_key:
  - Flask utiliza secret_key para firmar cookies, protect CSRF y sesiones de usuario.
  - Debe mantenerse secreta y aleatoria.
  - Se puede generar con os.urandom(24).
  - Debe configurarse distinta en producción.
- app.config:
  - Diccionario que contiene configuraciones de la aplicación Flask.
  - Permite configurar opciones como SQLALCHEMY_DATABASE_URI.
  - Puede también cargarse desde archivo, variable de entorno, etc.
- db.init_app(app):
  - Inicializa extensión Flask-SQLAlchemy, pasando la app.
  - Hace que SQLAlchemy use el contexto de aplicación flask.
  - db será la instancia para interactuar con la BD.
- Blueprint:
  - Utilizado para organizar el código en módulos reutilizables.
  - Se registran vistas y otros códigos relacionados.
  - Se registran con app.register_blueprint().

  - Permiten prefijar rutas, por ejemplo /auth/login.
- api = Api(app):
  - Crea API REST con Flask-RESTful pasando la app.
  - Permite crear clases Recursos a las que se asignan rutas y métodos HTTP.
  - Facilita la construcción de APIs RESTful.
- add_resource():
  - Registra Recursos de API a rutas específicas.
  - Recursos son clases que heredan de flask_restful.Resource.
  - GET en Recurso mapea a método HTTP GET en esa ruta.
- login_manager.init_app():
  - Inicializa extensión Flask-Login, pasando la app.
  - Permite manejar sesión de usuarios y autenticación.
  - Requiere función user_loader() para cargar usuario.
- db.create_all():
  - Crear las tablas de la BD según los modelos de SQLAlchemy.
  - Analiza las clases modelo y crea las tablas correspondientes.
  - Solo debe ejecutarse una vez, no cada request.
- return app:
  - Retorna la instancia de app configurada.
  - Permite importar y usar esta app factory en otro módulo.

config.py:

- Variables de configuración:
  - Por convención se nombran en mayúsculas.
  - Contienen parámetros que pueden variar según el entorno.
  - Se accede mediante app.config['NOMBRE_VARIABLE']
- db_*:
  - Configuraciones específicas para conectarse a la BD.
  - Tipo de conector, credenciales de usuario, dirección del host, nombre BD.
  - Se utilizan para construir SQLALCHEMY_DATABASE_URI.

app.py:

- create_app():
  - Llama a la función factory, creando una app con las configuraciones default.
  - La app está configurada mediante los parámetros en config.py.
- if **name** == '**main**':
  - Punto de entrada de la aplicación.
  - Permite importar este módulo sin ejecutar la app.
  - Solo se ejecuta la app si se corre app.py directamente.
- app.run():
  - Ejecuta la aplicación Flask incorporada en modo debug.
  - Recargará el servidor automáticamente al guardar cambios.

  - Útil en desarrollo, no recomendado en producción.

modules/auth.py:

- Blueprint:
  - Organiza las vistas/rutas de autenticación en un módulo reutilizable.
  - Puede tener su propio nombre, importar vistas, templates, etc.
- Decoradores:
  - @blueprint.route: registra ruta a una vista en el blueprint.
  - @login_required: requiere que el usuario esté logueado, caso contrario redirige.
- Flask-Login:
  - login_manager: objeto que contiene la lógica de inicio/cierre de sesión.
  - user_loader(): callback para buscar usuario por su id. Requerido.
  - unauthorized_handler: callback cuando el usuario no está autorizado para acceder.
  - login_user(): recibe el usuario y lo marca como logueado.
  - logout_user(): cierra la sesión del usuario.
  - current_user: usuario actualmente logueado.
- Valida credenciales:
  - Obtiene usuario de la BD filtrando por username.
  - Verifica password hash con check_password().
  - Si coincide, loguea al usuario con login_user().
- Manejo de sesión:
  - Usa session para almacenar datos temporales durante la sesión.
  - flash() envía mensajes que estarán disponibles la próxima request.
- Redirecciones:
  - url_for(): genera las URLs a partir del nombre de la vista.
  - redirect(): crea una respuesta HTTP 302 de redirección a otra vista.

modules/apis/personas.py:

Define un recurso RESTful para el modelo Persona utilizando Flask-RESTful.

Se importan las clases Resource y reqparse de Flask-RESTful para crear el recurso y analizar la entrada.

También se importa login_required para proteger los métodos requiriendo autenticación.

Y los modelos de SQLAlchemy como Persona, Genero y Lugar para interactuar con la base de datos.

Se define un analizador request_parser para validar y limpiar los parámetros entrantes en los requests.

La clase PersonasResource hereda de Resource e implementa los métodos HTTP para el CRUD:

- GET para obtener todas las Personas o una sola por ID. Serializa los resultados a JSON para devolverlos.

- POST para crear una nueva Persona a partir de los datos entrantes. La guarda en la BD y devuelve códigos HTTP adecuados.
- PUT para actualizar una Persona existente por ID. Toma los datos entrantes y los asigna a la Persona obtenida.
- DELETE para eliminar una Persona por su ID.

Se manejan posibles excepciones y se devuelven respuestas HTTP acordes en cada caso.

Finalmente, se registra el recurso PersonasResource en el API para que los métodos CRUD estén disponibles en determinadas rutas URL.

- Flask-RESTful Resource:
  - Clase base para definir recursos REST.
  - Permite rápidamente mapear métodos HTTP a funciones Python.
- Flask-RESTful reqparse:
  - Analiza y parsea parametros entrantes en requests.
  - Valida, limpia y hace conversiones.
- @login_required:
  - Decorador para requerir autenticación.
  - Protege el acceso a los recursos.
- Métodos HTTP:
  - GET: Obtiene recursos.
  - POST: Crea nuevo recurso.
  - PUT: Actualiza recurso existente.
  - DELETE: Elimina recurso.
- try/except:
  - Manejo de errores y excepciones.
  - Permite detectar y handlear fallas.
- Códigos de respuesta HTTP:
  - Indican el resultado de la operación al cliente.
  - 201 CREATED, 200 OK, 404 NOT FOUND, etc.

modules/apis/calificaciones.py: (modulo de ejemplo)

- Flask-RESTful:
  - Resource: clase base para crear recursos REST.
  - reqparse: analiza y valida datos de requests entrantes.
- @method_decorators:
  - Aplica decoradores a los métodos HTTP del recurso.
  - login_required protege los métodos requiriendo autenticación.
- Métodos HTTP:
  - GET: implementa el método HTTP GET. Devuelve los datos.
  - POST: implementa POST. Analiza args y crea un nuevo recurso.

  - DELETE: implementa DELETE. Borra recurso por id.
- API REST:
  - La clase CalificacionesResource expone los métodos CRUD para calificaciones.
  - Los métodos se asignan a rutas y operaciones HTTP específicas.
  - Se obtiene una API RESTful fácilmente.

modules/models/entities.py

- SQLAlchemy: mapea clases Python a tablas de BD relacional.
- db.Model:
  - Clase base para modelos de SQLAlchemy que representan tablas.
- db.Column:
  - Define las columnas del modelo, indicando tipo de dato.
  - Puede especificar constraints como primary_key, unique, nullable.
- Relaciones:
  - db.relationship(): Define relaciones entre modelos.
  - backref: atributo en el otro modelo para acceder a este.
- Métodos:
  - **init**: Constructor del modelo. Se ejecuta al crear nuevas instancias.
  - hybrid_property: Propiedad que es getter/setter pero también funciona en queries.
- Constructor:
  - Crea instancias pasando valores para cada columna.
  - Permite crear objetos asociados a tablas para interactuar con la BD.
- Usuarios:
  - Usermixin proporciona funciones de autenticación.
  - set_password(): hashea y guarda la contraseña.
  - check_password(): verifica contraseña en texto plano con el hash.

modules/models/base.py

- db:
  - Instancia global de SQLAlchemy, representa la BD.
  - Se inicializa en la app factory pasando app.
  - Disponible para modelos a través de imports.
- BaseEntity:
  - Clase abstracta que modelo heredan.
  - Define comportamientos comunes para los modelos.
  - Evita duplicar código como guardar/borrar.
- Métodos:

  - guardar(): persiste una nueva instancia del modelo en la BD.
  - borrar(): elimina la instancia de la BD.
  - crear_y_obtener(): busca o crea una instancia.
  - Utilizan la session de SQLAlchemy para las operaciones.
- db.session:
  - Administra operaciones y trabajo con la BD.
  - Mantienen el estado durante requests.
  - init_app() la vincula al contexto de Flask.

modules/routes.py

- Blueprints:
  - Organiza las vistas en módulos independientes.
  - Se registran con app.register_blueprint().
- FlaskForm:
  - Clase base para formularios WTForms con validación.
  - Los campos se definen como clases de wtforms.
- Decoradores:
  - @login_required: requiere que el usuario esté autenticado.
  - @errorhandler(404): maneja errores HTTP 404 a nivel de app.
- Jinja2:
  - render_template(): renderiza templates Jinja2 pasando variables.
  - url_for(): genera URLs hacia vistas por su nombre.
- Request/Response:
  - request: solicitud entrante. Contiene parámetros, headers, cookies, etc.
  - session: almacena datos durante la sesión de usuario.
  - flash(): almacena mensajes para la próxima request.
  - redirect(): genera respuestas de redirección HTTP.
- Formularios:
  - FlaskForm con campos de wtforms definen el formulario.
  - validate_on_submit() dispara la validación al enviar datos.
  - Accede a datos del form con form.name.data.

/static/personas_script.js

- DOMContentLoaded:
  - Evento que se dispara cuando el HTML es parseado y el DOM está listo.
  - Permite ejecutar código cuando el documento está completamente cargado.
- fetch(url):
  - Método para realizar peticiones HTTP asíncronas (AJAX).
  - Retorna una Promise que se resuelve en la respuesta.
- .then(callback):
  - Método de Promise que ejecuta una función cuando la Promise se resuelve.
  - Utilizado para manejar la respuesta de la petición fetch.

- .catch(callback):
    - Método de Promise que maneja errores si la Promise es rechazada.
- response.json():
    - Parsea el body JSON de la respuesta fetch.
    - Retorna otra Promise con los datos.
- CRUD:
    - getPersonas(): Realiza petición get para obtener recursos.
    - agregarPersona(): Realiza petición post para crear nuevo recurso.
    - eliminarPersona(): Realiza petición delete para borrar un recurso.
- Template literals:
    - Permiten incrustar expresiones y variables en strings.
    - Facilitan construir strings de forma dinámica.

/static/script.js (ejemplo)

- Funcionalidad similar para recurso Calificaciones.
- Explicación técnica equivalente a personas_script.js.

En conclusión, se utilizan buenas prácticas de Flask como la factory y blueprints para estructurar el código, Flask-SQLAlchemy para bases de datos relacionales, Flask-Login para autenticación, Flask-RESTful para APIs REST, y Jinja2 para templates.