Documentación Técnica

Estructura de Archivos y Carpetas

Archivo/Carpeta	Descripción
app.py	Archivo principal que crea y ejecuta la aplicación Flask
app_factory.py	Define la fábrica de aplicaciones para crear la app Flask
config.py	Contiene configuraciones globales como conexión a BD
templates/	Carpeta con archivos HTML de las vistas
static/	Carpeta para archivos estáticos como CSS, JS, imagenes
modules/	Carpeta con módulos de la aplicación
modules/auth.py	Módulo de autenticación y login de usuarios
modules/routes.py	Módulo con rutas básicas de la aplicación
modules/routes_personas.py	Módulo con rutas relacionadas al CRUD de Personas
modules/apis/	Carpeta con módulos de APIs REST
modules/apis/personas.py	API REST para el modelo Persona
modules/models/	Carpeta con modelos de la BD
modules/models/entities.py	Define los modelos de la BD como clases Python
modules/models/base.py	Base para los modelos con SQLAlchemy

Tecnologías Utilizadas

Las principales tecnologías/librerías utilizadas son:

- **Flask**: Framework web en Python para crear la aplicación. Provee enrutamiento, templates, etc.
- **Flask-SQLAlchemy**: Extensión que integra SQLAlchemy para acceder a BD.
- **Flask-Login**:Simplifica manejo de sesiones y login de usuarios.
- **Flask-RESTful**: Crea APIs REST fácilmente.
- **Flask-WTF**: Biblioteca para crear formularios.
- **SQLAlchemy**: Biblioteca ORM para mapear objetos Python a tablas de BD.
- **MySQL**: Motor de base de datos utilizado.

Módulos

- **app.py**: Archivo principal, crea la aplicación Flask importando la fábrica `create_app()` de app_factory.py. Ejecuta el servidor de desarrollo.
- **app_factory.py**: Define la función `create_app()` que crea y configura la aplicación Flask:
  - Inicializa la BD con Flask-SQLAlchemy.

  - Registra los blueprints de rutas, auth y APIs.
  - Configura el manejo de errores 404.
  - Crea las tablas si no existen.
  - Retorna la instancia de la aplicación.
- **config.py**: Contiene configuraciones globales como conexión a BD y otras variables utilizadas por algunos módulos.
- **auth.py**: Define rutas y lógica de autenticación y login con Flask-Login:
  - Usa blueprint `auth_bp` para agrupar las rutas.
  - `login`: Ruta y formulario de login. Valida usuario y clave, login con Flask-Login.
  - `logout`: Cierra la sesión actual.
  - `load_user`: Callback para cargar el usuario a partir de su id.
- **routes.py**: Rutas básicas con blueprint `routes_bp`:
  - `index`, `about`, `contact`: Vistas renderizadas con templates.
  - `page_not_found`: Handler de error 404.
- **routes_personas.py**: Este archivo define las rutas para el CRUD (crear, leer, actualizar, eliminar) de Personas utilizando un blueprint llamado `personas_bp`.:
  - Rutas para listar, crear, editar y eliminar Personas.
  - Paginación y validaciones de Flask-WTF.

  Contiene las siguientes rutas:

  - `obtener_personas_paginadas()`:
    - Ruta GET para listar personas paginadas.
    - Obtiene el número de página mediante parámetro GET.
    - Consulta personas paginadas de la BD con Paginator de Flask-SQLAlchemy.
    - Renderiza template `personas/personas.html` pasando los datos.
  - `editar_persona()`:
    - Ruta GET y POST para editar una persona por su id.

    - Obtiene la persona a editar o muestra 404 si no existe.
    - Valida y actualiza los campos enviados vía POST.
    - Renderiza el template `personas/editar_persona.html` para el formulario.
- `eliminar_persona()`:
    - Ruta POST para eliminar una persona por su id.
    - Obtiene la persona y la elimina de la BD.
    - Redirecciona al listado.
- `crear_persona()`:
    - Ruta GET y POST para crear una nueva persona.
    - Valida y crea la persona en la BD con los datos del formulario.
    - Redirecciona al listado.
    - Usa el template `personas/crear_persona.html` para el formulario.

- Persona.query.paginate(): Método para paginar los resultados. Recibe dos parámetros:
    - page: El número de página a consultar.
    - per_page: La cantidad de registros por página.

  Este método retorna un objeto Pagination de SQLAlchemy que contiene:

- items: Los resultados de la consulta paginada.
- has_prev: True si hay una página anterior.
- has_next: True si hay una página siguiente.
- pages: Total de páginas.

  paginate() se utiliza para implementar paginación en el backend cuando consultamos los datos de una tabla de la base de datos. Nos permite obtener solo un subconjunto de filas por página.

Utiliza Flask-WTF para validar datos de los formularios y Flask-SQLAlchemy para consultar/persistir en la BD.

Templates

- `base.html`: Plantilla base que se extiende en las demás. Define el menú y la estructura común.
- `index.html`: Página de inicio, extiende de base.html.
- `about.html`: Página de Acerca de, extiende de base.html.
- `contact.html`: Página de contacto, con formulario. Extiende de base.html.
- `404.html`: Página de error 404, extiende de base.html.
- `login.html`: Formulario de login, no usa plantilla base.
- `personas/personas.html`: Listado paginado de Personas. Extiende de base.html.
- `personas/crear_persona.html`: Formulario para crear Persona. Extiende de base.html.
- `personas/editar_persona.html`: Formulario para editar Persona. Extiende de base.html.

Los templates renderizan HTML con datos dinámicos utilizando el lenguaje de plantillas Jinja. Se organizan en una estructura modular extiendo de la plantilla base cuando corresponde.

- **personas.py**: API REST de Personas con Flask-RESTful (No utilizado en esta solución):
  - Extiende de `Resource` para heredar métodos GET, POST, PUT, DELETE.
  - Usa analizadores de args `RequestParser` para validar datos.
  - Maneja serialización y respuestas JSON.
- **Models**: Definen modelos de BD con Flask-SQLAlchemy:
  - `BaseEntity`: Clase base para los modelos.
  - `Persona, Genero, Lugar, etc.`: Modelos con relaciones y propiedades híbridas.
  - `User`: Usuario que autentica con Flask-Login.

Funcionalidades

Las principales funcionalidades de la aplicación son:

- Sistema de autenticación y login de usuarios.
- CRUD básico de Personas:

    - Listar / paginar personas.
    - Crear / editar / eliminar.
    - Búsquedas / filtros.
- API RESTful para acceder a Personas via JSON.
- Base de datos MySQL con ORM SQLAlchemy.
- Plantillas HTML con Jinja.
- Formularios con validaciones Flask-WTF.
- Framework Flask para estructurar el código.

Conclusión

Se implementó una aplicación web básica en Python con Flask utilizando buenas prácticas de estructura de proyecto, autenticación, ORM, APIs REST, plantillas, y más. El código está organizado en módulos y blueprint para facilitar el mantenimiento y expansión de funcionalidades. La documentación explica en detalle las tecnologías y la arquitectura implementada.