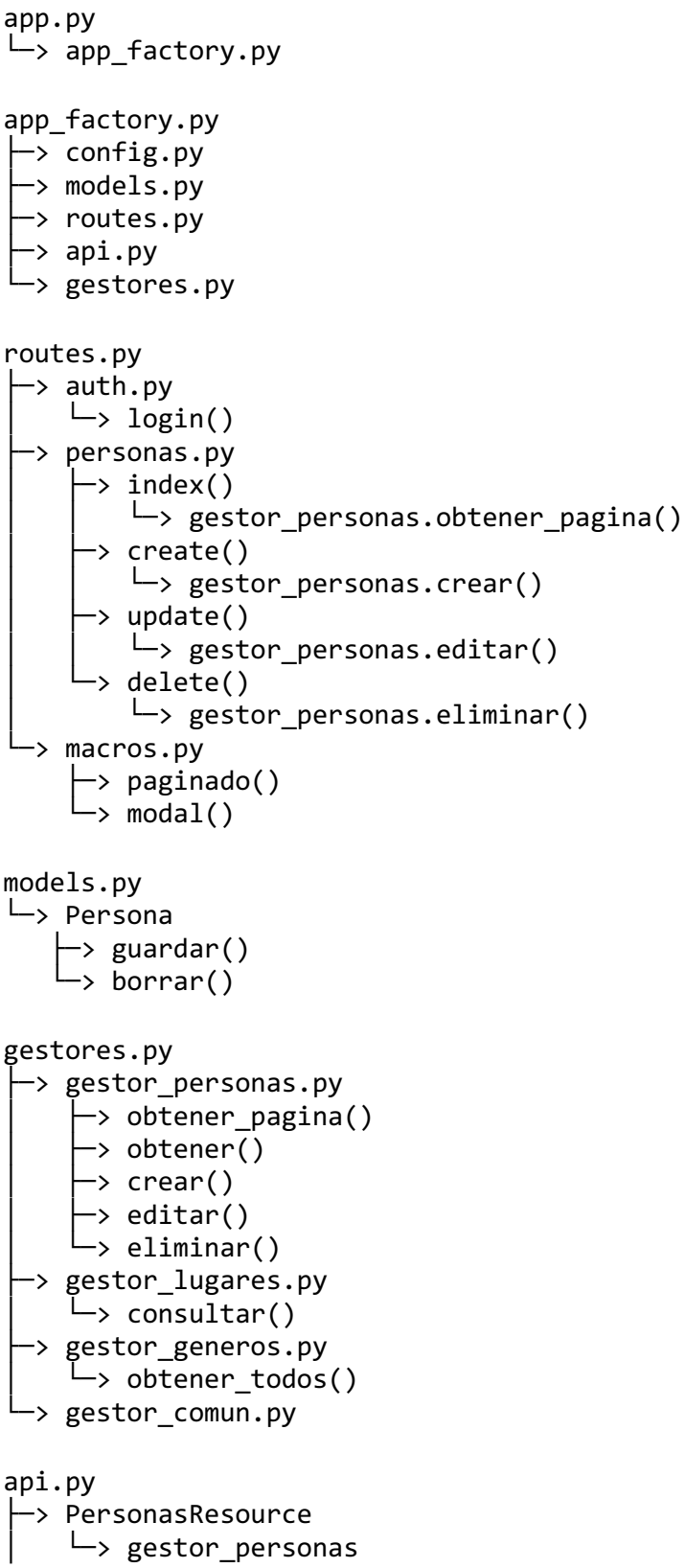
# Tabla de contenido

## Diagrama de la solución:



```
└─> LugaresResource
    └─> gestor_lugares
```

Explicación:

- app.py crea la aplicación Flask.
- app_factory.py tiene la configuración.
- routes contiene los módulos de rutas.
- auth.py maneja la autenticación.
- personas.py las rutas CRUD de personas.
- macros.py contiene macros reutilizables.
- models tiene los modelos SQLAlchemy.
- gestores encapsula lógica de negocio.
- api los recursos API REST.

De esta forma se tiene:

- app.py como punto de entrada
- app_factory.py para crear la app
- Rutas agrupadas por funcionalidad
- Modelos y lógica en módulos separados
- Macros para reutilizar código
- APIs REST

Es una arquitectura modular, limpia y efectiva para Flask.

## Tabla de contenido por módulo

| Archivo | Descripción |
|---|---|
| app.py | Creación de la app Flask |
| app_factory.py | Configuración de la app |
| config.py | Configuraciones globales |
| modules/apis/lugares.py | API REST de Lugares |
| modules/apis/personas.py | API REST de Personas |
| modules/common/gestor_comun.py | Lógica común a gestores |
| modules/common/gestor_lugares.py | Lógica de negocio de Lugares |
| modules/common/gestor_personas.py | Lógica de negocio de Personas |

| Archivo | Descripción |
|---|---|
| modules/models/base.py | Configuración base de datos |
| modules/models/entities.py | Modelos de la app |
| modules/auth.py | Autenticación de usuarios |
| modules/routes.py | Rutas generales |
| modules/routes_personas.py | Rutas CRUD Personas |
| templates/404.html | Página de error 404 |
| templates/about.html | Página Acerca De |
| templates/base.html | Template base |
| templates/contact.html | Formulario de contacto |
| templates/index.html | Página de inicio |
| templates/login.html | Formulario de login |
| templates/modal_eliminar.html | Macro modal eliminar |
| templates/paginado.html | Macro paginación |
| templates/personas/crear_persona.html | Formulario crear persona |
| templates/personas/editar_persona.html | Formulario editar persona |
| templates/personas/personas.html | Listado de personas |

## Archivo app.py

- Importa la función create_app() desde app_factory.py
- Crea la aplicación Flask llamando a create_app()
- Ejecuta la aplicación en modo debug cuando se ejecuta directamente

Este es el archivo principal para crear y ejecutar la aplicación. Utiliza el patrón de aplicación factory para crear la app.

## Archivo app_factory.py

- Importa dependencias como Flask, Flask-SQLAlchemy, Flask-Login, etc.
- Define la función create_app() que creará y configurará la aplicación Flask
- Inicializa la base de datos
- Registra los blueprints de rutas, authenticación, API's, etc.
- Configura Gestor de Login y CSRF
- Retorna la instancia de la aplicación Flask configurada

Este archivo encapsula toda la creación y configuración de la aplicación Flask. Sigue el patrón de aplicación factory.

## Archivo config.py

- Contiene configuraciones globales como conexión a la BD, parámetros, etc.
- Se utiliza en app_factory.py para configurar la aplicación

Mantiene configuraciones reutilizables para la aplicación Flask.

## Archivo index.html

- Extiende el template base
- Muestra un título de bienvenida
- Enlace a la lista de personas paginada

Template HTML que muestra la página de inicio. Usa herencia de templates.

## Archivo about.html

- Extiende el template base
- Muestra un título sobre la página Acerca De
- Botón para regresar al Inicio

Template HTML que muestra la página Acerca De. Usa herencia de templates.

## Archivo contact.html

- Extiende el template base
- Muestra formulario para contacto
- Maneja envío y validación con Flask-WTF

Template HTML que muestra el formulario de contacto. Usa herencia de templates y Flask-WTF.

## Archivo base.html

- Template base que provee la estructura común
- Navbar, block content y scripts JS comunes
- Permite heredar y extender los otros templates

Template base con la estructura HTML común a todas las páginas.

## Archivo paginado.html

- Define macro para renderizar paginación en templates
- Recibe el endpoint y los items paginados

- Muestra los enlaces de paginación

Template que contiene lógica reutilizable para renderizar la paginación. Define la macro “render_paginado” para renderizar la paginación de resultados de forma reusable.

### Macro render_paginado

Recibe dos parámetros:

- endpoint: la ruta o endpoint donde hacer la petición para obtener una página específica.
- items: el objeto Pagination de Flask-SQLAlchemy con los items paginados.

Usa los métodos y atributos de Pagination para renderizar la paginación:

- items.page: obtener el número de página actual.
- items.pages: obtener el total de páginas.
- items.has_prev: verificar si hay página anterior.
- items.has_next: verificar si hay página siguiente.
- items.iter_pages(): iterar sobre los números de página para renderizar.
- url_for(): genera las URLs dinámicamente para cada número de página.

Renderiza los links Previous, 1, 2, 3, Next etc. de forma dinámica según los datos pagination.

## Archivo modal_eliminar.html

- Define macro para mostrar modal de confirmación
- Recibe id, ruta y mensaje para mostrar
- Muestra modal con botones para confirmar o cancelar

Template que encapsula lógica reutilizable para mostrar modales de confirmación. Define la macro “confirmar_eliminar” para mostrar un modal de confirmación antes de eliminar un item.

### Macro confirmar_eliminar

Recibe tres parámetros:

- id: el id del item a eliminar.
- delete_route: la ruta o endpoint para eliminar el item.
- confirm_message: el mensaje a mostrar para confirmar.

Usa Bootstrap para renderizar el modal con los botones Confirmar y Cancelar.

El formulario dentro del modal hace POST a delete_route para eliminar el item si se confirma.

Envía el csrf_token para proteger contra ataques CSRF.

Permite crear fácilmente modales de confirmación reusables antes de eliminar datos.

## Archivo 404.html

- Extiende el template base
- Muestra mensaje de error 404 Página No Encontrada
- Link para regresar al Inicio

Template HTML que muestra mensaje de error 404.

## Archivo login.html

- Muestra formulario de login con Flask-WTF
- Campos usuario y contraseña
- Maneja envío y autenticación

Template que muestra el formulario de login. Usa Flask-WTF.

## Archivo personas/crear_persona.html

Template que muestra el formulario para crear una nueva Persona.

- Extiende el template base
- Envía el formulario al endpoint de creación
- Muestra los campos necesarios para crear una persona
- Puede rellenar los campos con datos previos

Permite llenar un formulario para crear una nueva persona en la base de datos.

## Archivo personas/personas.html

Template que muestra la lista paginada de personas con varias funciones.

- Extiende el template base
- Importa macros para paginación y modal eliminar
- Muestra tabla con los campos de Persona
- Usa paginado.render_paginado para mostrar paginación

- Usa macro modal_eliminar antes de eliminar
- Links para editar y eliminar personas

Permite listar, editar y eliminar personas.

## Archivo personas/editar_persona.html

Template que muestra el formulario para editar una Persona.

- Extiende el template base
- Obtiene la persona a editar y la envía al template
- Muestra los campos del formulario rellenados con los datos de la persona
- Envía el formulario al endpoint de edición con el ID

Permite editar los campos de una persona existente.

## Archivo auth.py

- Contiene lógica para registrar y autenticar usuarios
- Endpoints para login y logout
- Maneja inicio de sesión con Flask-Login

Módulo que implementa la autenticación de usuarios con Flask-Login.

## Archivo routes.py

- Define rutas para home, acerca de, contacto, etc.

- Renderiza los templates relacionados
- Protege rutas con @login_required

Módulo que implementa las rutas básicas con sus vistas asociadas.

## Archivo routes_personas.py

- Implementa las rutas CRUD para el modelo Persona
- Endpoints para crear, editar, borrar y listar personas
- Usa gestor_personas para la lógica de negocio

Módulo que implementa las operaciones CRUD para el modelo Persona. Contiene las rutas o endpoints que manejan las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para el modelo Persona.

Importa los módulos necesarios como Flask, gestor_personas, etc.

Define el blueprint “routes_personas” que agrupa estas rutas.

### Endpoint obtener_lista_paginada

- Ruta GET /personas
- Obtiene el parámetro page de la URL para la paginación.
- Utiliza gestor_personas para obtener una página de personas.
- Renderiza la plantilla personas.html pasándole la lista paginada.

### Endpoint editar_persona

- Ruta POST /personas//editar
- Obtiene el ID de persona a editar de la URL.
- Si es POST, obtiene los datos del formulario.
- Utiliza gestor_personas para actualizar la persona.
- Redirige a la lista o muestra errores.

### Endpoint eliminar_persona

- Ruta POST /personas/
- Obtiene el ID de persona a eliminar.
- Utiliza gestor_personas para eliminar la persona.
- Redirige a la lista.

### Endpoint crear_persona

- Ruta GET y POST /personas/crear
- Si es GET, muestra el formulario para crear.
- Si es POST, obtiene los datos del formulario.
- Utiliza gestor_personas para crear la nueva persona.
- Redirige a la lista o muestra errores.

## Archivo entities.py

- Define los modelos de la aplicación con Flask-SQLAlchemy
- Clases para Persona, Usuario, Lugar, etc.
- Relaciones entre modelos y métodos de base de datos

Módulo que contiene los modelos de la aplicación definidos con Flask-SQLAlchemy.

## Archivo base.py

- Configura la instancia de Flask-SQLAlchemy
- Clase base para los modelos
- Métodos auxiliares para paginación y operaciones comunes

Módulo con la configuración de Flask-SQLAlchemy y métodos base para los modelos.

## Archivo gestor_personas.py

- Lógica de negocio y validaciones para operaciones con Personas

- Métodos para crear, editar, borrar y listar personas
- Se utiliza en routes_personas.py

Módulo que encapsula la lógica de negocio reutilizable de Personas. Contiene la lógica de negocio reutilizable para las operaciones con Personas.

Define la clase gestor_personas que encapsula esta lógica.

### Método obtener_pagina

- Utiliza el método de clase obtener_paginado del modelo Persona.
- Obtiene una página de personas según número de página y cantidad por página.

### Método obtener

- Obtiene una persona por su ID.
- Retorna la persona o error si no existe.

### Método editar

- Obtiene la persona a editar por ID.
- Valida que los campos obligatorios existan.
- Valida email y fecha única si se incluyen.
- Edita los campos enviados de la persona.
- Guarda los cambios con el método guardar().
- Retorna resultado de la operación.

### Método eliminar

- Obtiene la persona por ID.
- Utiliza el método borrar() para eliminarla.
- Retorna resultado de la operación.

### Método crear

- Valida que los campos obligatorios existan.

- Valida email y fecha únicos.
- Crea una nueva instancia de Persona con los datos.
- Guarda la nueva persona con el método guardar().
- Retorna resultado de la operación.

## Archivo gestor_lugares.py

- Lógica de negocio para operaciones con lugares

- Consulta lugares por diferentes criterios
- Se utiliza en la API de lugares

Módulo con lógica de negocio reutilizable para operaciones con lugares. Contiene la lógica de negocio reusable para lugares.

Define la clase GestorLugares.

### Método consultar()

- Crea consulta SQLAlchemy a la tabla Lugar.
- Filtra por país, provincia, etc. según parámetros.
- Une con tablas relacionadas según criterios.
- Ejecuta la consulta y retorna lugares.

Encapsula la consulta de lugares con varios criterios de búsqueda.

Se reutiliza desde el API para proveer la funcionalidad de búsqueda.

En resumen, se separa la lógica de negocio de lugares en un gestor reusable, mientras que el API expone esta funcionalidad via HTTP.

## Archivo personas.py

- Implementa el recurso REST de personas
- GET, POST, DELETE para el modelo Persona
- Usa gestor_personas para la lógica

Recurso API REST para el modelo Persona.

## Archivo lugares.py

- Implementa recurso REST de lugares
- GET para consultar lugares con criterios

- Usa gestor_lugares para la lógica

Recurso API REST para consultar lugares con criterios. Este módulo define el recurso API REST para lugares.

Importa:

- Flask-RESTful: para crear recursos API REST.
- flask_login: para autenticación.
- gestor_lugares: lógica de negocio de lugares.

Define la clase LugaresResource que hereda de Resource de Flask-RESTful.

### Método GET

- Obtiene parámetros de consulta como país, provincia, etc.
- Utiliza gestor_lugares para consultar los lugares según criterios.
- Convierte los lugares a JSON con serialize()
- Retorna JSON con los lugares consultados.

Permite buscar lugares por diferentes campos utilizando el gestor_lugares.

## Archivo gestor_comun.py

- Clases base para gestores de entidades
- Métodos y lógica común reutilizable
- Validaciones comunes de emails, fechas, etc.

Módulo con lógica y clases base comunes para los gestores de entidades.

En resumen, se tienen:

- Archivos app.py y app_factory.py para crear la aplicación Flask.
- Templates HTML para las vistas.
- Módulos Python para rutas, modelos, lógica de negocio, APIs, etc.
- Varias librerías como Flask-SQLAlchemy, Flask-Login, Flask-WTF, etc.

## Protección CSRF (Cross-Site Request Forgery)

La protección CSRF (Cross-Site Request Forgery) sirve para prevenir ataques en los que se envían peticiones no autorizadas desde otro sitio web hacia nuestra aplicación web.

Algunos ejemplos de ataques que previene:

- Un sitio web malicioso que envía una petición POST a nuestro sitio para crear un nuevo usuario sin consentimiento del usuario.
- Un formulario oculto en otra página que envía una petición POST para transferir fondos en un sitio bancario sin que el usuario lo sepa.
- Un botón oculto que aprovecha que el usuario ya inició sesión y envía peticiones GET para cambiar su email o contraseña.

Para prevenir estos ataques, la protección CSRF requiere que cualquier petición POST/PUT/DELETE venga con un token CSRF que solo la aplicación conoce y valida.

Este token se genera en el formulario cuando se renderiza y debe coincidir con el que espera la aplicación al recibir la petición.

Así se asegura que la petición viene de una página controlada por la aplicación y no de un sitio externo malicioso.

CSRF protege contra la falsificación de peticiones desde otros sitios y previene que se ejecuten acciones no autorizadas en nuestra aplicación web.

## Python

Se utiliza la extensión Flask-WTF para protección CSRF.

Se crea una instancia de CSRFProtect y se inicializa con la app:

```
from flask_wtf.csrf import CSRFProtect

csrf = CSRFProtect()

csrf.init_app(app)
```

Esto activa la protección CSRF automáticamente en todos los formularios.

## HTML

En los templates con formularios se agrega:

```
<form>
    <!-- Campos del formulario -->

    <input type="hidden" name="csrf_token" value="{{ csrf_token() }}">

</form>
```

Esto renderiza un campo oculto con el token CSRF actual.

## Validación

Al recibir el formulario, Flask-WTF valida que:

- El campo csrf_token exista y sea válido.
- El valor coincida con el token esperado.

Si no coinciden, se produce un error CSRF 400.

Esto protege contra ataques que envíen formularios falsificados desde otros sitios.

## Ventajas

- Protección CSRF automática en todos los formularios.
- No se requiere código adicional en las vistas.
- Fácil de implementar en HTML con {{ csrf_token() }}
- Mayor seguridad sin comprometer facilidad de uso.

Flask-WTF y csrf_token() proveen una protección CSRF simple y efectiva, previniendo ataques comunes en aplicaciones web.

## Uso en la solución:

Aquí está la explicación del uso de CSRF basada en el código específico de la solución propuesta:

### app_factory.py

Se crea una instancia de CSRFProtect y se inicializa con la aplicación Flask:

```
from flask_wtf.csrf import CSRFProtect

...

csrf = CSRFProtect()

csrf.init_app(app)
```

Esto activa la protección CSRF en toda la aplicación.

### routes_personas.py

Se importa CSRFProtect:

```
from flask_wtf.csrf import CSRFProtect

...

csrf = CSRFProtect()
```

### templates/personas/crear_persona.html

El formulario de crear persona tiene un campo hidden con el token:

```
<form method="POST">

  <input type="hidden" name="csrf_token" value="{{ csrf_token() }}">

</form>
```

### templates/personas/editar_persona.html

El formulario de editar persona también tiene el campo CSRF:

```
<form method="POST">

  <input type="hidden" name="csrf_token" value="{{ csrf_token() }}">

</form>
```

De esta manera, todos los formularios quedan protegidos contra ataques CSRF.

Cuando se reciben los formularios en los endpoints de crear y editar persona, Flask-WTF validará que el csrf_token sea válido antes de procesar la petición, evitando así la falsificación de solicitudes.

Esta implementación aprovecha la integración directa entre Flask-WTF y CSRFProtect para una protección CSRF simple y efectiva.