1. Autenticación y gestión de sesiones

- **Flask-Login**
    - Gestión centralizada de sesiones de usuario.
    - `login_required` en rutas críticas garantiza que sólo usuarios autenticados accedan.
    - `login_manager.login_view = 'auth.login'` redirige automáticamente al login si no hay sesión activa.
- **Timeout de sesión**

Middleware en `app.before_request` que cierra la sesión tras 2 horas de inactividad:

```
if elapsed > timedelta(hours=2):
    logout_user()
```

    - Previene abuso de sesiones olvidadas en dispositivos compartidos.
- **Recuperación de usuario**
    - `@login_manager.user_loader` carga el usuario desde la base mediante su ID.

2. Autorización por roles

- **Decorador `@require_role(...)`**
    - Comprueba `current_user.rol` contra una lista de roles permitidos.
    - Retorna **403 Forbidden** si no tiene permiso.
- **Context processor**
    - Inyecta en las plantillas la función `has_perm(key)` y `current_role`, para mostrar/ocultar elementos UI según permisos.

3. Protección CSRF

- **Flask-WTF / WTForms**
    - Cada formulario generado con `{{ form.hidden_tag() }}` incluye token CSRF.
    - Validación automática en `form.validate_on_submit()`.
    - Asegúrate de que `app.config['WTF_CSRF_ENABLED'] = True` y de usar siempre `POST` para acciones mutativas.

4. Gestión de contraseñas y tokens

- **Hashing de contraseñas**
    - La entidad `User` debe usar un algoritmo seguro (bcrypt/argon2) al almacenar contraseñas.
    - No guardas contraseñas en texto plano.
- **Restablecimiento de contraseña**
    - Tokens únicos de un solo uso enviados por correo.
    - Caducidad del token configurada (p.ej. 1 hora).

5. Envío de correos seguro

- **Flask-Mail con TLS**

Variables en `.env`:

```
MAIL_USE_TLS=True
MAIL_PORT=587
MAIL_SERVER=smtp.gmail.com
MAIL_USERNAME=...
MAIL_PASSWORD=...
```

    - Conexiones cifradas; nunca incluir credenciales en el repositorio.

6. Validación de datos de entrada

- **WTForms Validators**
  - Campos `DataRequired`, `Email()`, `Length()`, `NumberRange()`, etc., en todos los formularios.
  - Validaciones personalizadas en modelos (`@validates('emails')`, `@validates('promedio')`).
- **Servicio de documentos**
  - Solo extensiones permitidas (`{'pdf','doc','docx'}`) en `validar_documentos()`.
  - Uso de `werkzeug.utils.secure_filename()` para sanear nombres.

7. Seguridad en ficheros subidos

- **Directorio aislado**
  - Cada solicitante tiene su propia carpeta: `UPLOAD_FOLDER_SOLICITANTES/<solicitante_id>/`.
  - `send_from_directory()` con `@login_required` y `@require_role` controla el acceso.

8. Protección de la base de datos

- **SQLAlchemy + Flask-Migrate**
  - Evita concatenar SQL a mano, utiliza ORM y parámetros ligados.
  - Migraciones gestionadas con Alembic, versionadas para reproducir esquemas.
- **Índices y constraints**
  - Primary keys y foreign keys para integridad referencial.
  - Eliminado el `UNIQUE` en correo/documento de solicitantes para permitir reaplicaciones controladas.

9. Auditoría y logging

- **Auditoría de cambios**
    - Decoradores/event listeners (`auditar_cambio`) que registran en tabla de auditoría cada operación CRUD.
    - Campos: usuario, timestamp, tabla, tipo de operación y datos antes/después.
- **Logging de errores**
    - Configurado con `logging.basicConfig` y `StreamHandler`.
    - Nivel configurable con `LOG_LEVEL` (INFO/DEBUG/ERROR).

10. Buenas prácticas de configuración

- **Variables de entorno**
    - `.env` para secretos, nunca versionada.
    - `config.py` lee variables sensibles.
- **Despliegue**
    - Servidor WSGI (Gunicorn/uWSGI) detrás de proxy inverso (NGINX) con HTTPS.
    - HTTP Strict Transport Security (HSTS).