MANUAL TÉCNICO DE ACADEMYREPORT

**Índice**

[1. Introducción 4](#_Toc191913291)

[1.1. Objetivo del Documento 4](#_Toc191913292)

[1.2. Alcance del Proyecto 4](#_Toc191913293)

[2. Descripción General de AcademyReport 5](#_Toc191913294)

[2.1. Propósito del Sistema 5](#_Toc191913295)

[2.2. Tecnologías Utilizadas 5](#_Toc191913296)

[2.3. Arquitectura del Proyecto 5](#_Toc191913297)

[3. Estructura del Backend 7](#_Toc191913298)

[3.1. Organización de Archivos y Carpetas 7](#_Toc191913299)

[3.2. Configuración del Servidor Flask 7](#_Toc191913300)

[3.3. Base de Datos (MongoDB) 8](#_Toc191913301)

[3.4. Gestión de Rutas y Funcionalidades 9](#_Toc191913302)

[3.4.1. Administración de Profesores 9](#_Toc191913303)

[3.4.2. Administración de Asignaturas 12](#_Toc191913304)

[3.4.3. Exportación de Datos (Excel y PDF) 14](#_Toc191913305)

[3.4.4. Autocompletado y Validaciones 20](#_Toc191913306)

[3.4.5. Manejo de Historial de Exportaciones 24](#_Toc191913307)

[4. Estructura del Frontend 25](#_Toc191913308)

[4.1. Organización de Archivos en static y templates 25](#_Toc191913309)

[4.2. Estilos y Diseño del Proyecto 26](#_Toc191913310)

[4.3. Funcionalidad de la Interfaz de Usuario 27](#_Toc191913311)

[4.3.1. Formulario de Registro y Edición de Profesores 27](#_Toc191913312)

[4.3.2. Formulario de Registro de Asignaturas 28](#_Toc191913313)

[4.3.3. Selección Dinámica de Grupos y Carreras 28](#_Toc191913314)

[4.3.4. Botones y Navegación en la Interfaz 29](#_Toc191913315)

[5. Funcionamiento del Sistema 30](#_Toc191913316)

[5.1. Flujo de Trabajo en el Registro de Profesores 30](#_Toc191913317)

[5.2. Flujo de Exportación de Reportes 31](#_Toc191913318)

[5.3. Validaciones Implementadas en la Aplicación 32](#_Toc191913319)

[5.4. Administración de Imágenes y Documentos 33](#_Toc191913320)

[6. Manejo y Seguridad de la Aplicación 35](#_Toc191913321)

[6.1. Sistema de Autenticación y Roles de Usuario 35](#_Toc191913322)

[6.1.1. Registro y Validación de Usuarios 35](#_Toc191913323)

[6.2. Protección de Rutas y Accesos 36](#_Toc191913324)

[6.3. Control de Errores y Manejo de Excepciones 36](#_Toc191913325)

[6.3.1. Manejo de Errores en Flask 37](#_Toc191913326)

[6.3.2. Manejo de Excepciones en la Base de Datos 37](#_Toc191913327)

[7. Instalación y Configuración del Proyecto 39](#_Toc191913328)

[7.1. Requisitos Previos 39](#_Toc191913329)

[7.1.1 Instalar LibreOffice en el servidor/local 39](#_Toc191913330)

[7.2. Instalación en un Entorno Local 40](#_Toc191913331)

[7.3. Despliegue en un Servidor 40](#_Toc191913332)

[8. Conclusiones y Recomendaciones 44](#_Toc191913333)

# 1. Introducción

AcademyReport es un sistema desarrollado para la gestión eficiente de información académica, optimizando la administración de docentes, asignaturas y horarios. Este manual técnico proporciona una guía detallada sobre la estructura, funcionalidades y configuración del sistema, facilitando su mantenimiento y futuras actualizaciones.

Público Objetivo del Manual:

* Desarrolladores - Comprender la arquitectura y modificar funcionalidades.
* Administradores - Gestionar usuarios, asignaturas y reportes.
* Equipo de soporte - Resolver errores y configurar el sistema.

## 1.1. Objetivo del Documento

Este documento tiene como objetivo proporcionar una guía técnica detallada del sistema AcademyReport, incluyendo su arquitectura, funcionamiento, configuración y mantenimiento. Está dirigido a desarrolladores, administradores del sistema y cualquier persona encargada del mantenimiento y mejora del software.

Aspectos Clave del Manual:

* Arquitectura del sistema (backend, frontend, base de datos).
* Configuración e instalación en distintos entornos.
* Explicación del código con ejemplos y diagramas.
* Guía para la administración de datos dentro del sistema.

## 1.2. Alcance del Proyecto

AcademyReport es una aplicación diseñada para la gestión de datos académicos, permitiendo administrar profesores, asignaturas, horarios y reportes en formato Excel y PDF. Su implementación abarca la Facultad de Ingeniería en diversas carreras como Industrial, Sistemas Computacionales, Informática, Electrónica, Electromecánica y Administración.

Principales Funcionalidades del Sistema:

* Gestión de Profesores Permite registrar y actualizar información docente.
* Asignaturas y Horarios Facilita la asignación de materias y horarios por carrera.
* Exportación de Reportes Generación de informes en Excel y PDF.
* Autenticación Segura Control de acceso mediante roles de usuario.

# 2. Descripción General de AcademyReport

AcademyReport es un sistema diseñado para gestionar información académica de manera eficiente y estructurada. Su principal objetivo es automatizar la administración de profesores, asignaturas y horarios, además de facilitar la exportación de datos en formatos como Excel y PDF.

¿Qué resuelve AcademyReport?

* Agiliza la gestión de datos académicos.
* Facilita la exportación y generación de reportes.
* Evita errores manuales en la asignación de horarios.
* Permite un control estructurado de la información.

## 2.1. Propósito del Sistema

El propósito principal de AcademyReport es proporcionar una plataforma integral para la administración académica en diversas carreras.

Objetivos Específicos:

* Registro de profesores - Almacenar información de docentes en MongoDB.
* Asignación de asignaturas - Relacionar docentes con materias específicas.
* Gestión de horarios - Evitar solapamientos y distribuir carga horaria.
* Exportación de datos - Generar reportes detallados en Excel y PDF.
* Autenticación y seguridad - Controlar el acceso a la plataforma.

## 2.2. Tecnologías Utilizadas

AcademyReport utiliza un stack tecnológico robusto, permitiendo una integración eficiente entre el backend, frontend y la base de datos.

Tecnologías Principales:

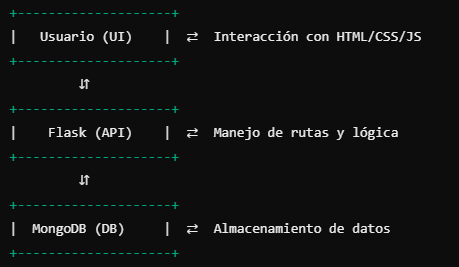
* Backend - Flask (Python)
* Frontend - HTML, CSS, JavaScript (Choices.js)
* Base de Datos - MongoDB
* Exportación de Reportes - Pandas, XlsxWriter, Openpyxl
* Autenticación - Flask-Login

## 2.3. Arquitectura del Proyecto

AcademyReport sigue una arquitectura MVC (Modelo-Vista-Controlador) estructurada de la siguiente manera:

* Modelo: Definido en MongoDB para profesores, asignaturas y administrativos.
* Vista: Implementada en HTML y CSS.
* Controlador: Flask gestiona las rutas y la lógica del sistema.

Diagrama de la Arquitectura General



# 3. Estructura del Backend

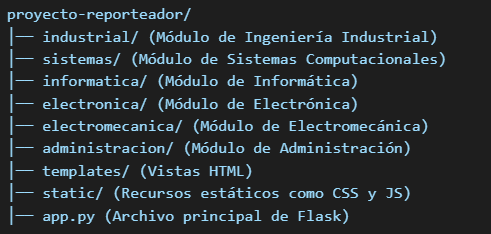
AcademyReport cuenta con un backend desarrollado en Flask, diseñado para manejar la gestión de datos, autenticación y exportación de reportes.

Funciones del Backend:

* Manejo de rutas y lógica con Flask.
* Uso de MongoDB como base de datos NoSQL.
* Exportación de datos en Excel y PDF.
* Validaciones y seguridad con Flask-Login.

## 3.1. Organización de Archivos y Carpetas

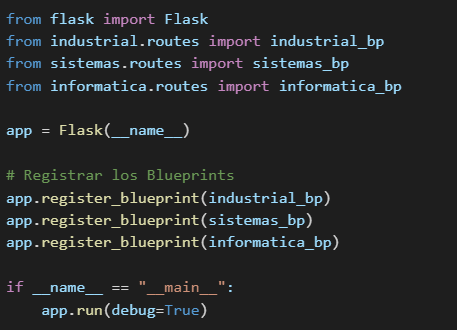
El proyecto se organiza modularmente, permitiendo mantener el código estructurado y escalable:



Cada módulo contiene sus propias rutas (routes), plantillas HTML (templates) y archivos estáticos (static), lo que permite una gestión independiente de cada carrera.

## 3.2. Configuración del Servidor Flask

Flask es el framework utilizado para manejar el backend. En app.py se configuran las rutas y se inicializan los diferentes Blueprints de cada carrera. Esto permite que cada módulo tenga su propio espacio de trabajo:

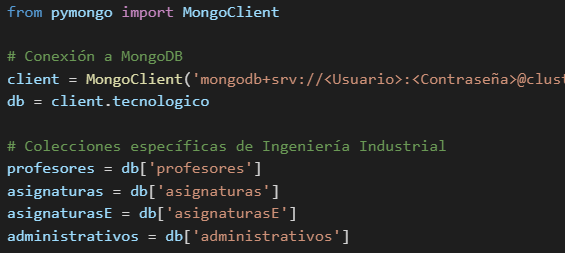


Cada Blueprint define las rutas específicas para cada módulo, lo que permite que las funciones CRUD (Crear, Leer, Actualizar y Eliminar) sean independientes por carrera.

## 3.3. Base de Datos (MongoDB)

MongoDB es la base de datos utilizada en AcademyReport. Se organiza en múltiples colecciones para facilitar la gestión de datos. A continuación, se definen algunas de las colecciones principales:

* **profesores**: Contiene la información de cada docente.
* **asignaturas**: Contiene las materias disponibles por carrera (administracion\_asignaturas, electronica\_asignaturas, etc ).
* **asignaturasE**: Contiene las materias especiales disponibles por carrera (administracion\_asignaturasE, electronica\_asignaturasE, etc ).
* **administrativos**: Contiene la información de personal administrativo.



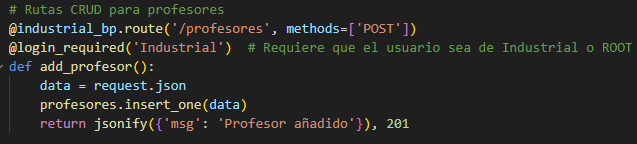
Cada colección almacena un tipo de entidad. Por ejemplo, profesores contiene datos sobre los docentes, mientras que asignaturas almacena las materias disponibles por carrera.

## 3.4. Gestión de Rutas y Funcionalidades

### 3.4.1. Administración de Profesores

Se implementan rutas CRUD para la gestión de profesores:

* GET /profesores → Obtiene la lista de profesores.
* POST /profesores → Agrega un nuevo profesor.
* PUT /profesores/<id> → Edita un profesor existente.
* DELETE /profesores/<id> → Elimina un profesor.

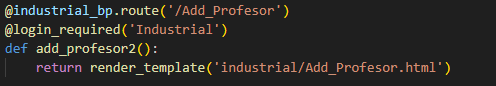


Método: POST → Permite agregar nuevos registros de profesores.

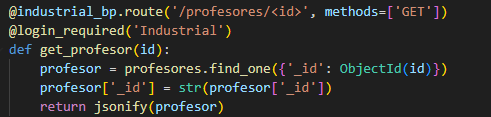
Seguridad: @login\_required('Industrial') → Solo usuarios autenticados en Industrial pueden acceder.

Proceso:

* Recibe los datos de un profesor en formato JSON.
* Inserta el registro en la colección profesores de MongoDB.
* Devuelve un mensaje de confirmación con código 201 (Creado).



* Muestra el formulario HTML donde se pueden ingresar los datos de un nuevo profesor.
* Usa render\_template() para cargar el archivo Add\_Profesor.html dentro de la carpeta templates/industrial/.
* Solo los usuarios autenticados en Industrial pueden acceder.

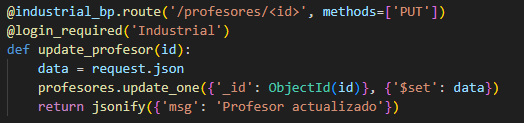


Método: GET → Recupera un profesor específico desde la base de datos.

Parámetro: <id> → Recibe el ID del profesor como parámetro en la URL.

Proceso:

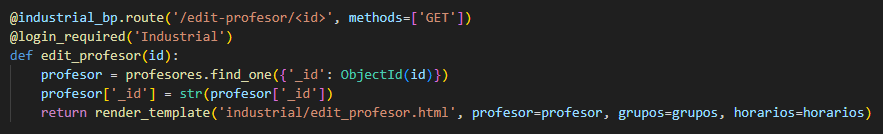
* Busca en la base de datos un profesor con el ID proporcionado.
* Convierte su \_id a un string (MongoDB usa ObjectId).
* Devuelve el resultado en formato JSON.



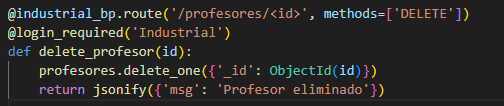
Método: PUT → Permite actualizar un registro existente.

Proceso:

* Recibe datos en JSON.
* Encuentra al profesor con el ID especificado.
* Actualiza solo los campos enviados mediante {'$set': data}.
* Retorna un mensaje confirmando la actualización.



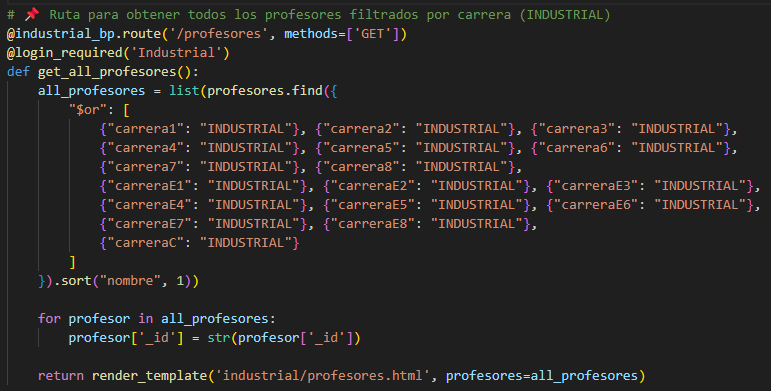
* Muestra el formulario HTML de edición con la información actual del profesor.
* grupos y horarios se pasan como variables para poblar listas desplegables.



Método: DELETE → Permite eliminar un registro de la base de datos.

Proceso:

* Busca el profesor con el ID recibido.
* Lo elimina con delete\_one().
* Devuelve un mensaje confirmando la eliminación.



Método: GET → Obtiene todos los profesores de la carrera Industrial.

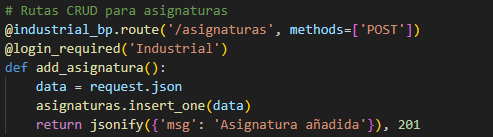
Consulta:

* Filtra profesores en las columnas carrera1 a carrera8.
* Ordena los resultados por nombre (sort("nombre", 1)).
* Convierte \_id a string antes de enviarlo a la vista.

### 3.4.2. Administración de Asignaturas

Similar a la gestión de profesores, se implementan rutas para las asignaturas que permiten:

* Registrar nuevas asignaturas.
* Editar y eliminar asignaturas existentes.
* Implementar autocompletado basado en la carrera seleccionad

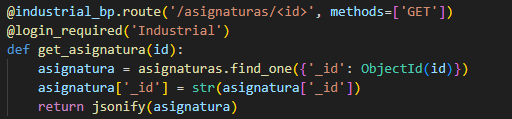


Método: POST → Permite agregar nuevas asignaturas.

Seguridad: @login\_required('Industrial') → Solo usuarios autenticados pueden acceder.

Proceso:

* Recibe los datos de una asignatura en formato JSON.
* Inserta el registro en la colección asignaturas de MongoDB.
* Devuelve un mensaje de confirmación con código 201 (Creado).

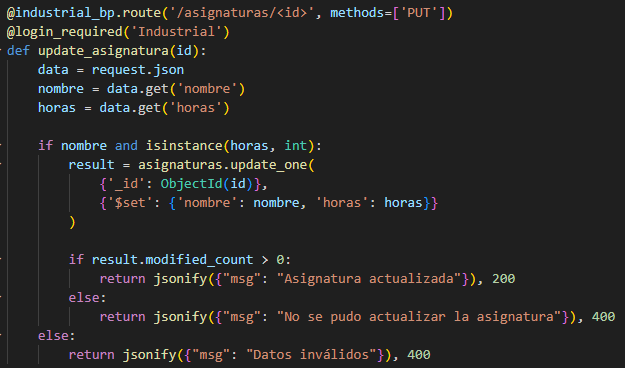


Método: GET → Recupera una asignatura específica desde la base de datos.

Parámetro: <id> → Recibe el ID de la asignatura en la URL.

Proceso:

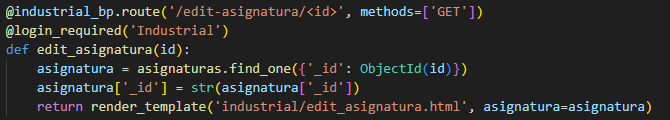
* Busca en la base de datos una asignatura con el ID proporcionado.
* Convierte \_id a un string (MongoDB usa ObjectId).
* Devuelve el resultado en formato JSON.



Método: PUT → Permite actualizar una asignatura existente.

Validaciones:

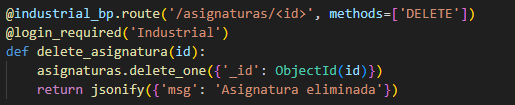
* Verifica que nombre sea una cadena y que horas sea un número entero.
* Si los datos son válidos, actualiza el registro con update\_one().
* Si la actualización fue exitosa, responde con 200 (OK); de lo contrario, devuelve 400 (Error de solicitud).



Método: GET → Muestra el formulario HTML de edición de asignaturas.

Proceso:

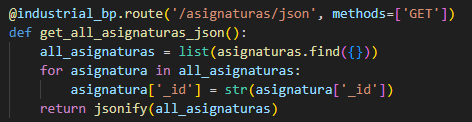
* Busca la asignatura con el ID especificado.
* Convierte \_id a string para su uso en la plantilla HTML.
* Renderiza edit\_asignatura.html, donde el usuario puede modificar los datos.



Método: DELETE → Permite eliminar un registro de la base de datos.

Proceso:

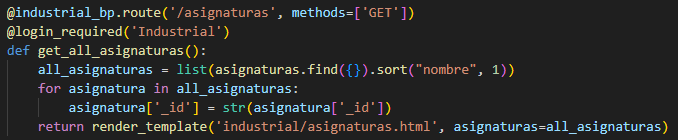
* Busca la asignatura con el ID recibido.
* La elimina con delete\_one().
* Devuelve un mensaje confirmando la eliminación.



Método: GET → Obtiene todas las asignaturas en formato JSON.

Proceso:

* Recupera todas las asignaturas de la base de datos.
* Convierte \_id a string.
* Retorna la lista de asignaturas en formato JSON.



Método: GET → Obtiene todas las asignaturas pero renderiza un archivo HTML en lugar de devolver JSON.

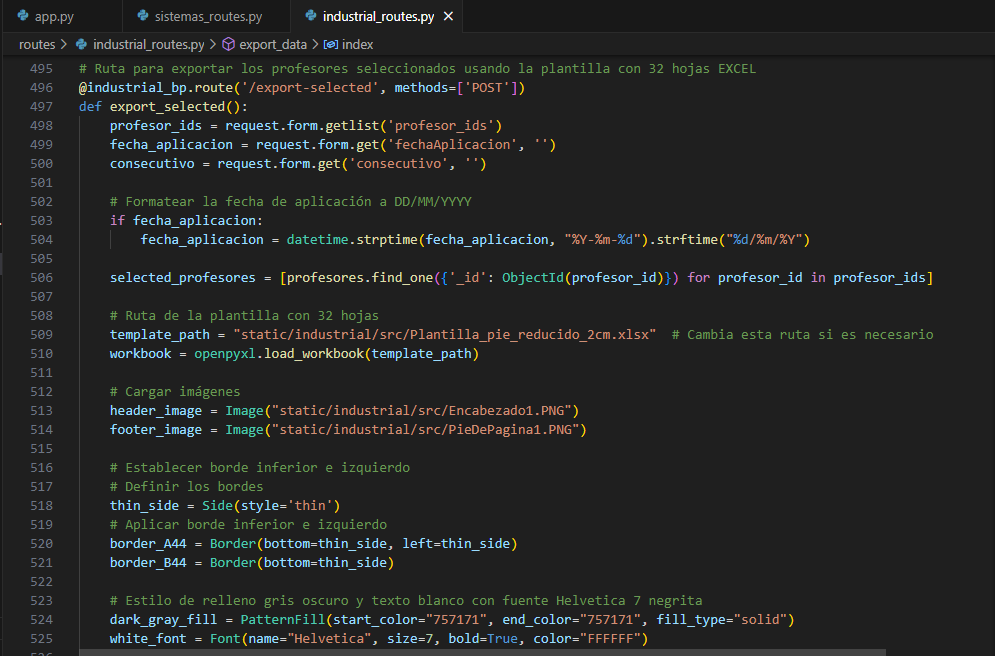
Uso: Se utiliza para mostrar la lista de asignaturas en la interfaz de usuario.

### 3.4.3. Exportación de Datos (Excel y PDF)

El sistema permite la exportación de reportes mediante Pandas y XlsxWriter, y para Formatos predefinidos PDF con reportlab.

Excel:

Se exporta la información de los profesores en la carrera de Sistemas Computacionales a un archivo Excel basado en una plantilla predefinida de 32 hojas. Dado que la lógica es la misma para todas las carreras, este código sirve como referencia general para la exportación de profesores en cualquier especialidad.

Recibir IDs de profesores a exportar (POST):

* Obtiene la lista de profesores seleccionados desde un formulario HTML.
* Recibe información como fecha de aplicación y consecutivo.

Cargar plantilla Excel:

* Usa openpyxl para abrir una plantilla con 32 hojas ubicada en:

template\_path = "static/sistemas/src/Plantilla\_pie\_reducido\_2cm.xlsx",

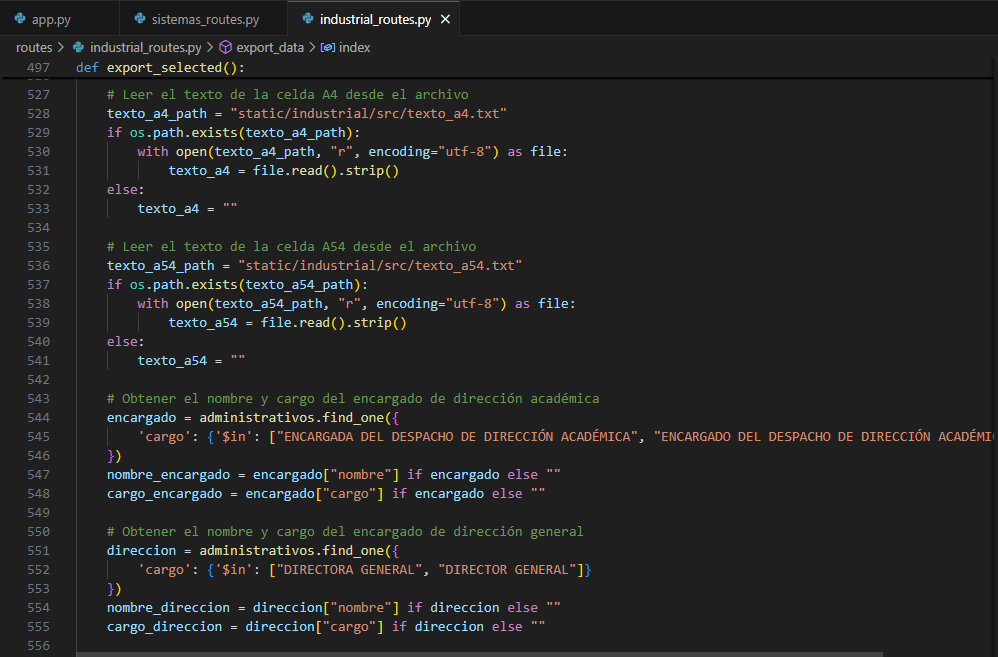
* La plantilla se usa para formatear los datos y mantener una estructura estandarizada.

Insertar imágenes en la plantilla:

* Agrega el encabezado y pie de página usando imágenes predefinidas:

header\_image = Image("static/sistemas/src/Encabezado1.PNG")

footer\_image = Image("static/sistemas/src/PieDePagina1.PNG")

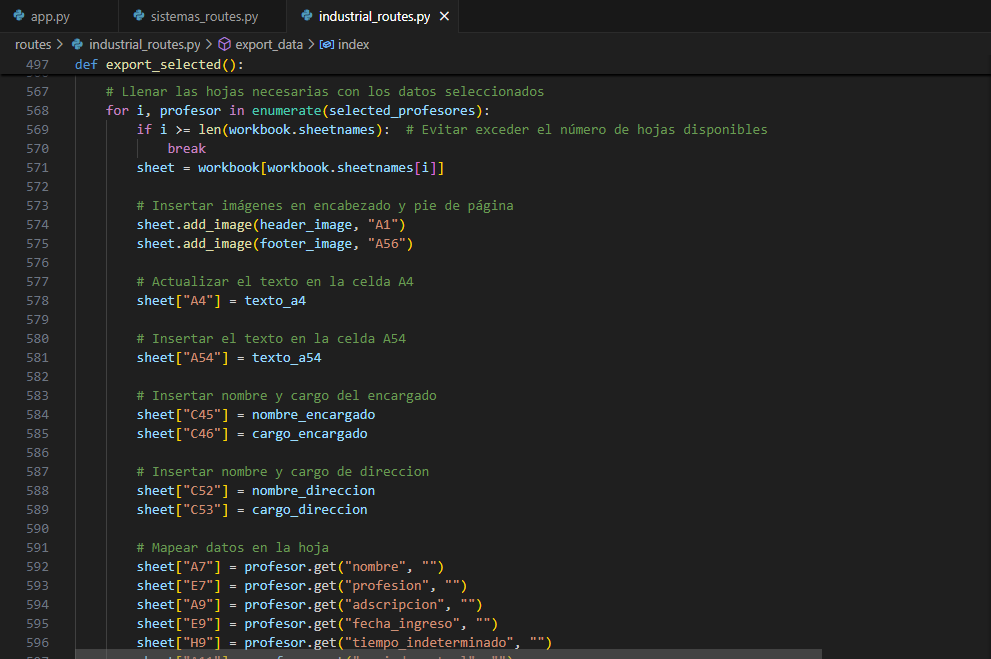


Cargar texto desde archivos externos:

* Carga información específica desde archivos de texto para las celdas A4 y A54, facilitando la actualización dinámica:

texto\_a4\_path = "static/sistemas/src/texto\_a4.txt"

texto\_a54\_path = "static/sistemas/src/texto\_a54.txt"



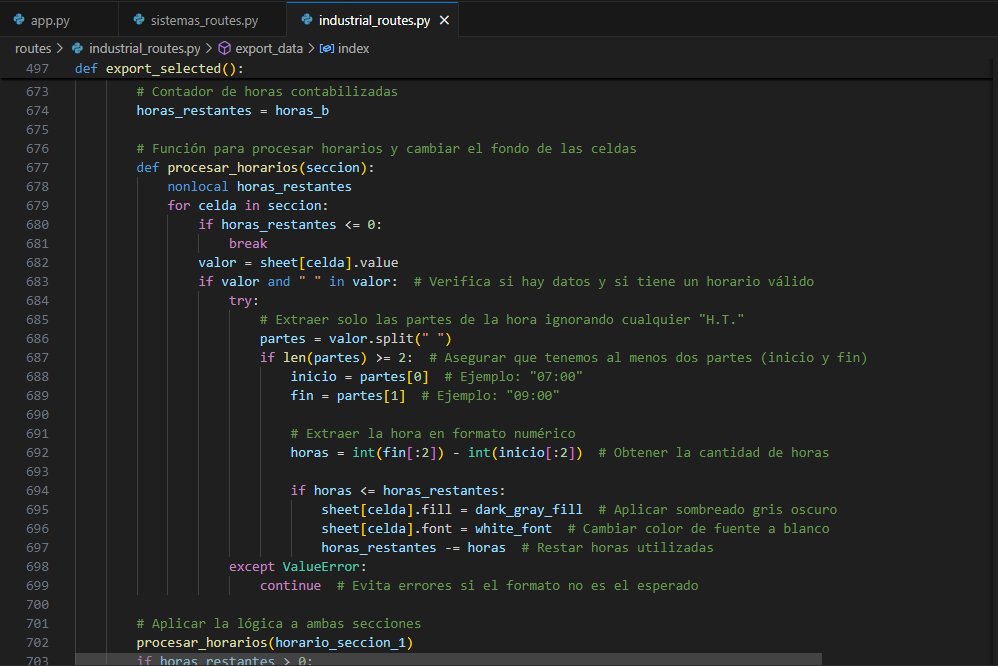
Llenar la hoja con información del profesor:

* Para cada profesor seleccionado, asigna datos como:

Nombre, profesión, adscripción, fecha de ingreso, horas asignadas.

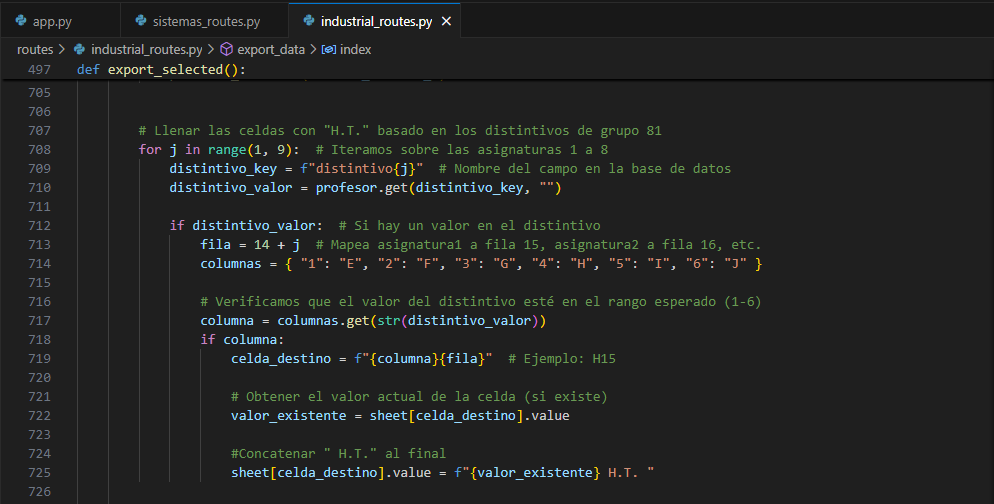
Asignaturas, grupos, horarios, cargo y vigencia.

* Además que los datos se insertan en celdas específicas de la hoja de Excel.



Procesar horarios y evitar solapamientos:

* Verifica horarios de clase y carga administrativa, asegurando que no haya conflictos en las asignaciones.

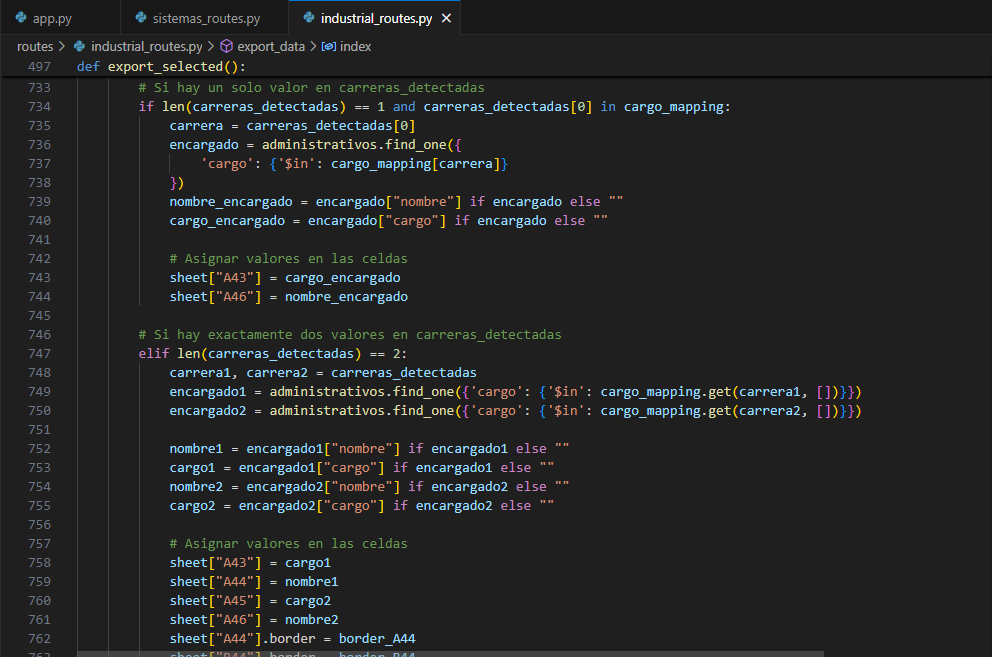


Asignación de H.T. (Horas Teóricas) a los distintivos de grupo:

* Si un grupo tiene un distintivo 81 o 71, se marca la celda correspondiente con " H.T. ":

if distintivo\_valor:

sheet[celda\_destino].value = f"{valor\_existente} H.T. "



Identificación de encargados de división:

* Asigna los nombres y cargos de los jefes de división de acuerdo con la ingeniería detectada:

cargo\_mapping = {

"SISTEMAS COMPUTACIONALES": ["JEFE DE DIVISIÓN DE ING. SISTEMAS COMPUTACIONALES"],

"INDUSTRIAL": ["JEFE DE DIVISIÓN DE ING. INDUSTRIAL"],...}

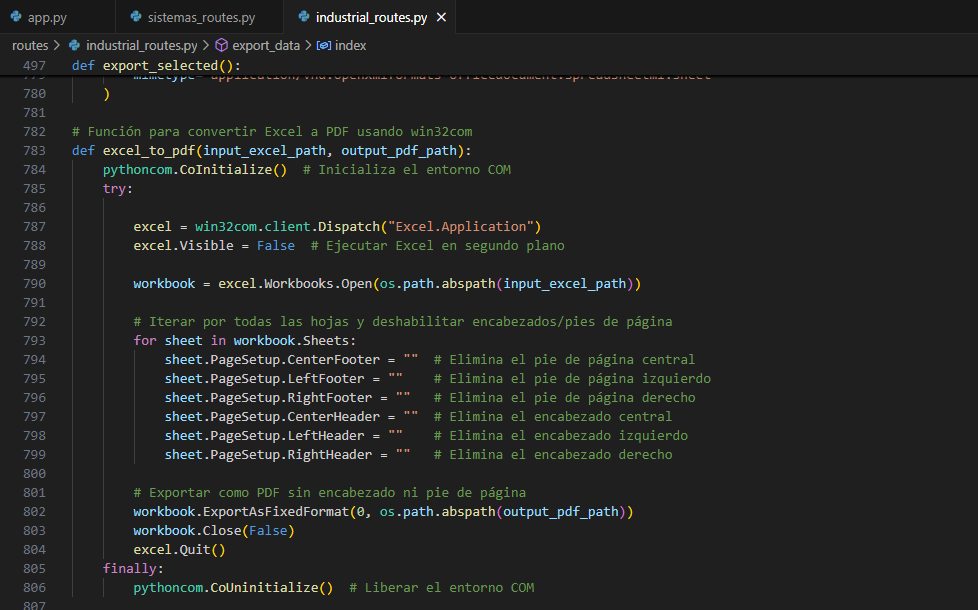
Eliminar hojas no utilizadas:

* Si el número de profesores seleccionados es menor a 32, se eliminan las hojas vacías

Guardar y enviar el archivo final:

* Se convierte el archivo en memoria para evitar almacenamiento local y se devuelve como una descarga automática

PDF:  
Se define una función para convertir un archivo Excel a PDF utilizando win32com.client en Sistemas Computacionales. La misma lógica puede aplicarse a todas las carreras con solo modificar las rutas y nombres de archivo.



Inicializa el entorno COM de Windows:

* pythoncom.CoInitialize() es necesario para manejar instancias de Microsoft Excel a través de Win32 COM.

Abre el archivo Excel en segundo plano:

* Evita interrupciones visuales ejecutando Excel de manera oculta:

excel = win32com.client.Dispatch("Excel.Application")

excel.Visible = False

Limpieza de Encabezados y Pies de Página:

* Itera por todas las hojas del Excel y elimina cualquier encabezado o pie de página, asegurando que el PDF resultante tenga formato limpio

Convierte el Excel a PDF:

* Usa el método ExportAsFixedFormat(0, path) para exportar el archivo sin necesidad de interacción manual.
* La conversión mantiene todos los estilos y formatos de las hojas.

Cierra Excel correctamente:

* Evita procesos en segundo plano no deseados liberando la memoria correctamente.

Libera el entorno COM después de la ejecución:

* Previene bloqueos de recursos en Windows.

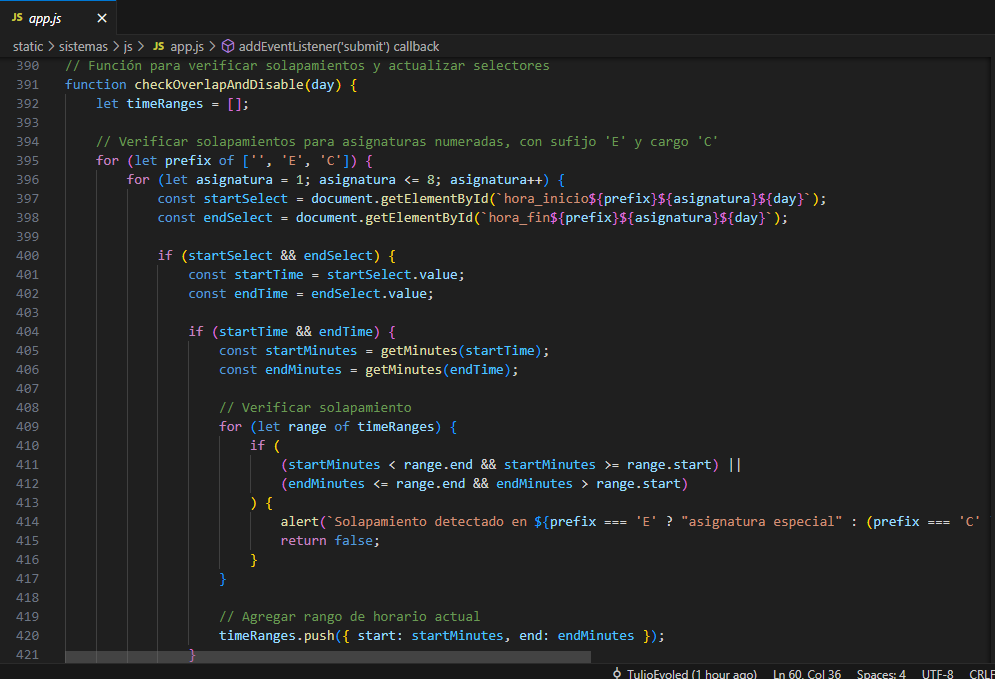
### 3.4.4. Autocompletado y Validaciones

Para facilitar la selección de asignaturas y evitar errores, se usa JavaScript con Choices.js. Esta biblioteca permite funcionalidades como:

* Búsqueda en tiempo real dentro de los <select>.
* Multiselección de opciones en algunos casos.
* Un diseño más limpio y ordenado, evitando listas desplegables extensas.
* Manejo eficiente de grandes listas de opciones, evitando sobrecargar el DOM.

Validación de Horarios y Solapamientos:

Se maneja la validación de solapamiento de horarios en la asignación de materias para profesores. Es crucial para evitar que un profesor tenga dos asignaturas al mismo tiempo y garantizar que no haya conflictos en su disponibilidad horaria.



Conversión de horas a minutos para comparación:

* Se define la función “getMinutes(timeString)”, que toma un horario en formato "HH:MM" y lo convierte a minutos totales.
* Esto permite comparar y detectar solapamientos de forma matemática.
* Ejemplo

"08:00" → 480 minutos

"10:30" → 630 minutos

Verificación de solapamientos:

* La función checkOverlapAndDisable(day) revisa todos los horarios asignados en un día específico y los compara para detectar choques.
* Se aplica a tres tipos de horarios:

Asignaturas regulares ("")

Asignaturas especiales ("E")

Carga administrativa (cargos) ("C")

* Ejemplo

Un profesor tiene Matemáticas de 08:00 a 10:00

No puede tener Física de 09:30 a 11:00, porque hay un solapamiento de 30 minutos.

Iteración sobre las asignaturas y horarios:

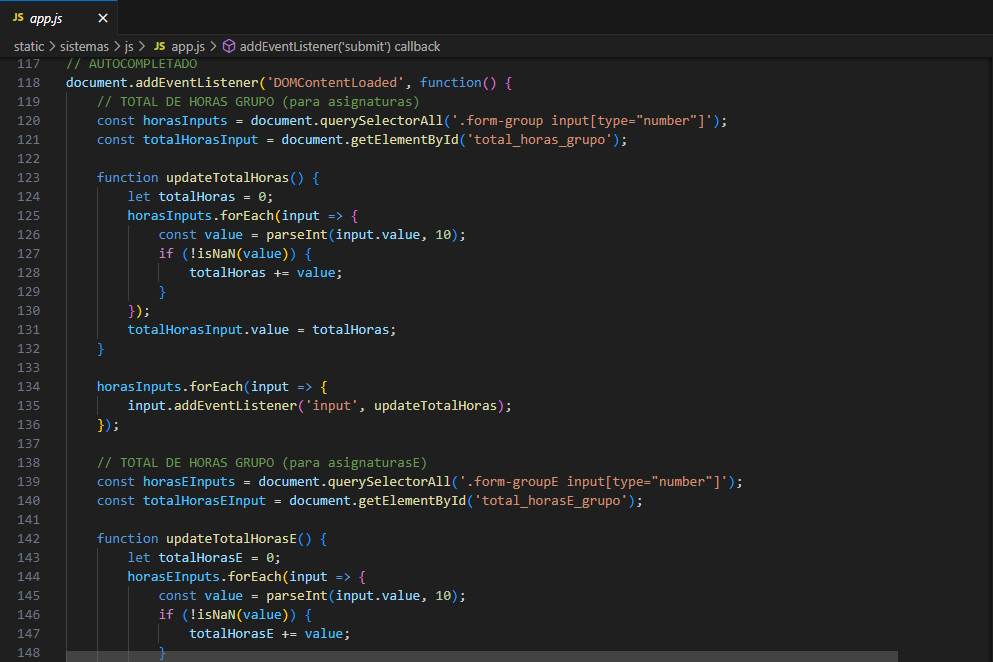
* Se recorren todas las asignaturas (1-8) y se obtienen los valores de hora de inicio y fin.
* Convierte estos horarios a minutos y verifica si hay superposición con otros horarios y si hay solapamiento.
* Se muestra una alerta indicando la asignatura, tipo y día del conflicto.

Deshabilitar horarios en conflicto:

* Si dos horarios se solapan, se deshabilitan las opciones de selección en el select del formulario y se evita que el usuario seleccione una hora que ya está en uso.

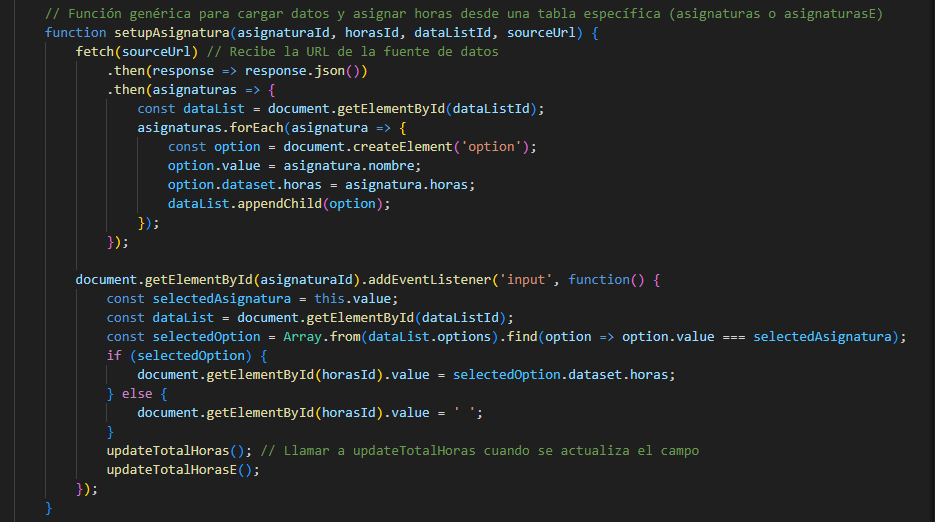
Autocompletado de Asignaturas:

Se busco automatizar el autocompletado de asignaturas y la gestión dinámica de horas en los formularios de asignación académica. Permite que los usuarios seleccionen rápidamente las asignaturas y que el sistema asigne automáticamente las horas correspondientes desde la base de datos.



Cálculo del Total de Horas:

* Se actualiza automáticamente el total de horas en los formularios cuando se introducen datos en los campos de asignaturas regulares y especiales.
* Se escuchan los cambios en los input[type="number"], sumando los valores ingresados.

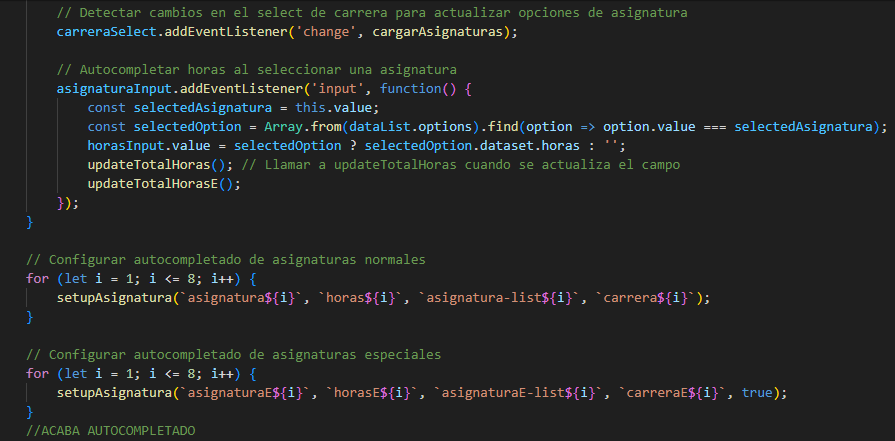


Configuración de Autocompletado:

* Usa la función setupAsignatura para obtener las asignaturas desde la base de datos y poblar los <datalist> del formulario.
* Esto permite que los usuarios escriban el nombre de la asignatura y el sistema complete la información de forma automática.

Asignación Automática de Horas:

* Cuando el usuario selecciona una asignatura, el sistema busca en el datalist la cantidad de horas predefinidas y la asigna al campo correspondiente.



Conexión con el Backend:

* Se obtiene la información de las asignaturas desde la API REST del servidor Flask y se hace una petición fetch a las rutas definidas en Flask (/sistemas/sistemas\_asignaturas/json).

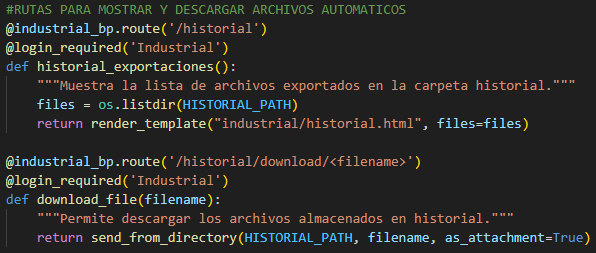
Implementación para Diferentes Tipos de Asignaturas:

* Se configura el autocompletado para asignaturas regulares y asignaturas especiales (asignaturasE).

Eventos para Navegación entre Páginas:

* Se agregan event listeners a los botones de navegación, para que redirijan a la URL correspondiente.

### 3.4.5. Manejo de Historial de Exportaciones

Los archivos exportados se almacenan en static/historial, permitiendo su consulta posterior:  


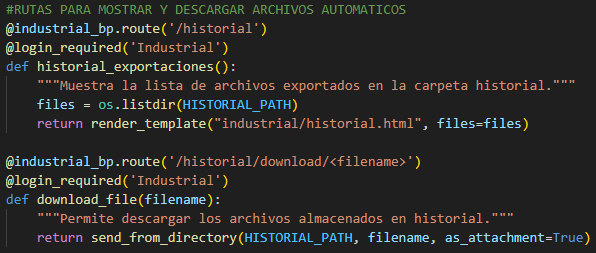
Método: GET

Seguridad:

* @login\_required('Industrial') → Restringe el acceso a usuarios autenticados de Industrial.

Funcionalidad:

* Obtiene la lista de archivos dentro de la carpeta HISTORIAL\_PATH.
* Renderiza la plantilla historial.html con la lista de archivos.



Método: GET

Seguridad:

* @login\_required('Industrial') → Solo los usuarios autenticados pueden descargar archivos.

Funcionalidad:

* Obtiene el nombre del archivo desde la URL (<filename>).
* Usa send\_from\_directory() para enviar el archivo desde la carpeta HISTORIAL\_PATH.
* La opción as\_attachment=True fuerza la descarga en lugar de abrir el archivo en el navegador.

# 4. Estructura del Frontend

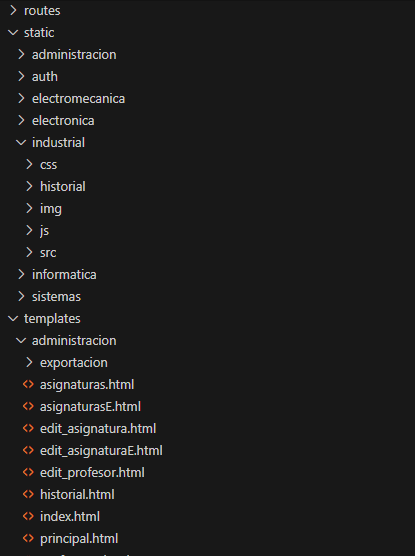
El frontend de AcademyReport es la parte visible e interactiva del sistema, donde los usuarios pueden realizar acciones como agregar profesores, asignaturas, exportar reportes y administrar información. Está diseñado con HTML, CSS y JavaScript, con interactividad mejorada mediante bibliotecas como Choices.js.

El código del frontend está estructurado dentro de dos carpetas principales:

* static/: Contiene los archivos estáticos como hojas de estilo (.css), scripts de JavaScript (.js) e imágenes.
* templates/: Contiene los archivos HTML que conforman la interfaz de usuario, organizados por módulos correspondientes a cada carrera.

## 4.1. Organización de Archivos en static y templates

El sistema sigue una organización modular, donde cada carrera tiene su propio conjunto de archivos de frontend. A continuación, se muestra la estructura de estas carpetas:

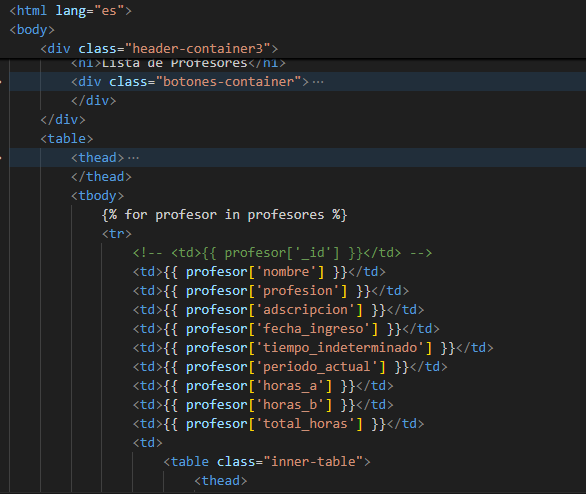


Cada módulo tiene su propio conjunto de archivos, permitiendo un mantenimiento independiente por carrera.

* static/<carrera>/css/: Contiene las hojas de estilo para definir el diseño del sistema.
* static/<carrera>/js/: Contiene los scripts de JavaScript para la interactividad.
* templates/<carrera>/: Contiene los archivos HTML que representan la interfaz de usuario.

Ejemplo de una plantilla HTML (profesores.html)

Este archivo muestra una lista de profesores y permite editarlos.



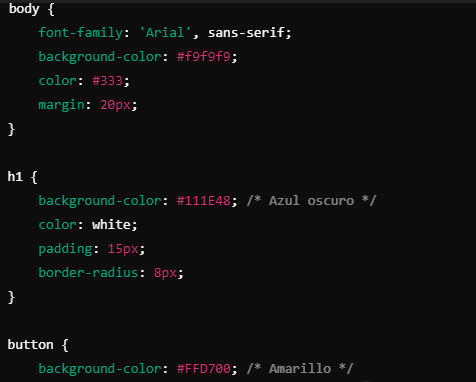
## 4.2. Estilos y Diseño del Proyecto

El diseño de AcademyReport está definido en la hoja de estilos styles.css. Se utiliza una paleta de colores institucional con fondos claros y detalles oscuros para una mejor legibilidad.

Algunas características del diseño incluyen:

* Uso de flexbox y grid para estructurar el contenido.
* Colores personalizados para cada carrera.
* Botones estilizados con efectos de hover.

Ejemplo de Estilos en styles.css



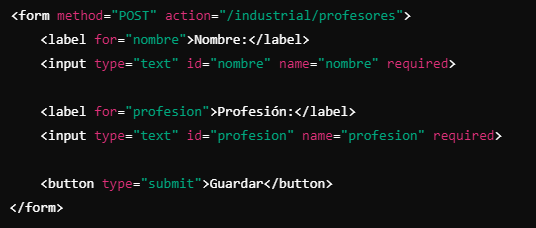
## 4.3. Funcionalidad de la Interfaz de Usuario

### 4.3.1. Formulario de Registro y Edición de Profesores

Los formularios permiten registrar y actualizar información de profesores. En este formulario se incluyen campos como:

* Nombre del profesor.
* Profesión.
* Asignación de asignaturas y horarios.

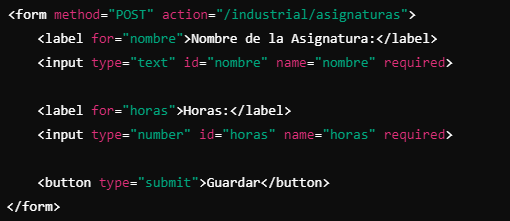
Ejemplo de Formulario de Registro



El backend se encarga de procesar esta información y almacenarla en la base de datos.

### 4.3.2. Formulario de Registro de Asignaturas

Este formulario permite registrar las asignaturas disponibles para cada carrera, asegurando que se vinculen correctamente a los profesores.



El sistema también incluye un autocompletado de asignaturas según la carrera seleccionada.

### 4.3.3. Selección Dinámica de Grupos y Carreras

Para facilitar la asignación de horarios y asignaturas, el sistema permite filtrar dinámicamente los grupos y asignaturas de acuerdo con la carrera seleccionada.

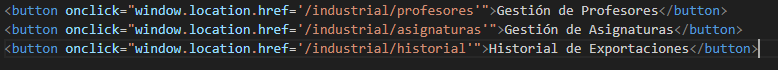
El siguiente código de JavaScript permite actualizar los grupos según la carrera elegida:



Este fragmento de código permite que, al seleccionar una carrera en un formulario, se filtren y muestren los grupos correspondientes.

### 4.3.4. Botones y Navegación en la Interfaz

El sistema tiene botones para navegar entre módulos y funcionalidades, con un diseño uniforme en todas las vistas.



Estos botones redirigen a diferentes secciones del sistema, permitiendo una navegación fluida.

# 5. Funcionamiento del Sistema

El funcionamiento del sistema AcademyReport se basa en una serie de flujos de trabajo que permiten la gestión eficiente de profesores, asignaturas y reportes académicos. Se han implementado mecanismos de validación y administración de imágenes/documentos para asegurar la integridad de los datos y la personalización de los reportes.

## 5.1. Flujo de Trabajo en el Registro de Profesores

El registro de profesores es una de las funcionalidades principales de AcademyReport. Su flujo de trabajo incluye:

1. Ingreso de Datos:

* El usuario completa un formulario con información del profesor, incluyendo nombre, profesión, adscripción, periodo actual y carga horaria.
* Se permite la asignación de hasta 8 asignaturas, con la opción de definir horarios sin superposición de clases.

2. Selección de Carrera y Asignaturas:

* Al elegir una carrera, el sistema filtra dinámicamente las asignaturas disponibles.
* Se usa un autocompletado basado en los datos existentes en la base de datos.

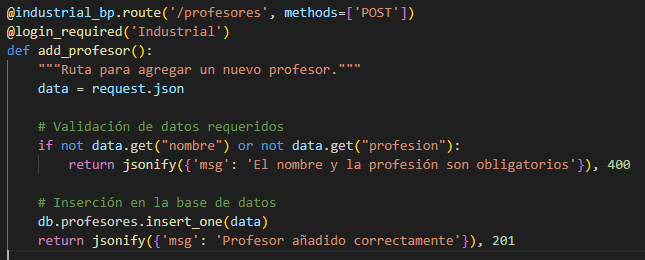
3. Registro de Horarios y Validaciones:

* Se verifica que no haya solapamiento de horarios al asignar clases a un profesor.
* Se calculan automáticamente las horas totales asignadas para asegurar que la carga horaria sea correcta.

4. Almacenamiento en Base de Datos:

* Una vez completado el formulario, los datos se almacenan en MongoDB.
* Se usa una estructura flexible que permite modificar y agregar nuevos campos sin afectar el funcionamiento general del sistema.

Ejemplo de Código (Registro de Profesores - Backend Flask)



## 5.2. Flujo de Exportación de Reportes

El sistema permite exportar datos de profesores en Excel y PDF, asegurando la generación de reportes personalizados con la información seleccionada.

Pasos del Flujo de Exportación:

1. Selección de Datos

* El usuario elige qué columnas desea exportar (Ejemplo: Nombre, Profesión, Horas Asignadas, etc.).
* Se selecciona el formato de exportación (Excel o PDF).

2. Generación del Archivo

* Se procesan los datos en pandas (para Excel) y reportlab (para PDF).
* Se formatean los datos para mantener un diseño coherente en los reportes.

3. Descarga del Reporte

* El usuario recibe el archivo generado, que se almacena temporalmente en el servidor.
* Existe un historial de exportaciones automáticas que permite acceder a reportes previos.

Ejemplo de Código (Exportación a Excel - Flask y Pandas)



Validaciones en la Exportación:

* Si el usuario no selecciona columnas, se muestra un mensaje de error.
* Se verifica que los datos no estén vacíos antes de generar el archivo.
* Se usan formatos predefinidos en los archivos exportados para mejorar la legibilidad.

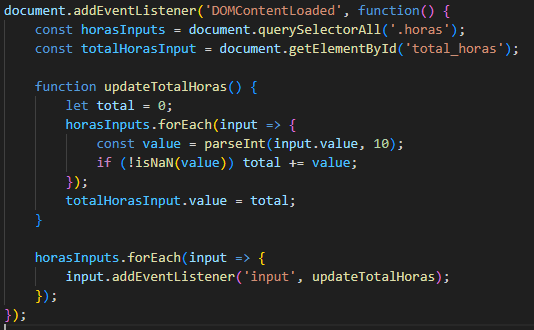
## 5.3. Validaciones Implementadas en la Aplicación

Para asegurar la integridad y coherencia de los datos, AcademyReport implementa diversas validaciones a nivel de frontend y backend.

Validaciones en Frontend (JavaScript):

* Se verifica que los campos obligatorios no estén vacíos.
* Se evita la superposición de horarios mediante alertas al usuario.
* Se valida que las horas totales asignadas sean correctas.

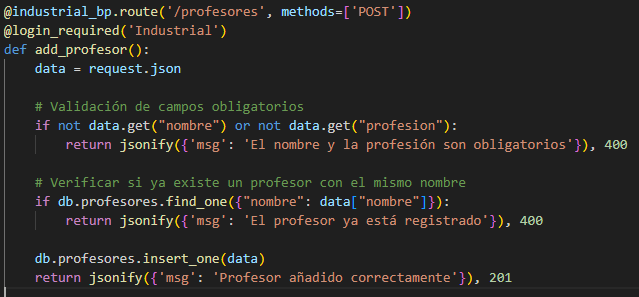
Ejemplo de Código (Validación de Horarios en JavaScript)



Validaciones en Backend (Flask - Python):

* Se verifica que los datos enviados por el usuario sean correctos.
* Se impide el registro de información duplicada.
* Se aseguran restricciones de acceso mediante autenticación y permisos.

Ejemplo de Código (Validación de Datos en Flask)



Tipos de Validaciones Implementadas

* Evitar datos vacíos: Se muestran mensajes de error si los campos obligatorios no se completan.
* Restricción de horarios: Se evita la duplicación de horarios para un mismo profesor.
* Seguridad de accesos: Solo los usuarios autenticados pueden modificar la base de datos.

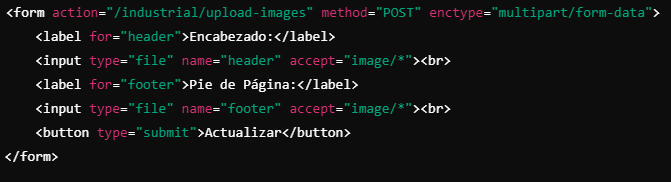
## 5.4. Administración de Imágenes y Documentos

El sistema permite la personalización de reportes mediante la carga de imágenes y documentos como:

* Encabezado y pie de página para los reportes en Excel.
* Actualización de textos personalizados en reportes institucionales.

Carga de Imágenes en el Sistema:

El usuario puede actualizar imágenes directamente desde la interfaz



Ejemplo de Código (Backend - Flask para Subida de Imágenes)



Beneficios de la Administración de Imágenes:

* Personalización de reportes con logotipos institucionales.
* Actualización rápida sin necesidad de modificar el código.
* Flexibilidad en el diseño de documentos exportados.

# 6. Manejo y Seguridad de la Aplicación

El manejo de seguridad en AcademyReport es un aspecto fundamental para garantizar la protección de los datos académicos y la integridad del sistema. Se han implementado múltiples capas de seguridad que incluyen autenticación de usuarios, restricción de accesos según roles, protección de rutas y manejo de errores.

## 6.1. Sistema de Autenticación y Roles de Usuario

El sistema de autenticación de AcademyReport se basa en Flask-Login, un módulo de Flask que permite gestionar sesiones de usuarios de manera segura.

### 6.1.1. Registro y Validación de Usuarios

Los usuarios deben autenticarse con un nombre de usuario y contraseña.

* Las credenciales se almacenan en MongoDB en formato encriptado usando bcrypt.
* Se permite el acceso solo a usuarios registrados.

Ejemplo de Código (Autenticación de Usuario - Flask-Login)



Mecanismos de Seguridad en la Autenticación:

* Contraseñas encriptadas con bcrypt: Evita que sean almacenadas en texto plano.
* Manejo de sesiones seguras con Flask-Login: Protege las credenciales del usuario.
* Redirección automática en caso de intento de acceso no autorizado.

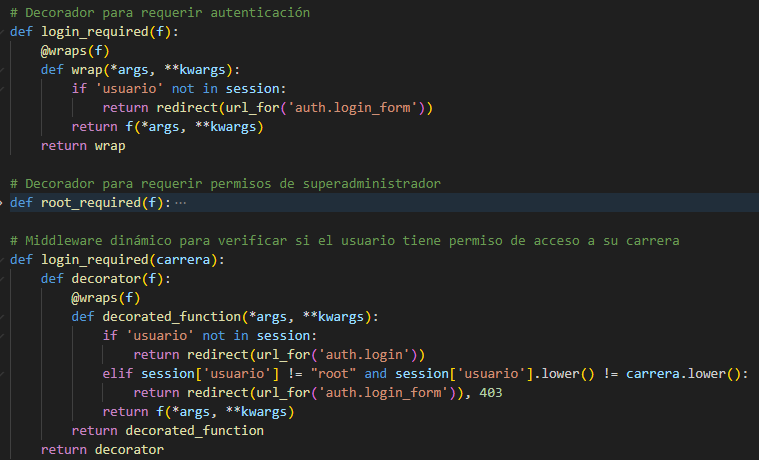
## 6.2. Protección de Rutas y Accesos

Una vez autenticado, cada usuario tiene permisos limitados según su rol. AcademyReport maneja diferentes tipos de usuarios:

* Administrador: Puede modificar cualquier dato.
* Jefe de Carrera: Solo tiene acceso a la información de sus docentes propios y compartidos.

Para restringir accesos, se usa un decorador personalizado que verifica los permisos de usuario antes de ejecutar una ruta.

Ejemplo de Código (Protección de Rutas según Rol de Usuario)



Beneficios de la Protección de Rutas:

* Evita accesos no autorizados a información sensible.
* Restringe funcionalidades según el rol del usuario.
* Asegura la integridad de los datos al evitar modificaciones indebidas.

## 6.3. Control de Errores y Manejo de Excepciones

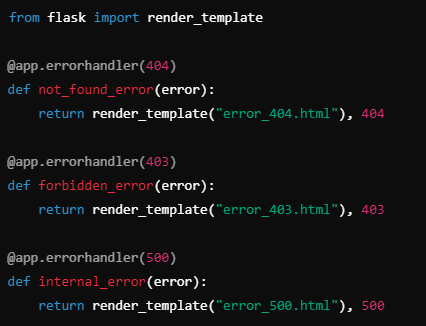
El sistema incorpora un robusto manejo de errores y excepciones para mejorar la experiencia del usuario y evitar fallos inesperados.

### 6.3.1. Manejo de Errores en Flask

Se han definido páginas personalizadas para errores como:

* Error 404 (Página no encontrada)
* Error 403 (Acceso denegado)
* Error 500 (Error interno del servidor)

Ejemplo de Código (Páginas de Errores en Flask)



Ventajas del Manejo de Errores Personalizado:

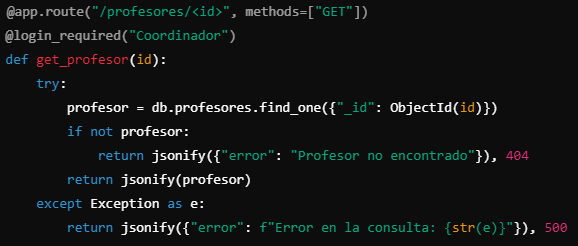
* Evita que los usuarios vean errores del servidor en crudo.
* Mejora la experiencia del usuario con mensajes explicativos.
* Permite registrar y depurar errores fácilmente en el sistema.

### 6.3.2. Manejo de Excepciones en la Base de Datos

Los errores en la base de datos pueden ocurrir si:

* Se intenta insertar datos duplicados.
* Se accede a una colección inexistente.
* Hay una falla de conexión con MongoDB.

Ejemplo de Código (Captura de Errores en Consultas a MongoDB)



Estrategias para el Manejo de Excepciones:

* Uso de try-except: Captura errores y evita que el sistema se caiga.
* Mensajes de error claros: Se informa al usuario si un registro no existe o hay un problema en la consulta.
* Logs de errores: Se pueden almacenar registros de errores en un archivo para depuración.

# 7. Instalación y Configuración del Proyecto

Esta sección detalla cómo instalar y configurar AcademyReport en un entorno local o servidor, asegurando un despliegue exitoso.

## 7.1. Requisitos Previos

Antes de instalar AcademyReport, asegúrese de cumplir con los siguientes requisitos:

* Software Necesario
* Python 3.8 o superior
* MongoDB (versión 5.x o superior)
* Flask y dependencias necesarias
* Git (opcional, para clonar el repositorio)

Dependencias del Proyecto:

AcademyReport requiere varios paquetes de Python, instalables con pip.

Ejecute el siguiente comando para instalar todas las dependencias:

pip install -r requirements.txt

Ventajas de usar un archivo de requerimientos:

* Facilita la instalación rápida del entorno.
* Asegura la compatibilidad con las versiones correctas de los paquetes.

### 7.1.1 Instalar LibreOffice en el servidor/local

En Windows

* Descarga e instala LibreOffice desde:

<https://www.libreoffice.org/download/download/>

* Luego, añade el directorio soffice a las variables de entorno (respetar la instalación en el disco C):

C:\Program Files\LibreOffice\program

* Abrir la configuración de variables de entorno

- Presiona Win + R, escribe sysdm.cpl y presiona Enter.

- En la ventana Propiedades del sistema, ve a la pestaña Opciones avanzadas.

- Haz clic en el botón "Variables de entorno...".

* Editar la variable Path

- En la sección "Variables del sistema", busca la variable llamada Path y selecciónala.

- Haz clic en el botón "Editar...".

- Luego, haz clic en "Nuevo" y agrega la ruta donde está soffice.exe:  
 C:\Program Files\LibreOffice\program

* Para probar que funciona, abre una terminal y ejecuta:

soffice –version

En Linux (Ubuntu/Debian)

* sudo apt update
* sudo apt install libreoffice

En Mac

* brew install libreoffice

## 7.2. Instalación en un Entorno Local

Para ejecutar AcademyReport en un entorno local, siga estos pasos:

1. Clonar el repositorio

Si usa Git, puede clonar el código fuente:

* git clone https://github.com/TulioEvoled/proyecto-reporteador.git
* cd proyecto-reporteador

2. Crear un Entorno Virtual (Opcional pero Recomendado)

* python -m venv venv
* source venv/bin/activate # Linux / Mac
* venv\Scripts\activate # Windows

3. Ejecutar la Aplicación

* flask run

Si la instalación fue exitosa, verá un mensaje como este:

* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

Abra un navegador y acceda a http://127.0.0.1:5000/.

## 7.3. Despliegue en un Servidor

Para hacer que AcademyReport esté disponible en línea, siga estos pasos:

1. Desplegar en un Servidor con Gunicorn

Ejecute el siguiente comando en el servidor:

* gunicorn --bind 0.0.0.0:8000 wsgi:app

Gunicorn es un servidor WSGI que permite ejecutar Flask en producción.

2. Configurar Nginx como Proxy Reverso

Edite /etc/nginx/sites-available/academyreport y agregue:

server {

listen 80;

server\_name academyreport.com;

location / {

proxy\_pass http://127.0.0.1:8000;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

}

}

* Luego, habilite la configuración con:

sudo ln -s /etc/nginx/sites-available/academyreport /etc/nginx/sites-enabled/

sudo systemctl restart nginx

3. Configuración en la Nube (Opcional)

Si desea desplegar en AWS, Digital Ocean o Heroku, asegúrese de:

* Configurar la Base de Datos Remota en MongoDB Atlas.
* Usar entornos virtuales o contenedores Docker.

4. Instalación de LibreOffice en el Servidor

Antes de usar LibreOffice para la conversión de Excel a PDF en un servidor, es necesario instalarlo.

Linux (Ubuntu/Debian)

* sudo apt update
* sudo apt install libreoffice

CentOS/RHEL

* sudo yum install libreoffice

Windows (Si el servidor es Windows)

* Descarga LibreOffice desde su sitio web: <https://www.libreoffice.org/download/download-libreoffice/>
* Instálalo y asegúrate de agregar la ruta a las variables de entorno (C:\Program Files\LibreOffice\program\).

5. Verificar la Instalación en el Servidor

* Para confirmar que LibreOffice está correctamente instalado y accesible, ejecuta:

libreoffice –version

* en algunos sistemas:

soffice –version

* Si ves la versión de LibreOffice, significa que está listo para usarse.

6. Configurar el Comando para Servidores

* En un servidor, es recomendable ejecutar LibreOffice en modo headless (sin interfaz gráfica), usando soffice:

soffice --headless --convert-to pdf temp\_reporte.xlsx --outdir .

* Si el servidor no reconoce soffice, prueba con la ruta completa:

/usr/bin/libreoffice --headless --convert-to pdf temp\_reporte.xlsx --outdir .

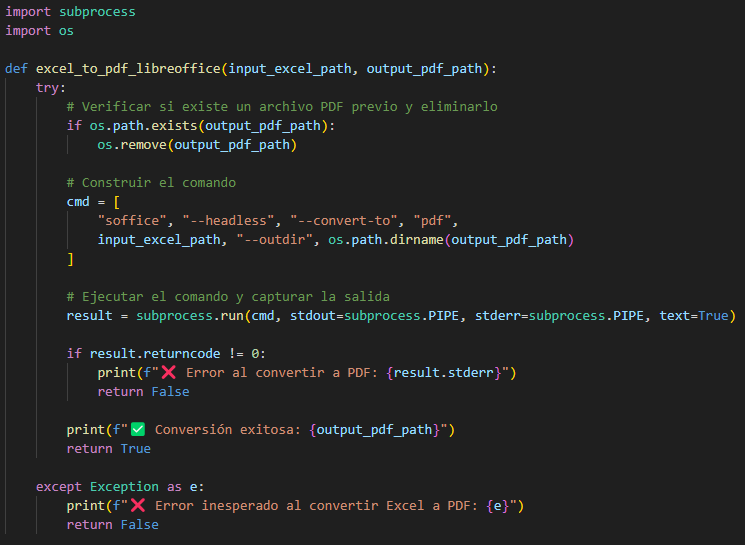
* en Windows:

"C:\Program Files\LibreOffice\program\soffice.exe" --headless --convert-to pdf temp\_reporte.xlsx --outdir .

7. Ajustar la Función de Conversión en el Código

Ahora que LibreOffice está en el servidor, la función excel\_to\_pdf\_libreoffice() debe asegurarse de:

* Ejecutar soffice con la ruta correcta.
* Manejar correctamente los errores y sobreescribir archivos si es necesario.



8. Prueba la Conversión en Producción

Después de subir y configurar en un servidor, realiza una solicitud para generar un PDF y revisa los logs:

* tail -f /var/log/nginx/error.log
* tail -f /var/log/syslog

Si todo está correcto, el servidor podrá convertir Excel a PDF sin problemas.

# 8. Conclusiones y Recomendaciones

AcademyReport ha sido diseñado para automatizar y optimizar la gestión académica, proporcionando una herramienta poderosa para la administración de profesores, asignaturas y reportes.

Beneficios Clave del Sistema:

* Modularidad - Se adapta a distintas carreras.
* Gestión de Reportes - Exportación en formatos Excel y PDF.
* Autocompletado y Validaciones - Evita errores al registrar datos.
* Seguridad Robusta - Protege la información con autenticación y control de accesos.
* Interfaz Amigable - Fácil uso para administradores y jefes de carrera.

Posibles Mejoras Futuras

* Integración con Moodle para sincronizar horarios automáticamente.
* Generación de reportes estadísticos sobre la carga de docentes.
* Soporte para múltiples idiomas en la interfaz.
* Despliegue con Kubernetes para alta disponibilidad.