Facultad

de

Ciencias

API REST y herramienta web para el cálculo del color en materiales multi-capa

(API REST and web tool to calculate the color of multi-layer materials)

Trabajo de Fin de Grado

para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Víctor Argüeso Cano

Director: Alfonso de la Vega Ruiz

Codirectora: Yael Gutiérrez Vela

Junio – 2024

Resumen

Este trabajo presenta el desarrollo de una API REST y una herramienta web para calcular el color resultante de la luz reflejada en materiales multicapa, utilizando el método de matriz de transferencia (Transfer Matrix Method, TMM). El TMM es un método empleado para analizar la propagación de ondas electromagnéticas a través de materiales de diferentes espesores, cada una con propiedades ópticas distintas (índice de refracción).

En el ámbito de la óptica, el TMM permite estudiar cómo la luz interactúa con un material multicapa, determinando qué porciones de la luz son reflejadas, transmitidas y cuáles son absorbidas en cada capa. Esto es crucial para identificar el color resultante del material, el cual se obtiene combinando el espectro de reflectancia del material con el espectro del iluminante y la función de respuesta del observador.

Aunque existen librerías que permiten el cálculo del color en materiales multicapa utilizando TMM, no hay aplicaciones que faciliten este proceso a usuarios sin experiencia en programación. Este proyecto aborda esta limitación mediante dos objetivos principales:

1. Desarrollar una API REST que, dado un sistema de materiales multicapa y una luz incidente, calcule y retorne el color resultante.
2. Crear una interfaz web amigable que permita a los usuarios definir las características del sistema multicapa y utilice la API para mostrar el color resultante.

La interfaz web permite a los usuarios interactuar fácilmente con la aplicación, definiendo y modificando el material de cada capa, así como su espesor e índice de refracción, como el espesor y el índice de refracción. Este enfoque hace que la herramienta sea accesible para usuarios no expertos, brindándoles la capacidad de realizar cálculos complejos de manera sencilla y eficiente.

Palabras clave: API REST, herramienta web, Transfer Matrix Method, materiales multicapa, cálculo de color, interfaz amigable.

Abstract

This project presents the development of a REST API and a web tool for calculating the resulting color of light reflected in multilayer materials using the Transfer Matrix Method (TMM). The TMM is a method used to analyze the propagation of electromagnetic waves through materials of different thicknesses, each with distinct optical properties (refractive index).

In the field of optics, the TMM allows studying how light interacts with a multilayer material, determining which portions of light are reflected, transmitted and which are absorbed in each layer. This is crucial for identifying the resulting color of the material, which is obtained by combining the material's reflectance spectrum with the spectrum of the illuminant and the observer's response function.

Although there are libraries that allow calculating the color in multilayer materials using TMM, there are no applications that facilitate this process for users without programming experience. This project addresses this limitation through two main objectives:

1. Developing a REST API that, given a multilayer material system and an incident light, calculates and returns the resulting color.

2. Creating a user-friendly web interface that allows users to define the characteristics of the multilayer system and uses the API to display the resulting color.

The web interface allows users to interact easily with the application, defining and modifying the material of each layer as well as its thickness and refractive index, such as thickness and refractive index. This approach makes the tool accessible to non-expert users, providing them with the ability to perform complex calculations in a simple and efficient manner.

Keywords: REST API, web tool, Transfer Matrix Method, multilayer materials, color calculation, user-friendly interface.

Índice

Índice de Figuras

Índice de tablas

1 Introducción

La percepción del color de los objetos que nos rodean está directamente relacionada con la interacción de la luz con dichos objetos. La luz, al incidir sobre un objeto, sufre un proceso de absorción, reflexión y transmisión. Parte de las longitudes de onda de la luz se absorben por el material del objeto, mientras que otras se reflejan y otra se transmiten. La fracción de la energía incidente que es reflejada para cada longitud de onda de la luz incidente determina el color que percibimos. Por ejemplo, un objeto se verá blanco si refleja todas las longitudes de onda, y negro si absorbe todas o casi todas las longitudes de onda.

Sin embargo, en materiales compuestos de varias capas con grosores en el rango micro o nanométrico, ocurre un fenómeno interesante: no solo la capa superficial del material influye en el color resultante, sino que las capas internas también juegan un papel significativo. Las longitudes de onda de la luz que son transmitidas por la capa superficial pueden ser reflejadas por la siguiente capa del material. De modo que, en cada interfaz entre capas, una parte de la luz se refleja y otra parte se transmite, contribuyendo a procesos de reflexión múltiple e interferencia. Este fenómeno se puede analizar mediante el método de la matriz de transferencia (Transfer Matrix Method).

El Transfer Matrix Method, mostrado gráficamente en la figura 1, permite describir y calcular cómo la luz interactúa con cada una de estas capas, considerando las ondas reflejadas, transmitidas y absorbidas en cada capa, así como su interacción. Esto proporciona una comprensión detallada de cómo las propiedades ópticas de un material multicapa determinan el color final que percibimos. Este fenómeno es especialmente relevante en el diseño y análisis de materiales avanzados, como aquellos utilizados en revestimientos ópticos, pantallas y sensores.

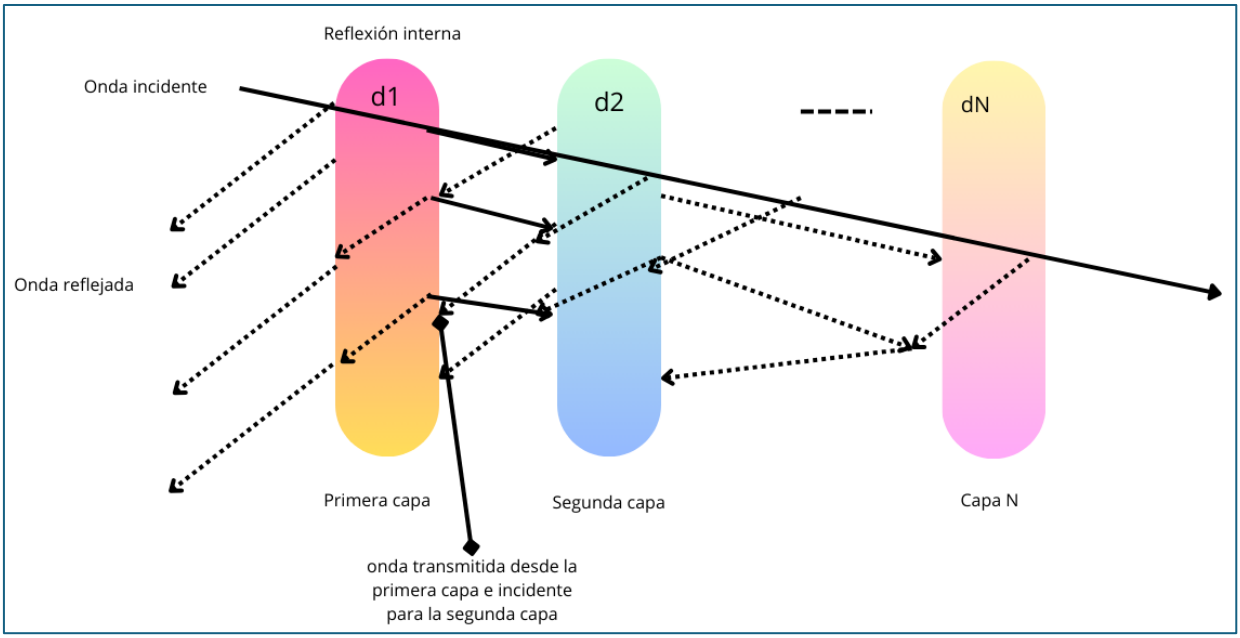


Figura 1 Proceso de reflexión múltiple en un sistema de multicapas

1.1 Motivación

En el mundo de la óptica, la precisión en la determinación del color resultante de la reflexión a través de múltiples capas de materiales es fundamental, al igual que comprender el origen de los colores que percibimos.

La falta de herramientas adecuadas para calcular estos colores y los inconvenientes presentados por las pocas aplicaciones disponibles motivaron este trabajo.

Decidimos trabajar en este proyecto tras identificar las limitaciones de las aplicaciones existentes que estábamos analizando. Estas limitaciones nos proporcionaron valiosas ideas para mejorar y superar las deficiencias observadas. Encontramos algunas aplicaciones que no permitían modificar las capas creadas por defecto y por lo tanto ya te imponían alguna capa a tu diseño. Otras aplicaciones mostraban de manera muy poco clara como iba quedando el diseño de capas que se iba haciendo, Por último, también detectamos falta de usabilidad, alguna aplicación no dejaba claro donde añadía la nueva capa cuando pulsabas el botón.

Con este TFG, no solo buscamos resolver problemas existentes, sino también brindar una herramienta versátil y eficaz que satisfaga las necesidades de los usuarios en el campo de la óptica y el análisis de colores resultantes de la reflexión.

1.2 Objetivo

El objetivo fundamental de este trabajo es desarrollar una aplicación que permita a los usuarios calcular colores en materiales multicapa mediante la aplicación de técnicas de TMM.

Esta aplicación se enfocará en la creación una interfaz intuitiva para que los usuarios ingresen parámetros específicos de los materiales y la iluminación incidente, y así calcular con precisión el color resultante. El objetivo incluye la utilización y adaptación de algoritmos de TMM y la creación de una interfaz de usuario amigable que permita una interacción eficiente.

El objetivo final es ofrecer una herramienta efectiva y versátil que satisfaga las necesidades de los usuarios en el campo del cálculo de colores en materiales multicapa, contribuyendo así al desarrollo y la innovación en el campo de la óptica aplicada.

2 Tecnologías y herramientas utilizadas

En esta sección se va a hablar brevemente de las herramientas y las tecnologías utilizadas para la realización del trabajo, describiendo brevemente su funcionamiento y explicando su utilidad en este trabajo.

2.1 Python

Python [3] es un lenguaje de programación de alto nivel, interpretado y de propósito general. Es conocido por su sintaxis clara y legible, lo que lo hace ideal tanto para expertos como para jóvenes programadores. Python es muy utilizado en unos diferentes campos, como desarrollo web, ciencia de datos, inteligencia artificial, automatización, entre otros. Su extensa biblioteca estándar y la gran cantidad de paquetes disponibles para descargar de manera sencilla hacen que sea fácil y rápido desarrollar una amplia gama de aplicaciones.

Las razones para seleccionar Python para el desarrollo de este trabajo fueron varias:

- La principal fue la existencia de un par de librerías que realizaban el cálculo de colores en un sistema multicapa utilizando el método de la matriz transformada: tmm [4] y colorpy [5]. (enlaces a las librerías).

- Una razón secundaria fue la continuidad de este trabajo: al tratarse de un tema de óptica, las personas que trabajarán y puede que tengan que mantener esta herramienta son físicos, perfil en el cual es más probable haber utilizado Python en el pasado que otras opciones como Java o C#.

2.2 PyCharm

PyCharm [7] es un potente entorno de desarrollo integrado (IDE) específicamente diseñado para programadores de Python. Su interfaz intuitiva y personalizable facilita el desarrollo de proyectos Python de cualquier tamaño. PyCharm es compatible con una variedad de frameworks y herramientas populares en el ecosistema de Python.

En este trabajo ha sido utilizado para realizar toda la parte de la API Rest, utilizándolo la mayoría de las veces como entorno de desarrollo de Python, a la vez que ha hecho funciones de editor de texto en otras.

2.3 JavaScript, HTML y CSS

HTML [8], JavaScript [9] y CSS [10] son lenguajes fundamentales para el desarrollo web.

HTML se encarga de la estructura y el contenido de una página web. JavaScript es un lenguaje de programación que permite añadir interactividad y diferentes funcionalidades a las a las páginas web. CSS se utiliza para definir el diseño y la apariencia visual de una página web, incluyendo aspectos como colores, tipografías y distribución de elementos.

Estos tres lenguajes suelen trabajar en conjunto para crear experiencias web completas y atractivas. A través de HTML, se define la estructura del contenido; con CSS, se aplica el estilo visual; y mediante JavaScript, se añade funcionalidad interactiva para mejorar la experiencia del usuario.

En este trabajo se ha trabajado simultáneamente con los 3 lenguajes para dar forma al cliente de nuestra aplicación.

2.4 Visual studio code

Visual Studio Code [11] es un editor de código ligero y altamente personalizable desarrollado por Microsoft. Ofrece una amplia gama de características, como resaltado de sintaxis, autocompletado inteligente y soporte para una variedad de lenguajes de programación, lo que lo convierte en una opción popular entre los desarrolladores. En este trabajo ha sido utilizado para la parte del cliente, soportando tanto HTML como JavaScript como CSS.

2.5 SourceTree / Git

SourceTree [12] es una interfaz gráfica de usuario (GUI) para el sistema de control de versiones Git, que simplifica su uso y gestión.

Git [13] es un sistema de control de versiones. Permite crear un repositorio en el que ir almacenado las distintas versiones del código, crear distintas ramas para trabajar, además de otras ventajas para desarrollos multiusuario que no se han utilizado al tratarse de un proyecto individual. Un repositorio git almacenado en una plataforma como GitHub o GitLab permite compartir casi de manera instantánea el desarrollo de una persona con sus compañeros sin necesidad de enviarse código ni tener que fusionar códigos manualmente.

2.6 Postman

Postman [14] es una plataforma colaborativa que simplifica el desarrollo de APIs basadas en HTTP permitiendo a los desarrolladores diseñar, probar y depurar sus servicios web de manera eficiente.

En este trabajo ha sido utilizada para la realización de las pruebas de integración.

2.7 Selenium

Selenium [14] es una herramienta de automatización de navegadores web, ampliamente utilizada para pruebas automatizadas de aplicaciones web. Permite interactuar con páginas web de manera programática, simulando acciones de usuarios como clics, relleno de formularios y navegación entre páginas. Selenium soporta múltiples lenguajes de programación, entre ellos Python.

3 Metodología y planificación

En este apartado describimos de la metodología de trabajo seguida para la elaboración del trabajo, y explicamos el procedimiento que llevamos desde el principio y la evolución semanal de las ideas iniciales.

3.1 Idea inicial

La idea inicial de este proyecto era crear una aplicación que permitiera a los usuarios poder calcular el color resultante de la reflexión de la luz, con un ángulo determinado en un sistema compuesto por una serie de capas de un material y un grosor determinados.

A continuación, vamos a explicar las funciones de cada miembro en este proyecto, ya que más adelante se mencionarán en base a su rol.

- Yael Gutiérrez tomó el papel de Product Owner, como experta en óptica y originadora de la idea del proyecto.

  Un Product Owner es la persona que se encarga de decidir qué características y mejoras se deben incluir en un producto. Es el puente entre los clientes y el equipo de desarrollo, asegurándose de que el producto final cumpla con lo que los usuarios necesitan y lo que el negocio quiere lograr.

- Alfonso de la Vega supervisó y asistió al alumno durante el proceso de desarrollo.

La idea extraída de las primeras reuniones con la Product Owner nos llevaron a crear un mockup de partida, que evolucionó durante el transcurso del proyecto. Este mockup puede encontrarse en la Figura 2.

La idea inicial constaba de un cuadro de texto donde se escribía el número de capas que se querían crear, y el programa te generaba ese número de capas. Esta idea fue modificada con el paso de las semanas debido a que hacía algo compleja de usar la web ya que no permitía añadir fácilmente una capa en el lugar deseado como sí se puede en la aplicación final.

También en la idea inicial creímos conveniente mostrar la representación gráfica de la distribución de capas para que el usuario viera dibujado lo que estaba tratando de simular. En reuniones posteriores decidimos que iba a reducir la claridad de la aplicación a la hora de identificar los elementos, ya que en caso de tener muchas capas iba a hacer un dibujo de gran tamaño e iba a ser imposible representar las capas en una escala de tamaño adecuado para todos, ya que una capa puede ser de una unidad mayor como metros y otra de una unidad menor como milímetros, e iba a ser imposible representarlo de una manera realista.

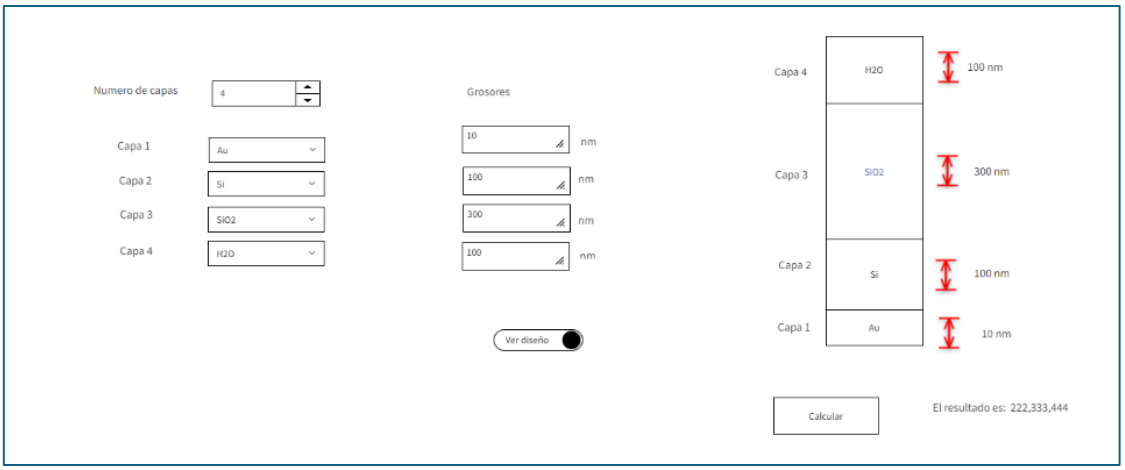


Figura 2 Primeros mockups de la aplicación resultado de las reuniones iniciales

Además de las ideas de diseño de aplicación, en la idea inicial también decidimos utilizar Python como lenguaje de desarrollo por lo mencionado con anterioridad al respecto de las librerías existentes (tmm y colorpy) y el futuro mantenimiento de la aplicación.

Por último, acordamos una metodología de trabajo iterativa que se describe en la siguiente sección, con periodos de trabajo semanales con constante comunicación de los avances entre todas las partes.

3.2 Metodología y desarrollo

Para llevar a cabo el desarrollo del proyecto se ha seguido una metodología iterativa, con la ayuda de la herramienta GitHub, donde se creó un repositorio publico [1] donde se han ido incluyendo los avances durante el proceso de desarrollo.

Tras la reunión inicial descrita en el anterior apartado, se siguió un método de trabajo. Se llevo a cabo un método de trabajo donde nos reuníamos el alumno y el profesor encargado del trabajo cada dos semanas, revisábamos el trabajo hecho durante este tiempo, hacíamos las correcciones y comentarios necesarios, resolvíamos las dudas que habían podido surgir en ese tiempo y asignábamos nuevas tareas para el siguiente sprint.

Después de esta reunión, para la parte de backend no hicimos más reuniones con la Product Owner, nos encargamos del desarrollo del servidor el alumno y el profesor encargado, hasta obtener un desarrollo correcto del sistema. Esto fue así ya que el objetivo era tener un componente que calculara colores de forma correcta a partir de unos sistemas conocidos (se sabía el resultado de color que debíamos obtener), por lo que la colaboración de la Product Owner no fue necesaria.

Cuando consideramos que la parte de backend estaba funcionando correctamente, nos pusimos en contacto con la Product Owner para que nos

suministrara datos de pruebas y así poder realizar las pruebas necesarias y comprobar que habíamos hecho un desarrollo correcto. Para esto no fue necesario realizar reuniones presenciales, nos fue suficiente con comunicarnos por correo electrónico.

Cuando empezamos a realizar el diseño de la parte de frontend mantuvimos la segunda reunión con la Product Owner para hablar nuevamente. En este momento ya con más conocimiento acerca de la aplicación y nuevas ideas adquiridas durante el proceso de desarrollo del servidor, necesitábamos su ayuda para perfeccionar las ideas iniciales del diseño y crear un boceto más definitivo de la aplicación.

A partir de este punto seguimos manteniendo reuniones semanales sin la presencia de la Product Owner hasta que dimos por finalizado todo el diseño, teniendo este un funcionamiento correcto. En este punto mantuvimos la última reunión los tres miembros juntos, haciendo la Product Owner los últimos comentarios, mejoras y correcciones para dejar la interfaz de la aplicación totalmente a su gusto.

Después de esta reunión volvimos a tener reuniones sin la Product Owner para continuar con todo el trabajo final del TFG, siendo este consciente de los cambios realizados y dándonos su aprobación al trabajo realizado sin la necesidad de mantener más reuniones.

A continuación, vamos a explicar las tareas desarrolladas en cada sprint de manera que quede un poco más claro el reparto de trabajo y el desarrollo durante todo el proceso de realización del trabajo.

- Sprint1: Este tramo de tiempo fue dedicado a realizar el análisis de requisitos extraídos de la primera reunión con la Product Owner y para empezar a familiarizarnos con la librería TMM y a comprender el funcionamiento de la función dada para calcular el color y decidir la manera de utilizarla en el siguiente sprint.

- Sprint 2: En este periodo de tiempo se creó una función para calcular el color extrayéndola del código dado por la librería TMM, suministrado por el director del proyecto, y se desarrolló una función principal sencilla para probar el código extraído junto con las funciones creadas para leer ficheros que contenían datasets en formato tabla y calcular la ruta para encontrar los ficheros en las carpetas del proyecto.

- Sprint 3: En este periodo se comenzó a diseñar una api para dar cobertura a las dos funciones necesarias para el funcionamiento de la aplicación, la función para obtener materiales de los ficheros y la función para calcular el color en base a los métodos de la librería TMM. Por último, se realizaron las pruebas unitarias para las funciones creadas durante los sprints 2 y 3.

- Sprint 4: Se empezó con la creación de una interfaz web básica, con un número de filas estático, desarrollando una tabla inicial con tres filas predeterminadas, sin ninguna funcionalidad y con la creación de algún botón sin ninguna funcionalidad tampoco.

- Sprint 5: En este sprint trabajamos en crear la funcionalidad de añadir una nueva fila debajo de una existente. También implementamos la función que elimina una fila.

- Sprint 6: En este tramo de tiempo se realizó la conexión cliente servidor dando funcionalidad a obtener la lista de materiales para poder acceder a la lista de materiales y añadirlos a las capas y se dio funcionalidad a calcular colores.

- Sprint 7: En este sprint de integración además de realizar un refinamiento en la interfaz para pulir los detalles requeridos por la Product Owner y añadimos nuevos materiales para utilizarlos en la realización de pruebas.

- Sprint 8: Durante este sprint realizamos las últimas modificaciones en la interfaz y, por último, realizamos las pruebas de interfaz utilizando Selenium. Durante este sprint realizamos las últimas modificaciones en la interfaz

4 Análisis de requisitos

4.1 Descripción general de la aplicación

Esta aplicación solo cuenta con una vista de usuario así que vamos a explicar el funcionamiento y las funcionalidades de esta desde un único punto de vista genérico.

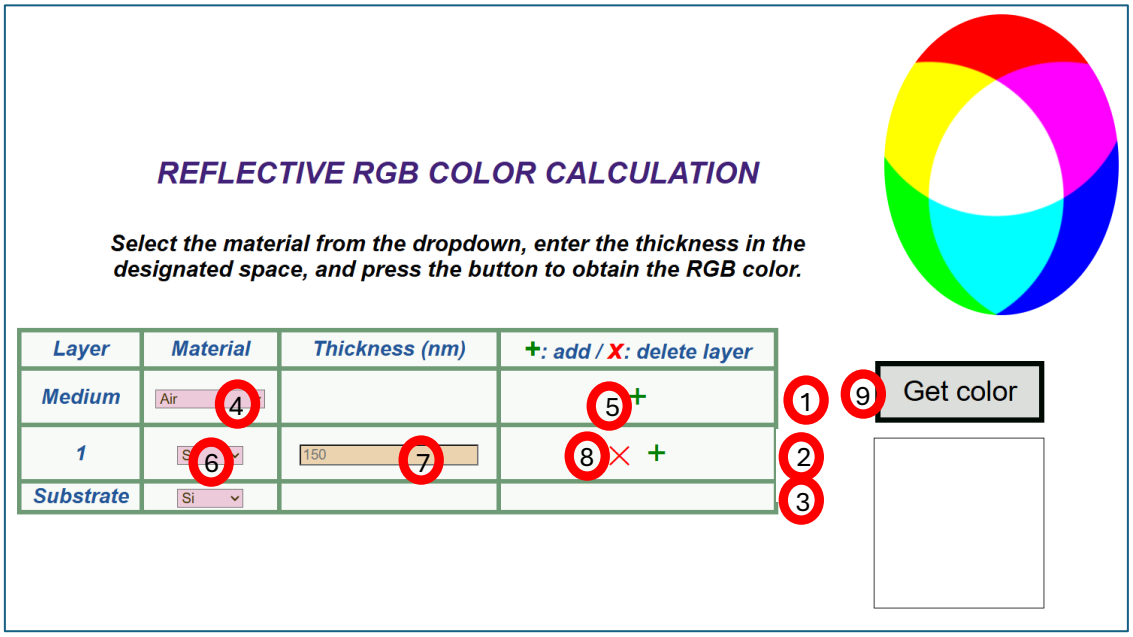


Figura 3 Imagen inicial de la interfaz al abrir la aplicación

En la figura 3 se podemos ver la interfaz inicial que ve el cliente al iniciar la aplicación. Cuando el usuario inicia la aplicación puede apreciar tres capas ya creadas, la primera es la capa que representa al medio (punto 1), que no permite la opción de eliminarla, por defecto es aire cuyo índice de refracción es 1 (punto 4), pero en caso de que el usuario desee utilizar este material, puede seleccionar la opción “other” en el desplegable y aparecerá un cuadro de texto donde se puede añadir el valor del indie de reflexión del material que desee.

Otra de las capas que aparece creada es la capa inferior denominada sustrato (punto 2). Por defecto es silicio, pero cuenta con un desplegable que muestra todas las opciones de materiales que disponemos en la base de datos.

Ambas capas no permiten modificar su grosor, ya que este es “infinito”, esto es debido a que están representando el medio y el sustrato.

Al iniciar, el usuario también se va a encontrar creada una capa intermedia (punto 3), que no es más que una capa creada por defecto para mejorar la impresión inicial pero que el usuario puede modificar a su antojo, pudiendo eliminarla, cambiar el material y cambiar el grosor.

A la derecha de cada una de las capas (excepto la del sustrato), existe un botón verde “+” que perite añadir nuevas capas (punto 5). Cada capa nueva se añade debajo de la capa sobre la que se ha seleccionado el botón añadir.

Al crearse la nueva capa, el cliente dispone de un desplegable en la columna “material” (punto 6), que le permite visualizar todos los materiales de los que disponemos en nuestra base de datos y poder elegir el que desee.

A continuación, en la columna “grosor” (punto 7) puede introducir el valor numérico que desee para este campo, medido en nanómetros.

El usuario en todo momento puede eliminar las capas que desee, menos las de medio y sustrato. Para eliminar una capa solo debe pulsar el botón eliminar “x” encima de la capa deseada (punto 8).

Cuando el usuario esté a gusto con su diseño, tiene que pulsar el botón “Get color” (punto 9) y la aplicación mostrará las coordenadas RGB del color resultante y junto a ellas en un recuadro mostrará visualmente dicho color.

4.2 Requisitos funcionales

Cuando se habla de los requisitos funcionales de una aplicación, se hace referencia a los servicios que ofrecerá y a la forma en que reaccionará en distintas situaciones. En la siguiente tabla se detallan los requisitos funcionales de la aplicación:

Identificador	Requisito	Descripción
RF001	Configurar capa de material	La aplicación dispone de un campo para introducir valores relativos al grosor la capa y dispone de un desplegable para elegir el material del que esa capa se compone.
RF002	Añadir nueva capa de material	El usuario puede añadir nuevas capas debajo de cada capa existente.
RF003	Eliminar capa de material	El usuario puede eliminar todas las capas que desee salvo la primera y la última, ya que estas son el medio y el sustrato y no pueden desaparecer.
RF004	Calcular coordenadas RGB del color	El usuario puede obtener el color resultante al incidir luz sobre el sistema de materiales multicapa que haya configurado.
RF005	Visualizar el color	La aplicación mostrará gráficamente el color real obtenido en el cálculo.
RF006	Establecer índice de refracción del medio	El material del medio por defecto es aire, pero también es posible introducir a mano un índice de refracción personalizado.

Tabla 1. Requisitos funcionales

A continuación, se incluyen las fichas de casos de uso de los requisitos funcionales descritos en la Tabla 1.

ID	***RF001***
Nombre	Configurar capa de material.
Descripción	La aplicación dispone de un campo para introducir valores relativos al grosor la capa, y de un desplegable para elegir el material del que esa capa se compone
Actores primarios	Usuario.
Flujo Principal	1. Hace click en el desplegable 2. El sistema muestra los materiales disponibles 3. El usuario escoge el material que desea 4. El Material escogido se queda fijado en el desplegable 5. El usuario se coloca sobre el cuadro de texto. 6.El usuario escribe el grosor que desea. 7.El grosor escogido queda escrito en el cuadro de texto.
Flujo alternativo	Alternativo1: Valor numérico menor o igual a 0. 2a. El usuario introduce un valor menor o igual que 0 en el campo de grosor. a.1.El sistema indica que el valor no es válido. a.2.El usuario vuelve a introducir el valor. a.3.Se retoma el flujo principal en el paso 2.
Postcondiciones	-El valor del material de la capa queda configurado. -El valor del grosor de la capa queda configurado.
Comentarios	La elección de material y la introducción de valor numérico podrían entenderse como dos casos de uso separados, pero dada su sencillez se han combinado en uno, ya que ambos componen la configuración de una capa.

Tabla 2. RF001 Configurar capa.

ID	***RF002***
Nombre	Añadir nueva capa de material
Descripción	El usuario añade una capa nueva a las capas existentes.
Actores primarios	Usuario.
Flujo Principal	1.El usuario selección el botón de añadir en alguna de las capas existentes. 2. El sistema añade la capa debajo de la capa seleccionada.
Flujo alternativo	
Postcondiciones	La nueva capa aparece en la tabla para ser configurada.

Tabla 3. RF002. Añadir una capa de un material.

ID	***RF003***
Nombre	*Eliminar capa de material.*
Descripción	*El usuario elimina una de las capas ya existentes.*
Actores primarios	*Usuario.*
Flujo Principal	*1.El usuario selecciona la opción de borrar capa sobre la capa que quiere eliminar.* *2.El sistema elimina la capa y reajusta la interfaz para "pegar" las capas inferior y superior a la capa eliminada.*
Flujo alternativo	
Postcondiciones	

Tabla 4. RF003.Eliminar capa de un material.

ID	***RF004***
Nombre	Calcular coordenadas RGB del color.
Descripción	El usuario puede obtener el color resultante al incidir luz sobre el sistema de materiales multicapa que haya configurado.
Actores primarios	*Usuario.*
Precondiciones	*El sistema debe estar operativo y mostrar la interfaz que permita configurar las capas.* *Las capas deben estar configuradas por los usuarios.*
Flujo Principal	*1.El usuario selecciona el botón calcular color.* *2. Se envía una petición de cálculo de color al servidor remoto.* *3.Se reciben de vuelta los resultados.* *2.Se muestran coordenadas en formato json al lado del botón seleccionado.*
Flujo alternativo	Alternativo A: Servidor no encontrado 2a. El sistema avisa de que no se puede encontrar el servidor mostrando el mensaje “server not found”. a.1 El usuario comprueba su conexión a internet. a.2 El usuario reintenta la conexión volviendo al punto 2 del flujo principal. Alternativo B: Servidor inactivo 2a. El sistema avisa de que el servidor no se encuentra activo mostrando el mensaje “Error: Unable to connect to the server”. a.2 El usuario reintenta la conexión volviendo al punto 2 del flujo principal. Alternativo C: Error en el cálculo del color 3c. El sistema no puede calcular los colores c.1.El sistema muestra un mensaje de error indicando "Error retrieving colors." c.2. El usuario reintenta la operación volviendo al punto 2 del flujo principal.
Postcondiciones	

Tabla 5. RF004. Calcular coordenadas RGB del color

ID	***RF005***
NOMBRE	Visualizar el color.
Descripción	Al calcular el color el usuario ve el color real obtenido.
Actores primarios	*Usuario.*
Flujo Principal	*1.El usuario selecciona el botón calcular color* *2.El color se podrá visualizar en un recuadro al lado del botón seleccionado.*
Postcondiciones	

Tabla 6. RF005. Visualizar el color

ID	***RF006***
Nombre	Establecer índice de refracción del medio.
Descripción	El material del medio por defecto es aire, pero si se selecciona la opción otro, aparece un cuadro de texto para introducir el índice de refracción deseado.
Actores primarios	*Usuarios.*
Precondiciones	El usuario puede obtener el color resultante a las capas introducidas en el formulario de la tabla.
Flujo Principal	*1.El usuario selecciona la opción "other material" en el campo material de la primera capa.* *2.Aparece un campo de texto donde el usuario introduce el valor numérico que desea para el índice de* refracción *del material.* *3.El índice de* refracción *introducido queda fijado en el cuadro de texto.*
Flujo alternativo	*Alternativo A. Valor erróneo* *2a. El usuario introduce un valor numérico negativo en el campo.* *a.1 El sistema muestra un mensaje diciendo que debe introducir un valor mayor que 0.* *a.2 El usuario introduce un nuevo valor mayor que 0.* *a.3 Se retoma el flujo principal en el paso 2.*
Postcondiciones	*El valor del índice de* refracción *del material de la capa queda configurado.*

Tabla 7. RF006.Establecer el índice de refracción del medio.

4.3 Requisitos no funcionales

Los requisitos no funcionales de una aplicación hacen referencia a aquellos que no tratan las funciones del sistema sino a sus propiedades. En la siguiente tabla se detallan los requisitos no funcionales de la aplicación:

Identificador	**Descripción**
RNF001	La interfaz de usuario del sistema debe ser fácil de utilizar.
RNF002	La aplicación debe desarrollarse en Ingles ya que se usará a nivel internacional.
RNF003	La aplicación debe asistir al usuario a la hora de detectar posibles errores.

Tabla 8. Requisitos no funcionales

En el siguiente párrafo vamos a explicar las medidas que hemos llevado a cabo para cumplir cada requisito no funcional.

RNF001. Existen descripciones textuales al lado de cada uno de los controles de la aplicación, que se han tratado de hacer lo más autoexplitativos posibles (un + verde para añadir capa, una x roja para borrarla, además del texto que las describe, etcétera).

RNF002. Realizado en cuanto a que la app está efectivamente en inglés.

RNF003. En aquellos campos libres donde el usuario debe introducir algún valor (básicamente valores numéricos de grosor o de índices de refracción), al usuario se le informa de la presencia de errores con un mensaje explicativo.

5 Arquitectura de la aplicación

Una vez se ha definido cómo debe funcionar la aplicación mediante la captura de requisitos, continuamos con el diseño del sistema.

La arquitectura de la aplicación es el diseño de más alto nivel de la estructura de un sistema. Hace referencia a los componentes que se requieren para construir el sistema, y las relaciones o comunicaciones que se producen entre ellos.

Esta aplicación se ha desarrollado siguiendo una estructura en capas, más concretamente una arquitectura en tres capas: capa de presentación, capa de negocio y capa de datos.

En la figura 4 se puede apreciar gráficamente el diseño arquitectónico de la aplicación, estando representados tanto las interacciones como los lenguajes y librerías utilizados para ello.

El objetivo principal de este tipo de arquitectura es la separación de cada una de las partes que componen el sistema software. Como indica el esquema, cada una de las capas solo podrá interactuar con capas vecinas. De esta forma, debería ser más sencillo poder realizar cualquier cambio en la aplicación, dado que este solo afectará a la capa involucrada y a lo sumo, a las vecinas.

A continuación, se explican más detalladamente cada una de las tres capas.

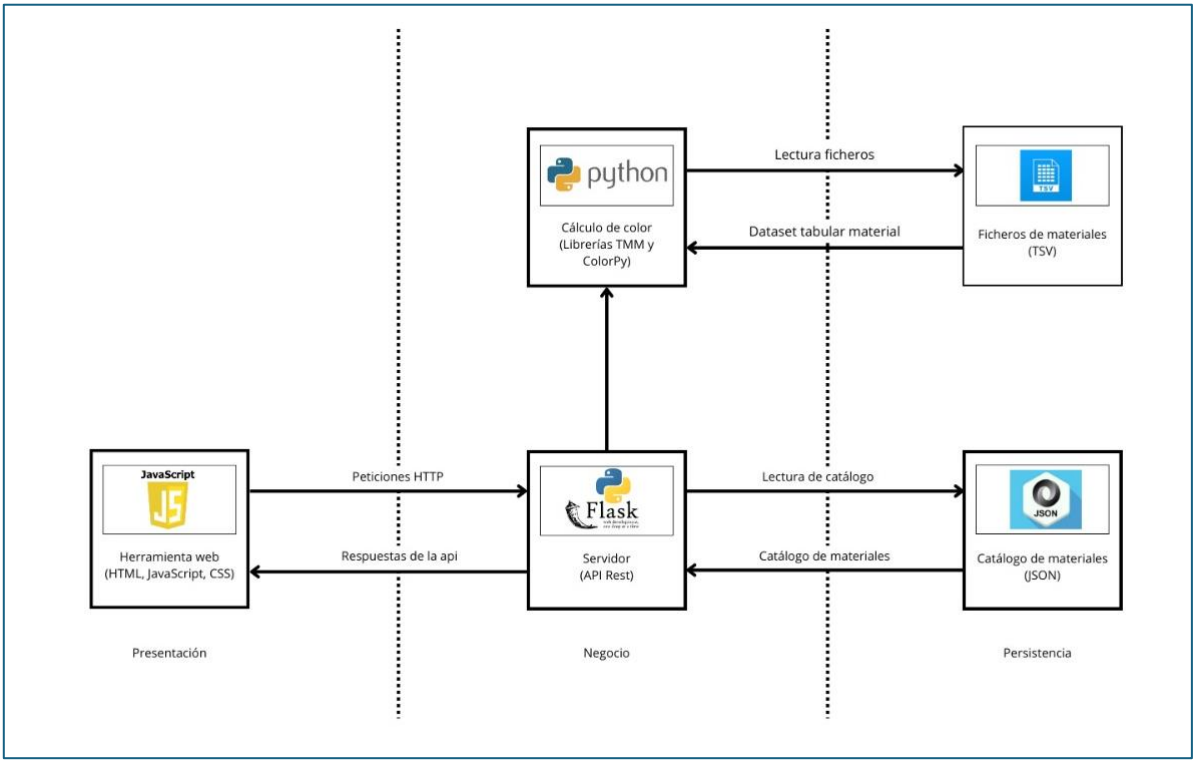


Figura 4 Diseño arquitectónico

5.1 Capa de presentación

La capa de presentación es la que ve el usuario y mediante la cual interactúa con el sistema. En nuestro caso hemos optado por crear una herramienta web, que es la encargada de comunicar la información al usuario y capturar su información, de forma amigable, entendible y procurando ser fácil de utilizar. Esta capa se comunicará únicamente con la inmediatamente inferior, es decir, con la capa de negocio.

Para diseñar el frontend web se ha utilizado HTML y JavaScript. Además de esto utilizamos CSS adaptar el estilo. Uno de los puntos más importantes en la creación de la interfaz de usuario es la usabilidad y la facilidad de aprendizaje para el usuario final. Esta aplicación solo cuenta de una vista y por lo tanto carece de interacciones entre vistas.

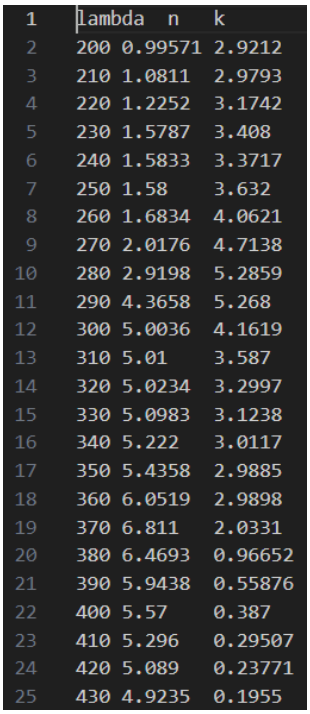
5.2 Capa de negocio

Esta es la capa que contiene la funcionalidad principal de la aplicación. Es la encargada de recibir y responder a las solicitudes recibidas de la capa de presentación, obteniendo los datos necesarios para procesar dichas solicitudes. De esto se encarga la API REST, que utilizando el framework Flask [2] y Flask - RESTX [16] crea un servidor que permite conectarse con el cliente de la capa de presentación.

Para ello, la capa de negocio se comunicará con la capa de acceso a datos y adquirirá información de la base de datos en función de la solicitud recibida. Esta capa está integrada en el backend de la aplicación, a través de las funciones de cálculo de color, construido utilizando Python utilizando las librerías TMM y ColorPy.

5.3 Capa de acceso a datos

Esta es la capa en donde residen los datos y la encargada de gestionar el acceso a los mismos. Esta capa en este proyecto consta de una lista de datasets tabulares almacenados en archivos de texto donde las columnas están separadas por tabuladores.



	lambda	n	k
2	200	0.99571	2.9212
3	210	1.0811	2.9793
4	220	1.2252	3.1742
5	230	1.5787	3.408
6	240	1.5833	3.3717
7	250	1.58	3.632
8	260	1.6834	4.0621
9	270	2.0176	4.7138
10	280	2.9198	5.2859
11	290	4.3658	5.268
12	300	5.0036	4.1619
13	310	5.01	3.587
14	320	5.0234	3.2997
15	330	5.0983	3.1238
16	340	5.222	3.0117
17	350	5.4358	2.9885
18	360	6.0519	2.9898
19	370	6.811	2.0331
20	380	6.4693	0.96652
21	390	5.9438	0.55876
22	400	5.57	0.387
23	410	5.296	0.29507
24	420	5.089	0.23771
25	430	4.9235	0.1955

Figura 5 Fragmento archivo con los datos que representan el índice de refracción de un material

En la Figura 5 se muestra un fragmento de uno de estos archivos que contienen los índices de refracción de los materiales, existiendo un fichero por cada uno de los materiales. Estos ficheros presentan tres columnas. La primera columna, lambda, indica una longitud de onda. las dos columnas siguientes, n y k, son el índice de refracción del material para esa longitud de onda concreta. Este valor ocupa dos columnas porque se trata de un numero complejo.

Además de esto, en la base de datos almacenamos un diccionario o catálogo con todos los materiales que están disponibles, incluyendo sus iniciales y su id, en formato json. Este diccionario nos permite tener listados todos los materiales de manera que sea fácil acceder a ellos para poder mostrar el listado en el método correspondiente a mostrar materiales en la interfaz.

Para este trabajo se decidió desde el principio no realizar una base de datos convencional debido a que esta aplicación solamente tuvo como objetivo el cálculo de colores de sistemas multicapa en un conjunto de materiales disponible y cerrado, por lo que solamente se realizarían lecturas de la información de estos materiales. Tenerlos almacenados en ficheros tabulares nos permite utilizarlos directamente en cálculos, por lo que no se vio ninguna ventaja a almacenarlos en una base de datos relacional frente a tenerlos simplemente en ficheros. Hay que tener en cuenta que estos ficheros han sido validados antes de su incorporación al catálogo de materiales, y que no son modificados en ningún momento desde la aplicación.

5.4 Esquema de posibles escenarios

En esta sección, presentamos ejemplos de las dos interacciones principales entre la interfaz de usuario y la API. Describimos las operaciones realizadas por el usuario en la interfaz, indicando con índices en el diagrama los puntos específicos en los que la API y la herramienta web interactúan.

1. El cliente abre la web. El frontend pide el catálogo de materiales a la capa de negocio (paso 1) a través de una petición HTTP GET, que a su vez lo obtiene de la capa de persistencia (paso 5) a través de la lectura de catálogo.
   La capa de persistencia proporciona un catálogo en formato JSON con todos los materiales (paso 6) y la capa de negocio a través de la respuesta de la API (paso 4) proporciona el catálogo que se usa para rellenar los desplegables de materiales.

2. Tras configurar las capas del material, el cliente hace click en el botón de calcular el color. Entonces, el sistema multicapa se transfiere a la API como una petición HTTP GET (paso 1) la API procesa este sistema para saber qué datos de materiales debe cargar de la capa de persistencia (paso 2). Una vez que obtiene los datos necesarios (paso 3), hace el cálculo, y devuelve el resultado (paso 4).

En la figura 4 se muestra el diagrama explicativo de la arquitectura indicando con números la acción que se realiza en cada paso.

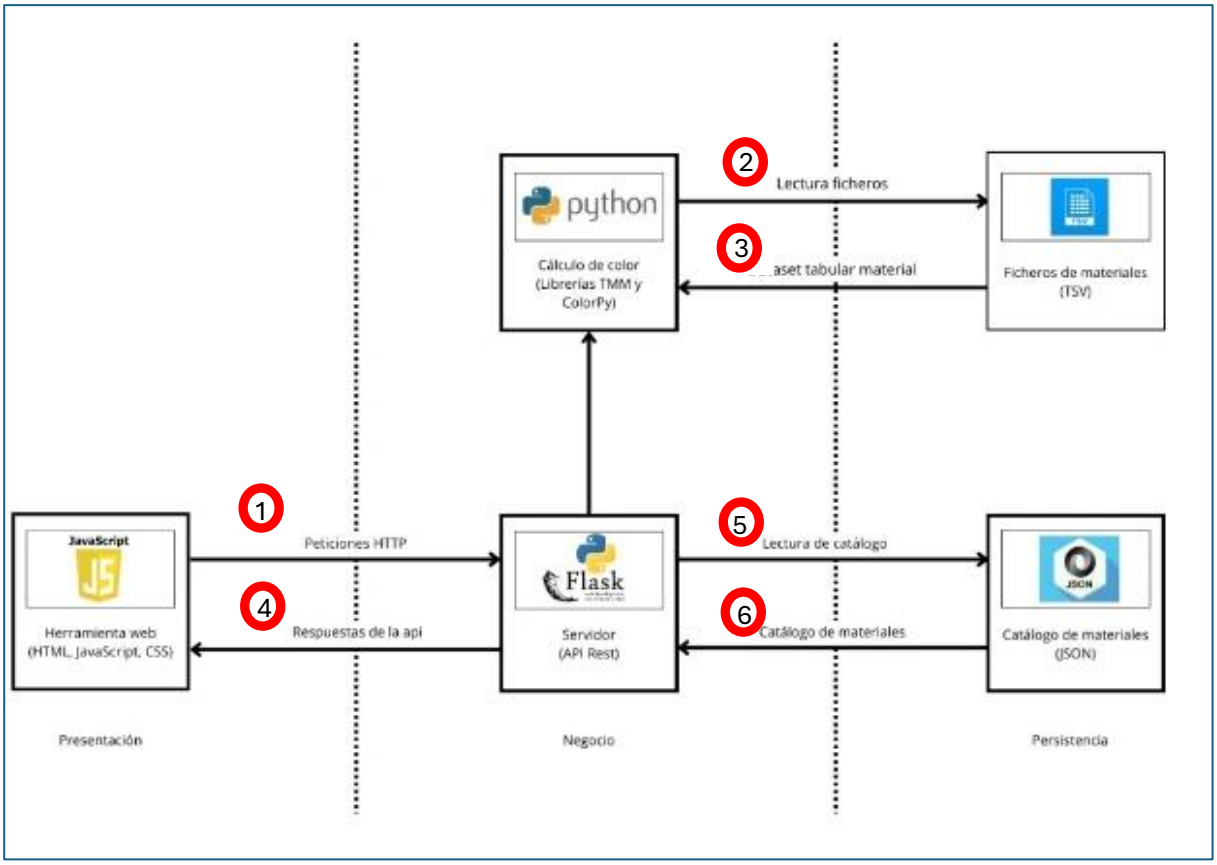


Figura 6 Diseño arquitectónico

6 Implementación

En este apartado a dar más detalles de bajo nivel acerca de cómo se ha implementado la aplicación descrita en las secciones anteriores.

6.1 Estructura del proyecto de desarrollo

En la figura 7 se puede observar la distribución global del proyecto.

La carpeta api_color contiene al servidor y la carpeta web_client contiene al cliente.

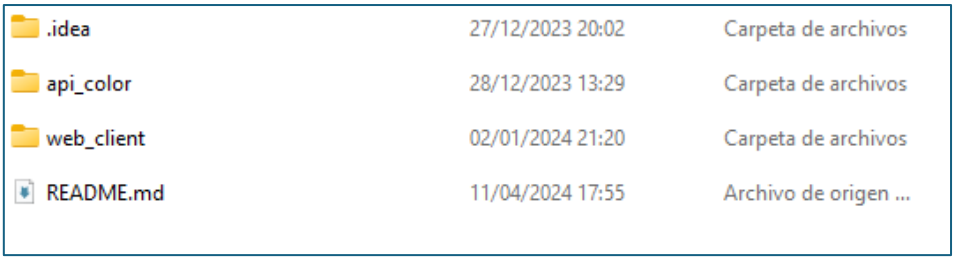


Figura 7 *Estructura en carpetas de la aplicación*

En la Figura 8 se puede observar la organización de las carpetas del servidor.

- La carpeta "*calculo_color"* contiene las clases de dominio donde se encuentran las funciones que vamos a utilizar en la api para dar funcionalidad a la aplicación.
- La carpeta "*rest*" contiene la api de la aplicación.
- La carpeta "*test*" contiene los casos de prueba que hemos tenido en cuenta para probar la aplicación.
- La carpeta "*others"* tiene otras clases auxiliares que hemos utilizado durante la implementación.

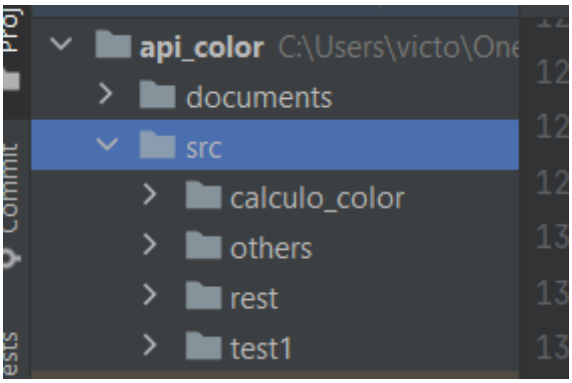


Figura 8 Estructura en carpetas del servidor

6.2 Frontend

En esta sección vamos a hablar de la parte del cliente web.

Por un lado, encontramos el archivo index.html. Este archivo está programado en lenguaje HTML. Este archivo es el encargado de crear una tabla dinámica inicialmente creada con tres filas y tres columnas, dispone de la estructura inicial de los botones de eliminar y crear filas y para calcula el color. Este fichero tiene relación con los archivos "*script.js* " y "*style.css".*

El archivo "*script.js*" contiene los métodos que dan funcionalidad a la interfaz, en él se encuentran los métodos que dan funcionalidad a los botones que tenemos en la interfaz de añadir y eliminar filas y el botón que calcula el color resultante. Este punto se puede entender mejor siguiendo el esquema del escenario 2 del apartado 5.4, ya que en este fichero lo que hacemos es dar funcionalidad a los pasos indicados en el esquema. También contiene las funciones que cargan los datos del backend. Esto se relaciona con el esquema de escenario número 1 del apartado 5.4.

El fichero style.css únicamente se encarga del formato de la interfaz. Modifica el tamaño de fuente, la colocación, los colores y demás de los elementos de la interfaz.

En la figura 9 se puede observar la organización del paquete cliente web. Dada la simplicidad de la interfaz, vemos que se ha podido organizar todo el código en un fichero de cada tipo de lenguaje.

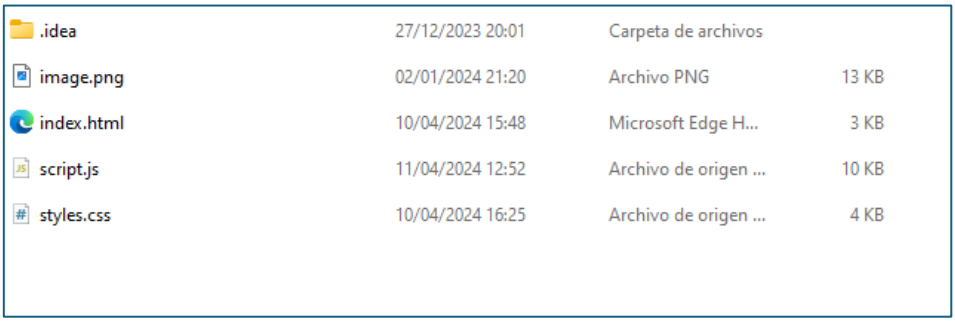


Figura 9 Estructura de carpetas del cliente

En la figura 10 se muestra la función utilizada para cargar los materiales que van a ser mostrados en el desplegable de la interfaz para que el usuario pueda elegir el material de cada capa. La función hace una llamada al servidor, llamando a la función materiales y esta retorna un archivo json con todos los materiales que tenemos en la base de datos. A continuación, se encarga de ir rellenando el select de la interfaz con los datos obtenidos de la petición al servidor.

En caso de que ocurra algún error, se muestra un mensaje de error por pantalla para que se pueda ver que el fallo está en esta petición y no en las posteriores.

```
function cargarMateriales(selectElement) {
  // Realizar la solicitud al servidor
  fetch('http://localhost:5000/materiales/label')
    .then(response => {
      if (!response.ok) {
        throw new Error('No se pudo cargar los materiales');
      }
      return response.json();
    })
    .then(data => {
      // Limpiar el contenido previo del elemento select
      selectElement.innerHTML = '';

      // Iterar sobre los materiales y agregar cada uno al desplegable
      data.forEach(material => {
        const option = document.createElement('option');
        option.value = material;
        option.textContent = material;
        selectElement.appendChild(option);
      });
    })
    .catch(error => console.error('Error al cargar materiales:', error));
}
```

Figura 10 Fragmento de código cargarMateriales() en frontend

En la figura 11 podemos ver la interfaz inicialmente mostrada en la aplicación. Inicialmente tenemos creadas la capas medio y sustrato cuyos grosores son infinitos y no se pueden eliminar. Además, nos sale creada una capa por defecto que se puede modificar y se puede eliminar. Tanto esta capa como las que vayamos añadiendo tienen un valor predefinido para el grosor que es 150 nm.

En la interfaz podemos ver los botones de añadir y eliminar al lado de cada columna, el botón eliminar elimina la fila sobre la que se pulsa y el botón añadir crea una nueva fila debajo de la señalada. Además, podemos ver el botón get color que calcula el color resultante de la operación.

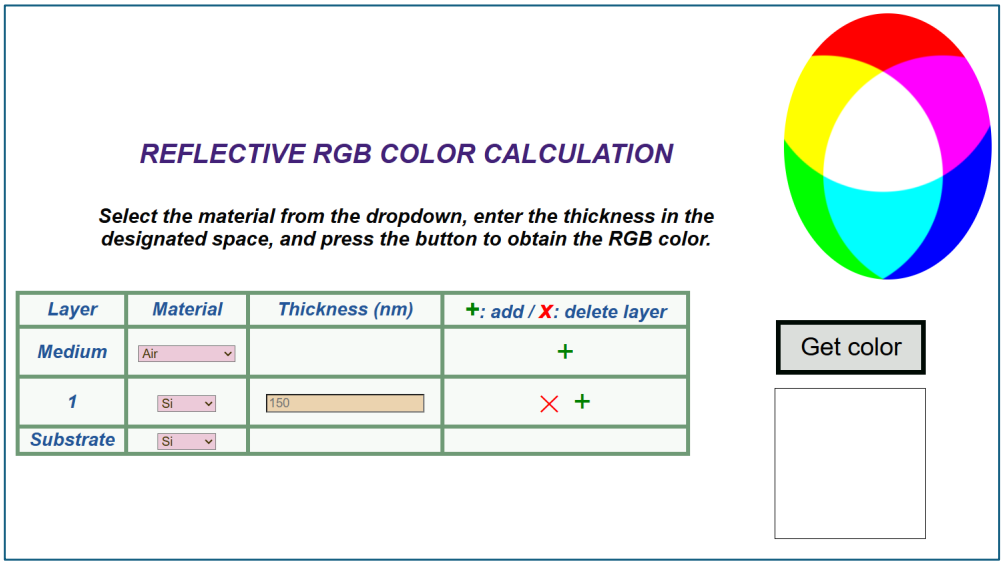


Figura 11 Imagen inicial de la interfaz al abrir la aplicación

En la figura 12, podemos observar el color resultante obtenido al añadir una capa debajo de la capa 1 con dos grosores editados manualmente.

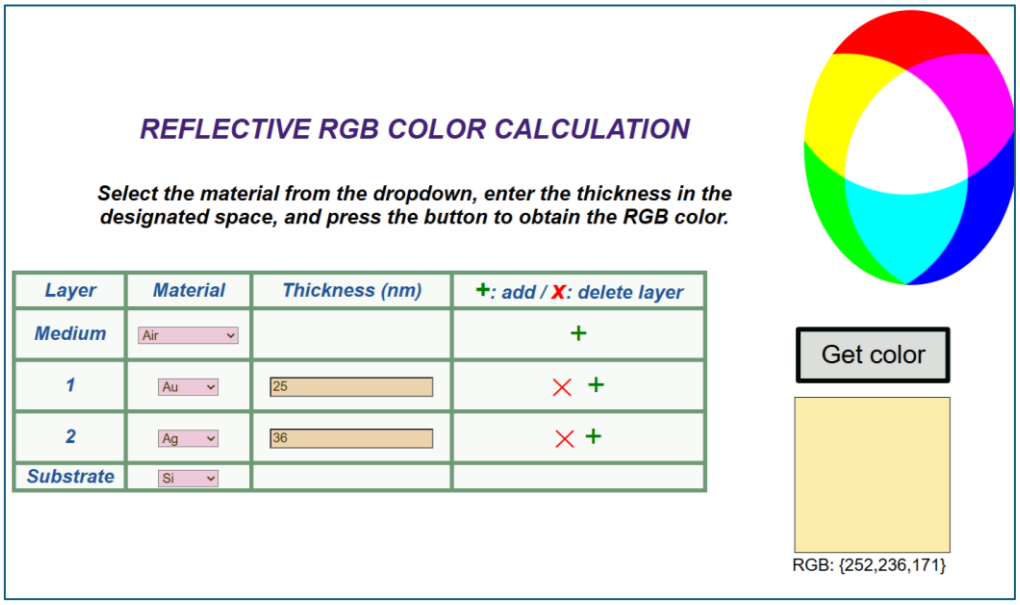


Figura 12 Interfaz mostrando un color

A continuación, en la figura 13 se puede ver las opciones de materiales que tenemos para modificar el material de la capa, basta con pulsar en el desplegable y elegir el material deseado.

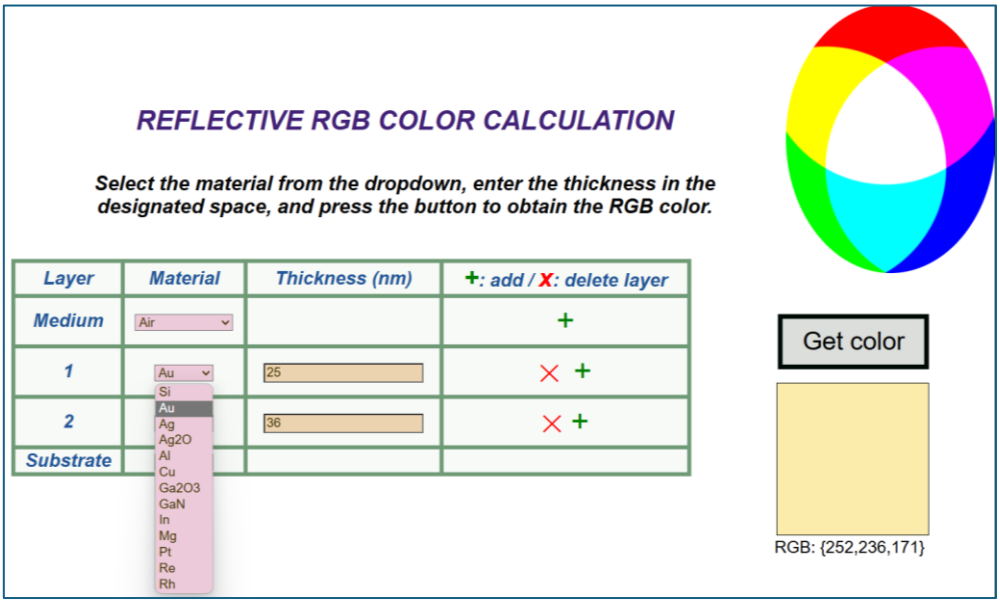


Figura 13 Interfaz desplegando lisa de materiales

En la figura 14 hemos añadido más filas y nos muestra un color distinto.

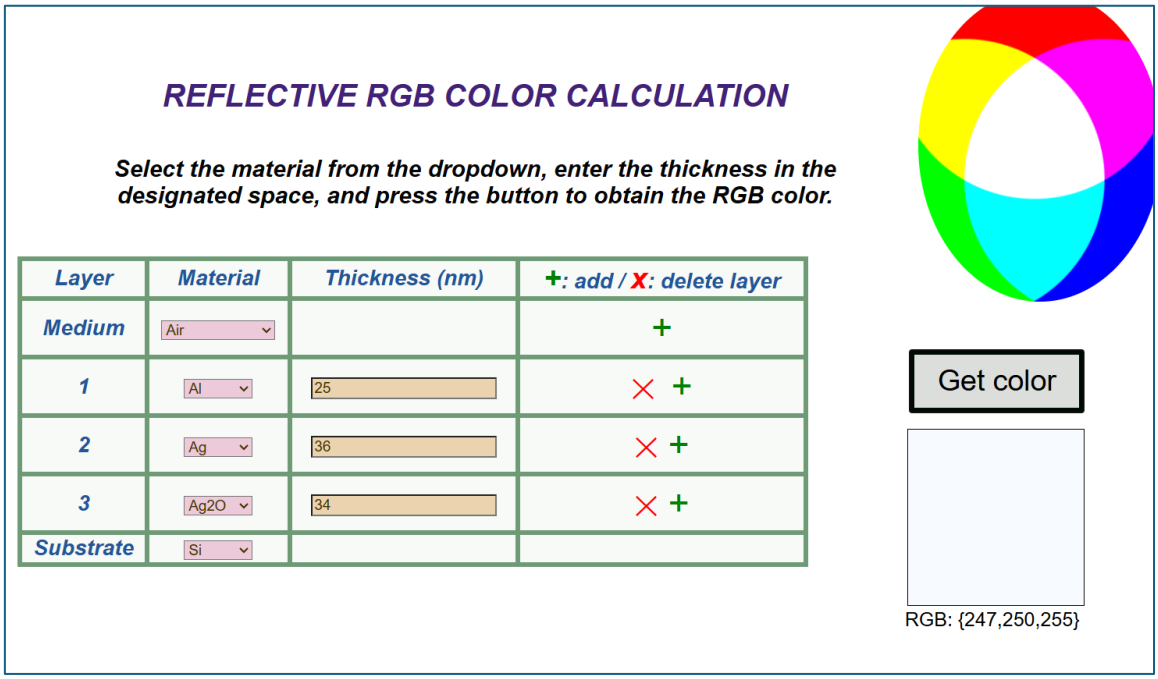


Figura 14 Interfaz después de añadir una capa

Posteriormente, en la figura 15 hemos eliminado dos filas y modificado el material de la fila restante y nos retorna un color completamente distinto.

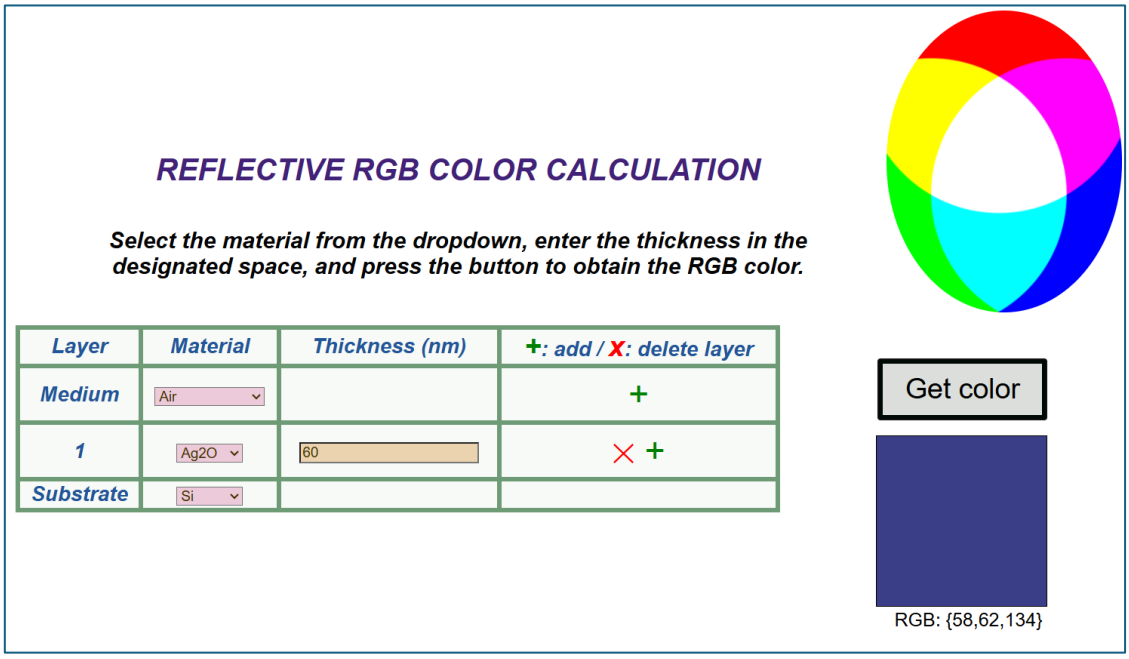


Figura 15 Interfaz después de eliminar capas

En la figura 16, hemos elegido la opción other material en vez de aire y hemos escrito un índice de refracción 2. El resultado varía respecto a la operación anterior.

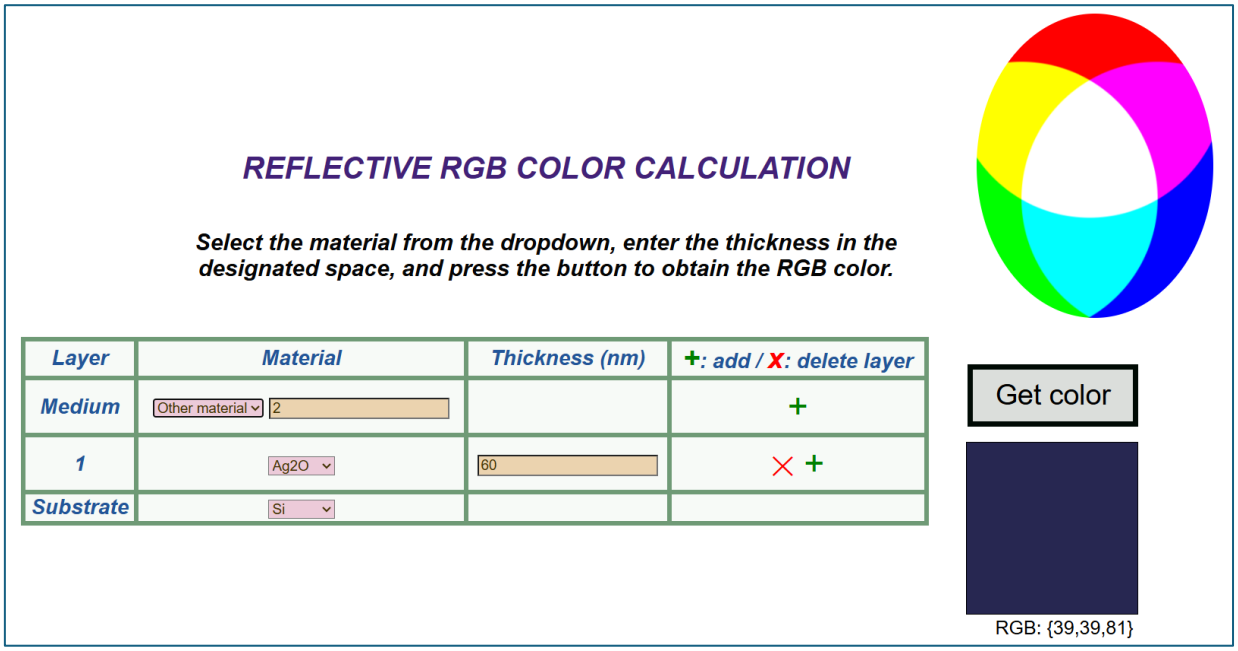


Figura 16 Interfaz mostrando opciones de la capa del medio

6.3 Backend

6.3.1 Api rest

La api rest ha sido realizada en el programa PyCharm y desarrollada en el lenguaje de programación Python.

En la tabla 9 se muestra la tabla de especificación de la api, con los dos métodos GET utilizados.

Recurso	***Método***	***Descripción***	***Parámetros***	**Respuesta esperada**	**Códigos de respuesta**
/materiales/	*GET*	*Devuelve el archivo JSON con el label de los materiales.*		Archivo JSON con todos los materiales	200, 404
/colors/<materiales>/<grosores>	*GET*	Calcula los valores RGB para los materiales y grosores especificados.	*materiales: lista de materiales separados por comas.* *grosores: lista de grosores separados por comas*	Valores RGB en formato JSON	200, 400, 404

Tabla 9 Tabla de recursos

Los dos métodos creados en la api son el método para obtener materiales y el método para calcular los valores RGB.

En la figura 17 podemos observar el método obtenerMateriales, este método se encarga de buscar el diccionario.json en su ruta y retorna ese fichero con el formato adecuado. Ese método va a ser utilizado por el cliente en el momento que selecciona el desplegable para elegir el material que desea en la capa seleccionada.

```python
@api.route('/materiales/label')
class obtenerNombreMateriales(Resource):
    def get(self):
        ruta_json = os.path.join(os.path.dirname(__file__), '..', '..', 'documents', 'catalogo.json')
        if os.path.exists(ruta_json):
            with open(ruta_json, 'r') as file:
                contenido = json.load(file)
                print("Contenido del archivo JSON:", contenido)  # Impresión para depuración

                if contenido:
                    # Obtener el primer elemento de la lista (índice 0)
                    nombres = [elemento["label"] for elemento in contenido]
                    return jsonify(nombres)



        else:
            # Si el archivo no existe, retorna un mensaje de error
            return {'error': 'Archivo JSON no encontrado'}, 404
```

Figura 17 Método obtenerMateriales()

El método calcula_color se encarga de adaptar los parámetros recibidos para que puedan ser utilizados por el método calcula_rgb y este método retorna en formato json una lista con las coordenadas RGB obtenidas.

En la figura 18 se puede ver cómo se da formato a la lista de grosores, la lista de materiales se realiza de una manera muy similar y se llama al método calcula_rgb con los valores obtenidos.

```python
        # Calcular los valores RGB
        d_list = [float('inf')] + grosores_list + [float('inf')]
        rgb_values = calcula_rgb([n_fn] + n_fn_list, d_list, th_0)
        rgb_values_list = rgb_values.tolist()

        # Crear un resultado con las variables individuales
        resultado = {
            'rgb': rgb_values_list
        }
        return jsonify(resultado)


```

Figura 18 Fragmento de la api que calcula el color RGB

6.3.2 Componente auxiliar que utiliza TMM y colorpy para calcular el color resultante

Para la realización de esta sección del código, se han extraído fragmentos de métodos que nos resultaban útiles de la librería TMM.

El método calcula_rgb mostrado en la figura 19 ha sido extraído del código obtenido de la librería TMM.

Este método recibe como parámetros una lista de materiales, una lista de grosores y un ángulo y retorna unas coordenadas IRGB.

En este punto se puede observar que hablamos de “RGB” cuando durante todo el trabajo se ha hablado siempre de RGB.

```
def calcula_rgb(n_fn_list, d_list, th_0):
    reflectances = color.calc_reflectances(n_fn_list, d_list, th_0)
    illuminant = colorpy.illuminants.get_illuminant_D65()
    spectrum = color.calc_spectrum(reflectances, illuminant)
    color_dict = color.calc_color(spectrum)
    return color_dict['irgb']
```

Figura 19 Método calcula_rgb()

6.4 Base de datos de ficheros con constantes ópticas

La base de datos de nuestro sistema consta de una lista para cada uno de los materiales de los que disponemos, como se puede ver en la figura 20. Cada fichero contiene los índices de reflexión de cada material.

Decidimos utilizar este formato de ficheros de texto en lugar de una base de datos debido a que solo necesitamos 3 columnas por material para calcular el índice de refracción y es mucho más accesible de esta manera. Además, en Python contamos con la Liberia pandas [6] que nos facilita la lectura d estos ficheros.

Además, nuestra base de datos contiene un catálogo, mostrado en la figura 20, donde se encuentran almacenados todos los materiales con su nombre e id para poder acceder a ellos de manera más fácil sin tener que acceder a varios ficheros cada vez que se necesite listarlos.

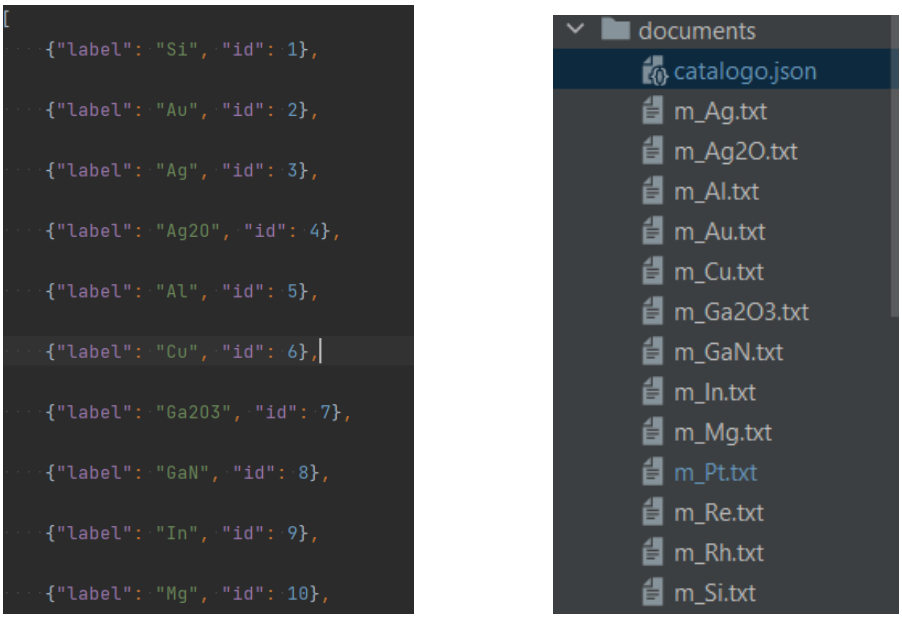


Figura 20 Izquierda: Catálogo Derecha Base de datos

El método leer_fichero representado en la figura 21 es el encargado de acceder al fichero del material correspondiente dentro de nuestra base de datos y realizar las operaciones necesarias para poder devolver un valor que pueda ser utilizado para calcular el color. Los ficheros de la base de datos contienen 3 columnas, por lo que se necesita tramitar cada una de las 3 por separado, para obtener el índice de refracción se necesita calcular el valor de las columnas dos y tres, el cual es un numero complejo formado por un número real y otro imaginario, lo cual resolvemos con el método complex de la librería math en Python.

Además, necesitamos realizar la interpolación entre las 2 columnas restantes, la resultante de la operación complex y la columna uno que no se ha modificado. Para ello utilizamos la función interp1d de la librería math de Python.

Este método ha sido creado desde cero, únicamente extraído de las necesidades de las clases de la librería TMM, teniendo en cuenta que se necesitaba realizar estas operaciones para darle total funcionalidad, pero en ningún momento lo encontramos en forma de método con esta única funcionalidad.

```
def leer_fichero(material):
    ruta = calcula_ruta(material)
    df = pd.read_csv(ruta, delimiter='\t')
    df['nk'] = df.apply(lambda fila: complex(fila.iloc[1], fila.iloc[2]), axis=1)
    df['lambda'] = df.iloc[:, 0]
    si_n_fn = interp1d(df['lambda'], df['nk'], kind='linear')
    # Si_n_data = si_n_fn(df['lambda'])

    return si_n_fn
```

Figura 21 Método leer_fichero() del servidor

7 Pruebas

Durante el desarrollo del proyecto se han ido realizando determinadas pruebas para comprobar el correcto funcionamiento del sistema, así como para detectar posibles errores.

Las pruebas realizadas han sido pruebas unitarias, pruebas de integración y pruebas de interfaz, las cuales se detallan en las secciones siguientes.

7.1 Pruebas unitarias

Para la realización de las pruebas unitarias hemos utilizado un módulo de pruebas integrado en Python que se llama “Unitest”. Unitest permite definir casos de prueba, agruparlos y ejecutarlos con facilidad. Además, ofrece aserciones para verificar el comportamiento esperado del código.

Hemos decidido realizar estas puestas sobre los métodos utilizados en el backend para el cálculo de colores de un sistema de materiales (componente descrito en la sección 6.3.2)
, para comprobar el correcto funcionamiento antes de utilizarlos para crear la api.

Estas pruebas han sido realizadas sobre el método calcula_rgb, método que se encarga de calcular el color RGB resultante recibiendo como parámetros una lista de materiales, una lista de grosores y un ángulo de reflexión. Este método fue extraído de la librería TMM y amoldado a nuestras necesidades por lo que la realización de las pruebas unitarias antes de utilizarlo se antojaba imprescindible.

Además de esto también realizamos pruebas unitarias sobre el método lee_fichero, método creado por nosotros desde cero, el cual también se antojaba imprescindible de probar antes de empezar a usarle.

Para la realización de estas pruebas, la Product Owner nos proporcionó sistemas multicapa y el color resultante asociado de forma que pudiéramos comprobar que el código funcionaba correctamente.

A continuación, vamos a ver algunas de las pruebas unitarias realizadas.

```
33      def test_sum3(self):
34          m_Ga2O3 = leer_fichero("Ga2O3")
35          m_Ag = leer_fichero("Ag")
36          air_n_fn = lambda wavelength: 1
37          n_fn_list = [air_n_fn, m_Ga2O3, m_Ag]
38          th_0 = 0
39          d_list = [inf, 2000, inf]
40          resultado_esperado = [244, 187, 98]
41          resultado_obtenido = calcula_rgb(n_fn_list, d_list, th_0)
42          for esperado, obtenido in zip(resultado_esperado, resultado_obtenido):
43              self.assertAlmostEqual(esperado, obtenido, delta=5)
44
```

Figura 22 Prueba unitaria

```python
    def test_sum6(self):
        m_GaN = leer_fichero("GaN")
        m_Ga2O3 = leer_fichero("Ga2O3")
        m_Ag = leer_fichero("Ag")
        const1 = lambda wavelength: 1.33
        const2 = lambda wavelength: 5.0
        n_fn_list = [const1, m_GaN, m_Ga2O3, m_Ag, m_Ga2O3, m_GaN, const2]
        th_0 = 0
        d_list = [inf, 50, 30, 2, 30, 50, inf]
        resultado_esperado = [174, 112, 106]
        resultado_obtenido = calcula_rgb(n_fn_list, d_list, th_0)
        for esperado, obtenido in zip(resultado_esperado, resultado_obtenido):
            self.assertAlmostEqual(esperado, obtenido, delta=5)

    def test_sum7(self):
        m_GaN = leer_fichero("GaN")
        m_Ga2O3 = leer_fichero("Ga2O3")
        const1 = lambda wavelength: 1
        const2 = lambda wavelength: 2
        n_fn_list = [const1, m_GaN, m_Ga2O3, m_GaN, m_Ga2O3, m_GaN, const2]
        th_0 = 0
        d_list = [inf, 50, 30, 50, 30, 50, inf]
        resultado_esperado = [124, 90, 165]
        resultado_obtenido = calcula_rgb(n_fn_list, d_list, th_0)
        for esperado, obtenido in zip(resultado_esperado, resultado_obtenido):
            self.assertAlmostEqual(esperado, obtenido, delta=5)


if __name__ == "__main__":
    unittest.main()
```

Figura 23 Pruebas unitarias realizadas

Estos dos casos de pruebas unitarias calculan el color RGB con una serie de materiales y grosores y realizamos la comparación con los valores reales proporcionados por la Product Owner. Por consejo de la Product Owner, hemos dejado un margen de error de 5 valores en el rgb, ya que es una variación casi insignificante en este ámbito y es casi imposible de evitar. Por ejemplo, una diferencia pequeña en la dimensión R podría compensarse con pequeños cambios en la G y la B, de forma que el color obtenido sea prácticamente el mismo.

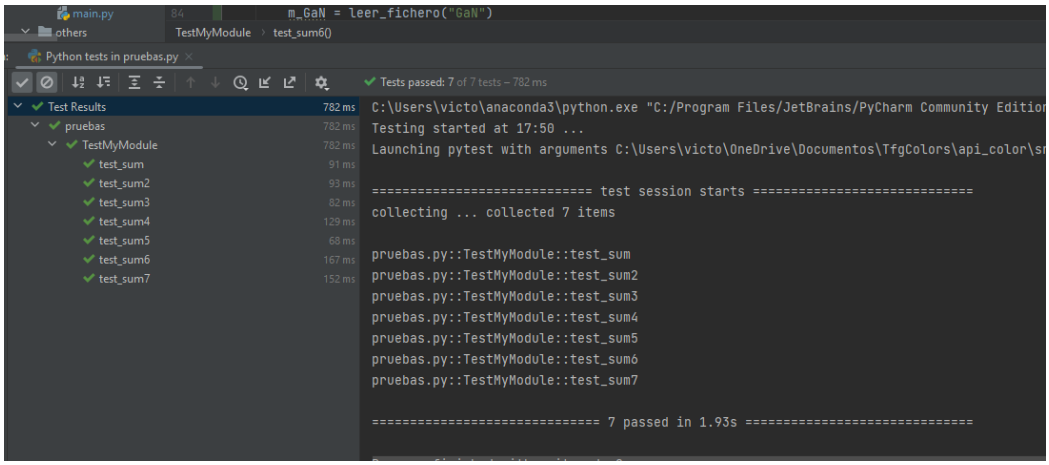


Figura 24 Resultado pruebas unitarias

7.2 Pruebas de integración

Para la realización de las pruebas de integración hemos utilizado Postman.

Inicialmente optamos por posponer la automatización de estas pruebas y realizarlas de manera manual ejecutando los métodos de la api en un navegador, inicializando el servidor en PyCharm y accediendo a la dirección adecuada con los parámetros.

Según íbamos avanzando en el desarrollo del trabajo decidimos que esto se quedaba bastante escaso y empezamos a trabajar en realizar unas pruebas de integración con Postman.

Caso 1: Éxito

Este caso de prueba representa un caso exitoso. Invocamos al método colors y le pasamos como parámetros los materiales,air,si,au,au y como grosores 2.5 y 2. Aprovechamos para recordar que las capas inicial y final no requieren grosor ya que este es infinito.

Este caso retorna un resultado en formato json como podemos ver en la figura 26 y cómo podemos ver en la figura 25 muestra un estado de acierto porque el RGB es correcto. Este caso de prueba se llevó a cabo en un momento anterior al siguiente caso de prueba, cuando aún no disponíamos de la totalidad de los materiales en nuestra base de datos.

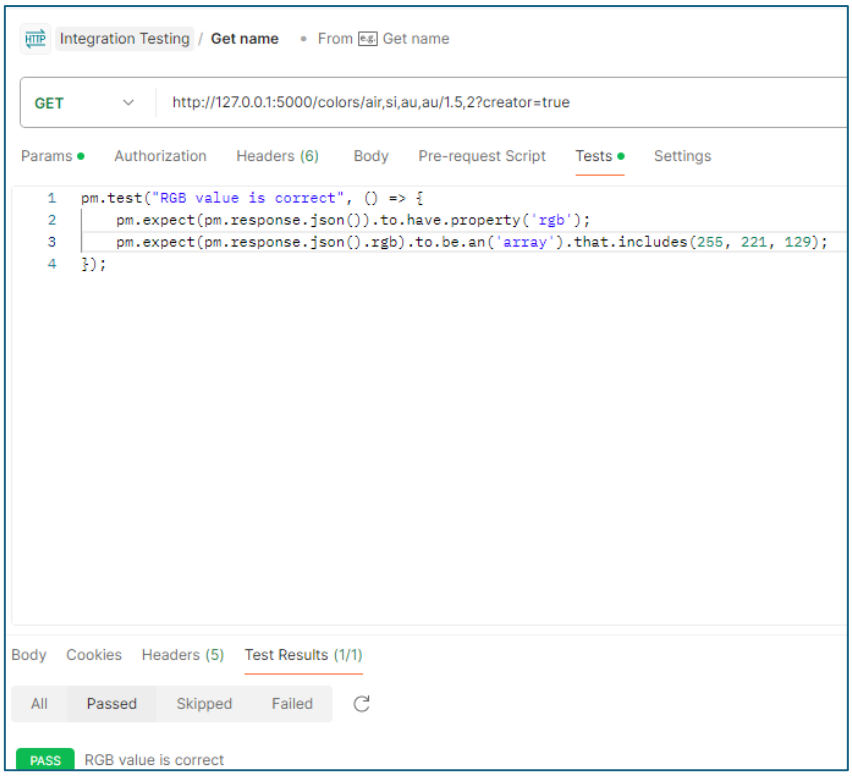


Figura 25 Prueba integración exitosa

Figura 26 Resultado primera prueba de integración

Caso 2: Éxito

Este caso de prueba es muy similar al anterior, la única diferencia es que fue creado cuando dispusimos de todos los datos de los materiales, la comprobación es idéntica al anterior, se le pasan unos materiales y unos grosores y se comprueba que el RGB resultante es el correcto. Como en este caso el resultado es correcto se retorna que la prueba es válida.

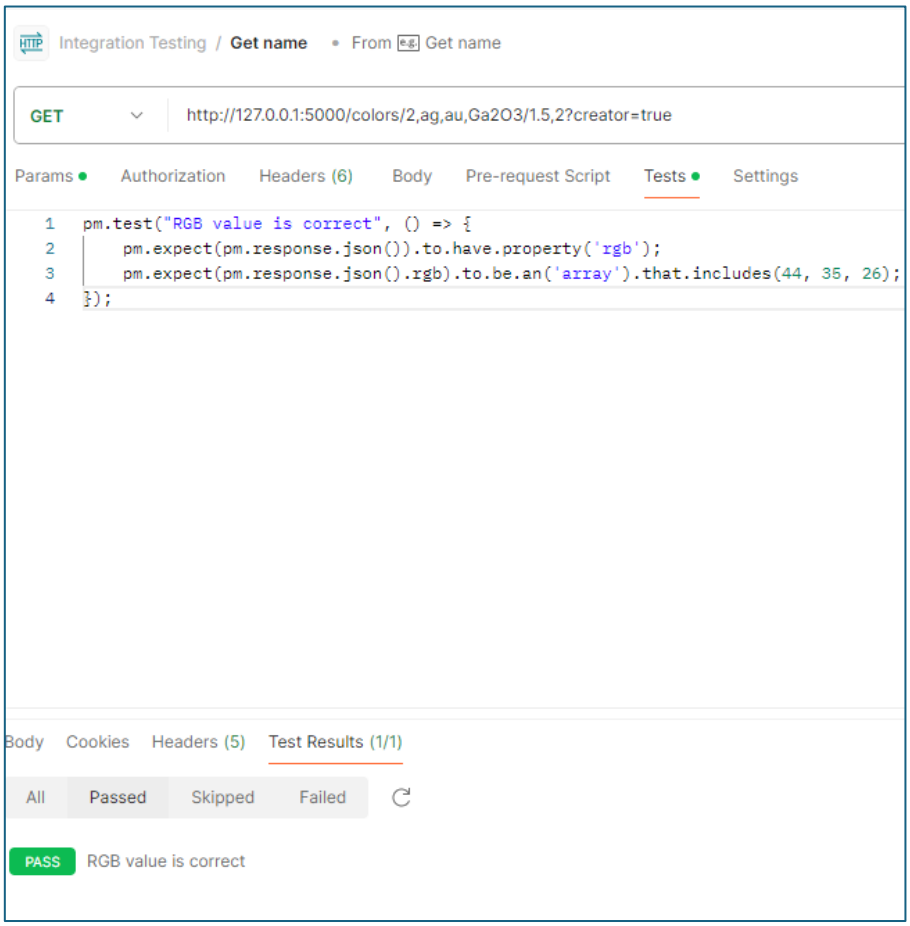


Figura 27 Prueba de integración exitosa

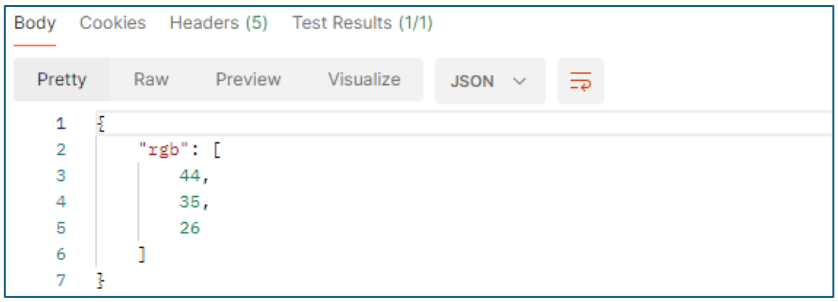


Figura 28 Resultado prueba de integración

Caso de prueba 3: Error

En este caso se guimos probando el método colors, le pasamos una lista de materiales como parámetros que contenga un material distinto de aire o de un valor numérico en el primer campo. El resultado es error y nos muestra el emnsaje de que el primer material no es válido, ya que para el primer material solo se permite el material aire o introducir el indice de refracción de otro material de manera numérica.

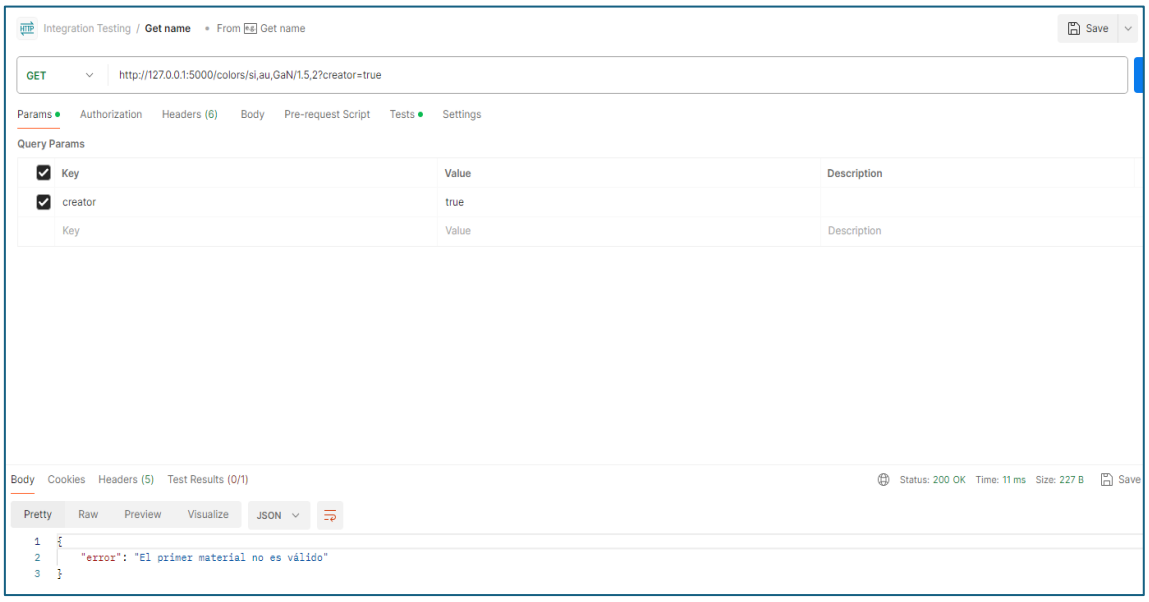


Figura 29 Prueba de integración primer material no válido

Caso de prueba 4: Error

En este caso probamos a pasar como parámetro respectivo al grosor de una capa un número negativo. Observamos que el test falla y muestra el error por pantalla avisándonos de que en grosor debe ser un número mayor que cero.

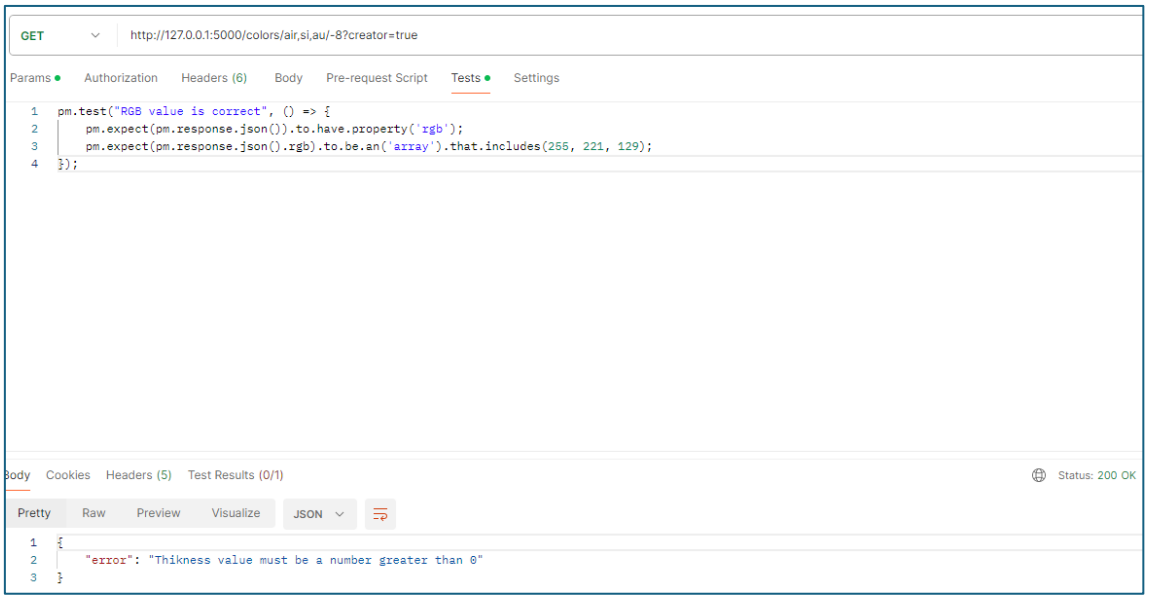


Figura 30 Prueba de integración grosor menor que 0

Caso de pueba 5: Error

En este caso probamos a pasar como parámetro respectivo al grosor de una capa un valor un numérico. Observamos que el test falla y muestra el error por pantalla avisándonos de que en grosor debe ser un número.

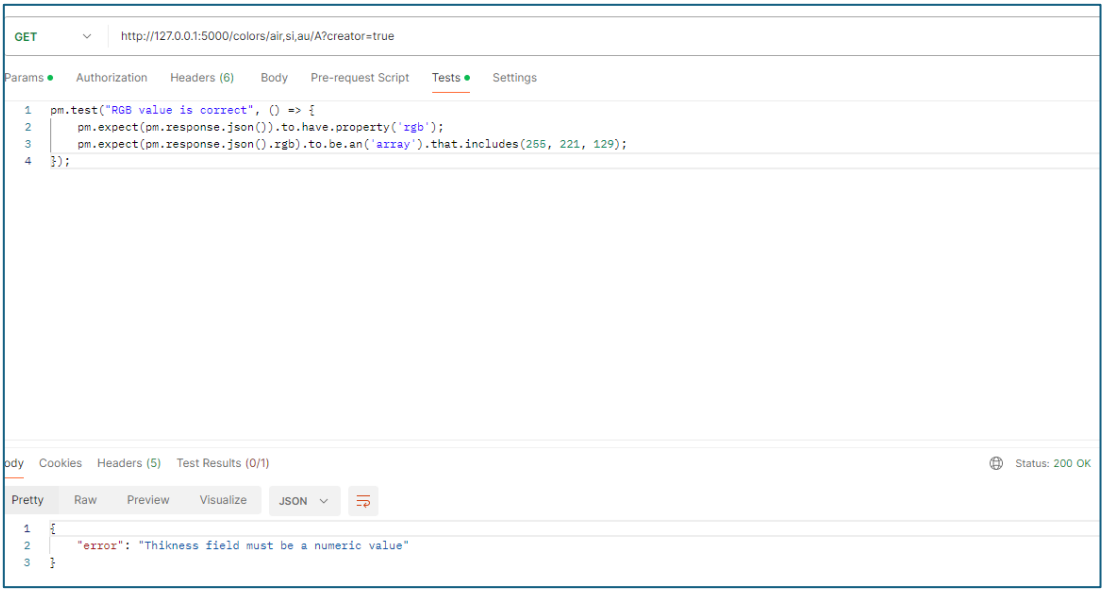


Figura 31 Prueba de integración grosor no numérico

Caso de prueba 6: Éxito

En este caso de prueba estamos probando el método materiales. Este metodo lo único que debe hacer es retornar toda la lista de materiales que se encuentra en un json llamado diccionario dentro de la base de datos del servidor.

En este caso de prueba unicamente comprobamos que el retorno del método se corresponde con nuestro fichero de datos.

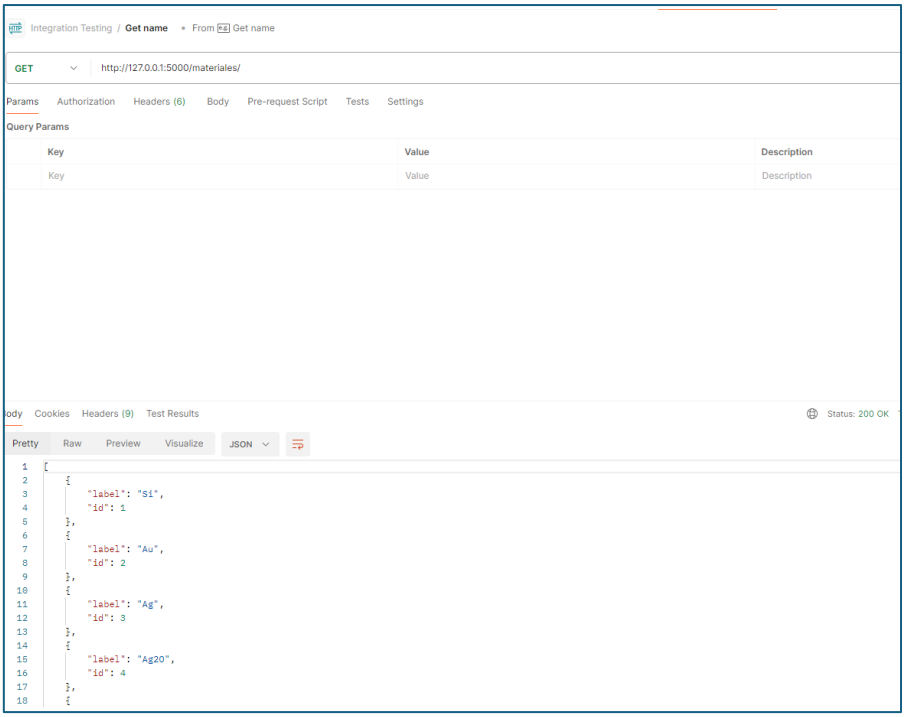


Figura 32 Prueba de integración exitosa método materiales ()

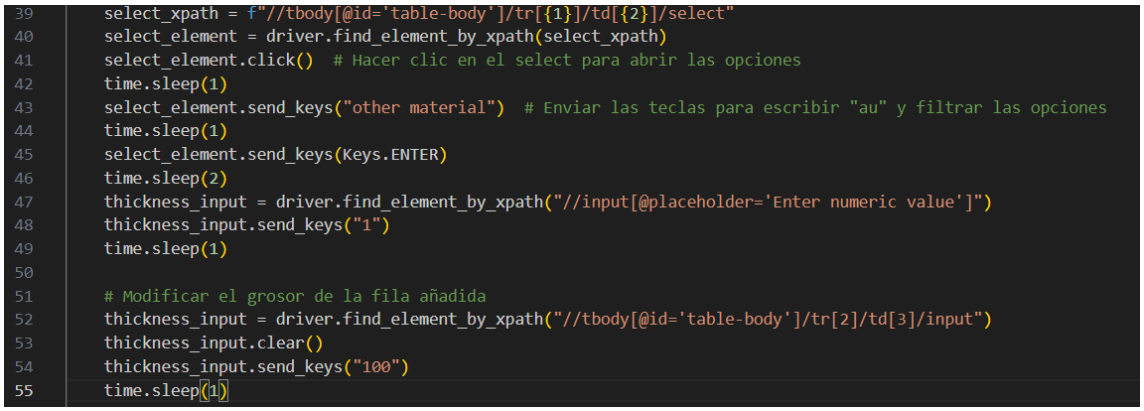
7.3 Pruebas de interfaz

Para la realización de las pruebas de interfaz hemos utilizado Selenium a través de python.

Selenium es un entorno de pruebas de software para aplicaciones basadas en la web. Selenium proporciona una herramienta para crear pruebas sin usar un lenguaje de scripting para pruebas llamada Selenium IDE. Las pruebas pueden ejecutarse usando la mayoría de los navegadores web modernos en diferentes sistemas operativos como Windows, Linux y OSX.

Hemos realizado un caso de pruebas que interactúe con todas las funcionalidades de la interfaz para poder comprobar que todo funciona correctamente.

En las figuras 33 y 34 se puede observar cómo se han ido llevando a cabo las pruebas.



```
39      select_xpath = f"//tbody[@id='table-body']/tr[{1}]/td[{2}]/select"
40      select_element = driver.find_element_by_xpath(select_xpath)
41      select_element.click()  # Hacer clic en el select para abrir las opciones
42      time.sleep(1)
43      select_element.send_keys("other material")  # Enviar las teclas para escribir "au" y filtrar las opciones
44      time.sleep(1)
45      select_element.send_keys(Keys.ENTER)
46      time.sleep(2)
47      thickness_input = driver.find_element_by_xpath("//input[@placeholder='Enter numeric value']")
48      thickness_input.send_keys("1")
49      time.sleep(1)
50
51      # Modificar el grosor de la fila añadida
52      thickness_input = driver.find_element_by_xpath("//tbody[@id='table-body']/tr[2]/td[3]/input")
53      thickness_input.clear()
54      thickness_input.send_keys("100")
55      time.sleep(1)
```

Figura 33 Fragmento de código de pruebas de interfaz

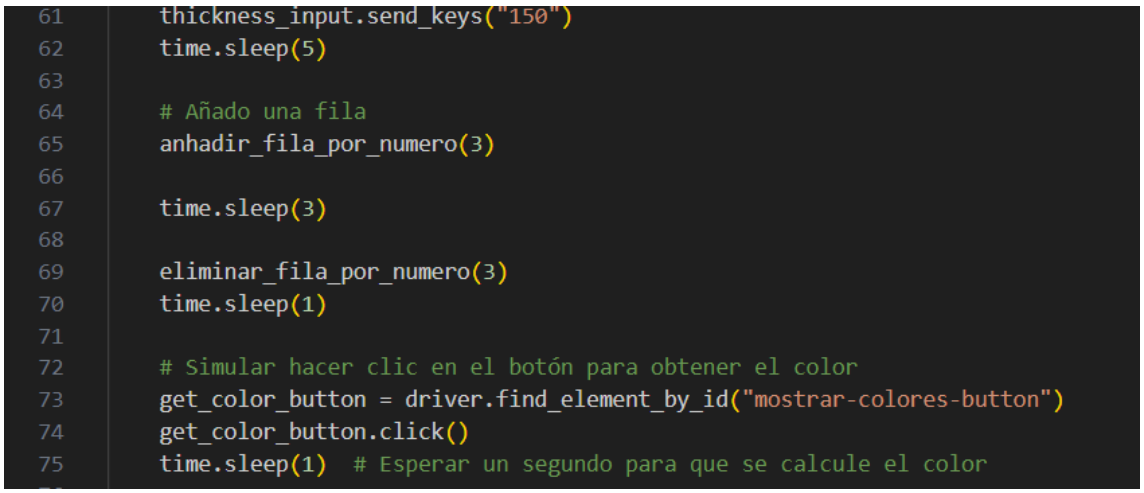
En la figura 33, en la línea 43, se puede ver como se selecciona el campo “other material” en la fila 1, y se añade manualmente un índice de refracción 1.

A continuación, en la figura 33, fila 54, se puede ver como se añade el valor 100 al grosor de la capa que se ha creado inicialmente utilizando el botón de añadir fila sobre la primera fila.

En la figura 34, en la línea 61, se puede observar cómo se añade el valor 150 al campo de grosor de la fila existente en la tabla creada por defecto al ejecutar el programa.

A continuación, probamos a añadir una fila debajo de la fila tres, figura 34, fila 65, y a eliminar la fila tres, en la fila 69.

Por último, utilizamos el botón de obtener color para comprobar que realiza correctamente la operación, muestra las coordenadas RGB y muestra el color correspondiente.



```
61      thickness_input.send_keys("150")
62      time.sleep(5)
63
64      # Añado una fila
65      anhadir_fila_por_numero(3)
66
67      time.sleep(3)
68
69      eliminar_fila_por_numero(3)
70      time.sleep(1)
71
72      # Simular hacer clic en el botón para obtener el color
73      get_color_button = driver.find_element_by_id("mostrar-colores-button")
74      get_color_button.click()
75      time.sleep(1)  # Esperar un segundo para que se calcule el color
```

Figura 34 Fragmento de código de pruebas de interfaz

8 Conclusiones

El objetivo de este Trabajo de Fin de Grado ha sido crear una aplicación web que proporcione a los profesionales e investigadores en óptica de materiales la capacidad de calcular el color resultante de la reflexión de la luz sobre un conjunto de capas de materiales basándonos en el índice de refracción de cada material, el grosor de éste y el ángulo con el que incide la luz sobre estas capas.

Se considera que se ha alcanzado el objetivo, ya que se dispone de una aplicación plenamente funcional y que cumple con los requisitos especificados.

A lo largo de estos meses se ha desarrollado cada una de las partes de la aplicación, probándolas tanto de forma individual como siendo parte de un único sistema, realizando durante el propio desarrollo pequeñas mejoras que pudieran afectar al rendimiento o a la experiencia de usuario.

Uno de los puntos a priori más difíciles detectados durante la preparación del proyecto fue la creación de un servidor utilizando el lenguaje Python y la creación de la conexión con el cliente utilizando este mismo lenguaje. Esto se debe a que, durante toda la carrera, aunque hemos trabajado con Python y hemos creado paginas web en algunas asignaturas, todo el trabajo relacionado con cliente y servidor, conexiones y demás se ha realizado utilizando el lenguaje java. Al comienzo de este trabajo, el director del proyecto me propuso realizarlo utilizando Python ya que teníamos ya acceso a las librerías TMM y ColorPy, lo cual me facilitaría el trabajo de búsqueda previa antes de comenzar el trabajo. Ante esto decidí afrontarlo como una oportunidad de desarrollarme en un lenguaje no tan conocido y utilizado por mí hasta el momento.

Considero que la decisión fue totalmente acertada, las librerías facilitadas me ayudaron mucho y creo que he mejorado mucho usando Python y me ha ayudado a comprender que se puede afrontar cualquier desafío relacionado con la programación partiendo de las bases adquiridas durante la carrera.

Además, he adquirido muchos conocimientos sobre la creación de páginas web, y afianzado los conocimientos adquiridos durante la carrera. En resumen, me he dado cuenta de que puedo afrontar retos que antes de este proyecto no consideraba posibles.

Por todo ello, a título personal, me considero satisfecho con el trabajo realizado. Sin duda, este aprendizaje me será muy útil en el futuro de cara a afrontar nuevos proyectos.

9 Trabajos futuros

Este trabajo no deja de ser la primera versión de una idea a desarrollar, por lo que siempre se pueden añadir mejoras.

Como en la mayoría de proyectos software, la capacidad de mejora es prácticamente infinita, siempre se puede hacer la interfaz de usuario un poco más bonita, o un poco más amigable para el usuario, además de añadir alguna nueva funcionalidad.

Durante el desarrollo se ha hablado de algunos puntos que podrían desarrollarse para mejorar la usabilidad del programa, que finalmente se decidió que era mejor no llevar a cabo su implementación. A continuación, vamos a comentar alguno de ellos:

- Una funcionalidad interesante seria permitir a los usuarios poder registrarse en la aplicación, de manera que puedan guardar sus configuraciones favoritas, que puedan guardar un historial de cálculos, o que puedan configurar los campos que deseen para que les salga un valor por defecto cada vez que inicien la aplicación.

- Otra funcionalidad de la que hablamos, pero nunca se incluyó para este trabajo fue la de poder exportar los resultados, de manera que el usuario pueda guardar sus resultados en un pdf o en una foto y tenerlo para su uso personal cuando lo desee.

- Durante el desarrollo del proyecto decidimos que únicamente íbamos a tratar con una unidad de medida para el grosor de las capas, pero podría facilitar la usabilidad a los clientes poder elegir que unidad de medida quieren usar y así evitar obligarles a tener que convertir sus datos.

- Otra funcionalidad interesante sería permitir introducir manualmente el índice de refracción de un material en cada capa, en lugar de elegir el material y que el programa calcule automáticamente ya que puede permitir utilizar algún material no existente en la base de datos.

- Por último, una posibilidad para expandir el campo de nuestra aplicación seria adaptar la interfaz de usuario para su funcionamiento correcto en interfaces móviles. Para ello, a nivel frontend, se deberá modificar el diseño de los componentes de manera que sean “responsivos”, es decir, que se adapten a la altura y anchura de la pantalla del dispositivo.

10 Referencias

[1] Repositorio en GitHub del proyecto. Último acceso: junio de 2024: https://github.com/vac686/TFGcolors

[2] Flask. Último acceso: junio de 2024: Welcome to Flask — Flask Documentation (3.0.x) (palletsprojects.com)

[3] Python. Último acceso: junio de 2024: https://www.python.org/downloads/

[4] Librería TMM. Último acceso: junio de 2024: https://pypi.org/project/tmm/

[5] Librería Colorpy. Último acceso: junio de 2024: https://pypi.org/project/colorpy/

[6] Librería Pandas. Último acceso: junio de 2024: https://pandas.pydata.org/

[7] PyCharm. Último acceso: junio de 2024: https://www.jetbrains.com/pycharm

[8] HTML. Último acceso: junio de 2024: https://developer.mozilla.org/es/docs/Web/HTML

[9] JavaScript. Último acceso: junio de 2024: https://developer.mozilla.org/es/docs/Web/JavaScript

[10] CSS. Último acceso: junio de 2024: https://developer.mozilla.org/es/docs/Web/CSS

[11] Visual Studio Code. Último acceso: junio de 2024: https://code.visualstudio.com/download

[12] SourceTree. Último acceso: junio de 2024: https://www.sourcetreeapp.com/

[13] Git. Último acceso: junio de 2024: https://www.git-scm.com/

[14]Postman. Último acceso: junio de 2024: https://www.postman.com/lemolina/workspace/estudiante-experto-en-conceptos-bsicos-de-la-api-de-postman/collection/28608847-acde320c-d0c4-47d4-9b91-ca95f96de303

[15] Selenium. Último acceso: junio de 2024: https://www.selenium.dev/

[16] Flask-RESTX. Último acceso: junio de 2024: https://flask-restx.readthedocs.io/en/latest/