GenList

INFORME TÉCNICO PARA SISTEMA DE MONITOREO DE ACCESO BASADO EN RECONOCIMIENTO FACIAL PARA EL CONTROL Y LA SUPERVSION PRECISA EN UN SALON DE CLASE

Autor 1. Samuel Andres Celis Lizcano

Autor 2. Juan Pablo Marquez Sanchez

Autor 3. Yorman Rodolfo Rodriguez Jaimes

Ingeniería de Software II

Ingeniería de Sistemas

Universidad de Pamplona

27 / 11 / 2024

GenList

GenList

Contenido

GenList

1. Resumen Ejecutivo

- **Objetivo principal de la aplicación:** Control de Acceso Basado en Reconocimiento Facial para la Supervisión Precisa en un Salón Clase.
- **Funcionalidades clave:** Reconocimiento Facial, Tomar asistencias, Generar listas de asistencias, etc.
- **Beneficios para el usuario:** fortalece la seguridad y la eficiencia.
- **Resultados destacados:** Asegura una experiencia educativa más fiable y equitativa.

GenList

2. Introducción

2.1. Contexto del problema

Para nadie es un secreto que hoy en día la toma de asistencia en algunos lugares que solicitan la misma es un poco accesible a fraudes, en la mayoría de las aulas estudiantiles las asistencias son llevadas de forma física (planillas de firmas), en algunos casos son llevadas en hojas de Excel. El problema radica en que ambos métodos son alterables por compañeros o el mismo docente que no desea poner inasistencias.

2.2. Objetivos específicos del proyecto

1. Identificar herramientas y tecnologías viables para el desarrollo de un sistema de reconocimiento facial
2. Establecer lenguajes y tecnologías factibles identificados en documentos y publicaciones relevantes para el desarrollo de un sistema de reconocimiento facial.
3. Crear un sistema de control de acceso basado en reconocimiento facial para la supervisión precisa en un salón clase
4. Validar el correcto funcionamiento del sistema de control de Acceso Basado en Reconocimiento Facial con datos tomados en un salón de clases

2.3. Alcance del proyecto

Se incluyo la funcionalidad de generar PDF, no se descarto ninguna funcionalidad durante el desarrollo.

3. Requerimientos del Sistema

3.1. Funcionales

Crear Clases, tomar asistencia, generar lista de asistencia.

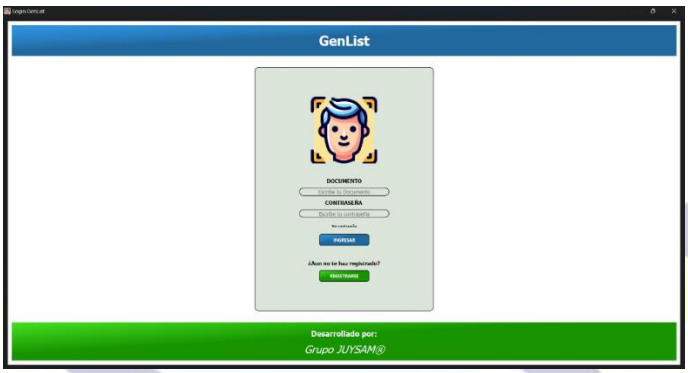
3.2. No funcionales

Seguridad, rendimiento, escalabilidad.

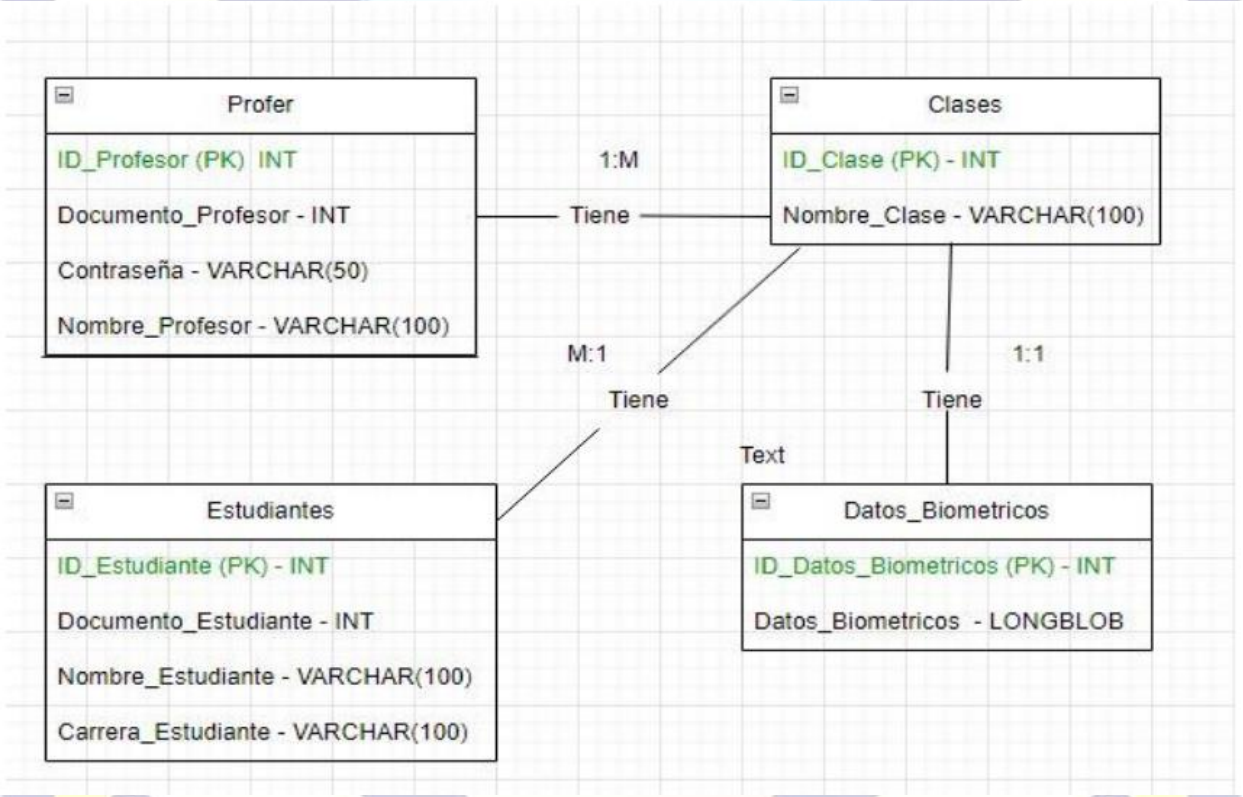
4. Diseño

4.1. Arquitectura de la aplicación

4.2. Diseño de la interfaz de usuario (UI):



4.3. Base de datos



Se eligió MySQLite por su funcionamiento ágil con una base de datos simple, como es acorde a la información que se trató.

5. Desarrollo

- **Tecnologías utilizadas**

OpenCV es una biblioteca de software de código abierto que permite a los desarrolladores acceder a rutinas utilizadas para aplicaciones de visión por computadora en la API (Interfaz de Programación de

Aplicaciones). Si alguien hace clic en el timbre de la puerta o si la cámara detecta actividad sospechosa, la imagen se grabará utilizando el código de software para la detección de actividad escrito en Python y OpenCV. Después de que la imagen esté registrada, se detecta y segmenta el código de Python y OpenCV del fotograma. El código de reconocimiento facial para identificar las caras almacenadas en la base de datos se proporciona como una cara segmentada.

Detección de objetos y seguimiento de su posición en tiempo real con Raspberry Pi

Una de las áreas de rápido crecimiento del aprendizaje profundo mediante inteligencia artificial es la visión por ordenador, cada vez más popular. Se trata de un campo de investigación en auge que busca crear técnicas que ayuden a los ordenadores a "ver" y reconocer la información de imágenes digitales como, por ejemplo, vídeos y fotografías. La detección de objetos es un método de visión por ordenador que nos permite reconocer objetos en una imagen o vídeo y localizarlos. Este artículo describe un método eficiente de identificación de objetos basado en la forma y su desplazamiento en tiempo real utilizando la librería OpenCV de roles de programación mayoritariamente orientados a visión por computador y Raspberry Pi con módulo de cámara

- **Metodología de desarrollo:** Ágil XP
- **Fases del desarrollo:** Planificación, diseño, codificación, pruebas y lanzamiento.
- **Desafíos y soluciones:** Se encontró el problema al desarrollar la interfaz de usuario, ya que el desarrollar una interfaz grafica en Python no es amigable con su desarrollador, se soluciono con resiliencia y una considerable cantidad de tiempo.

6. Pruebas

- **Tipos de pruebas realizadas:** Pruebas Unitarias, Pruebas de Interfaz de Usuario, Pruebas de Integración, Pruebas Funcionales/End-to-End y Prueba de la precisión del modelo.
- **Herramientas de prueba utilizadas:** la librería unittest de Python
- **Resultados de las pruebas:** Identificación de errores y mejoras realizadas

6.1. Enfoque de pruebas

Como ya se comentó las pruebas serán divididas dependiendo del componente que se desee probar, sin embargo, todos los componentes estarán siendo evaluados con Pruebas Unitarias. Pero cada componente tendrá ciertas pruebas especiales, las cuales son:

Front-End: Pruebas de Interfaz de usuario, Pruebas de Integración.

Back-End: Pruebas de Integración, Pruebas Funcionales.

Modelo Predictivo: Prueba de la precisión del modelo, Prueba de Carga.

6.2. Recursos de Pruebas

- **Personal de Pruebas:** Las pruebas serán realizadas por cada desarrollador en su área, en este caso por el encargado del Front-End (Samuel Celis), el encargado del Back-End (Yorman Rodriguez) y el encargado del modelo predictivo (Juan Marquez).
- **Ambiente de Pruebas:** Todas las pruebas serán realizadas en el editor de código Visual Studio Code, utilizando las librerías de Python especializadas en Pruebas Unitarias. En caso de las pruebas de Interfaz serán realizadas ejecutando la interfaz para verificar y comprobar el comportamiento y la ubicación de los elementos.
- **Herramientas de Pruebas:** Se utilizará la librería unittest de Python, los computadores de los desarrolladores y el editor de código Visual Studio Code.

6.3. Planificación de Pruebas

Front-End		
Pruebas	**Fecha de Inicio**	**Fecha de Finalización**
Pruebas Unitarias **(Unit Test)**	19 de octubre	25 de octubre
Pruebas de Interfaz de Usuario **(UI Test)**	26 de octubre	1 de noviembre
Pruebas de Integración	2 de noviembre	8 de noviembre

Back-End		
Pruebas	**Fecha de Inicio**	**Fecha de Finalización**
Pruebas Unitarias **(Unit Test)**	9 de noviembre	15 de noviembre
Pruebas de Integración	16 de noviembre	22 de noviembre
Pruebas Funcionales/End-to-End **(E2E)**	23 de noviembre	29 de noviembre

Modelo Predictivo		
Pruebas	**Fecha de Inicio**	**Fecha de Finalización**
Prueba de la precisión del modelo	30 de octubre	5 de noviembre
Prueba de la carga del modelo	6 de noviembre	8 de noviembre
Prueba del preprocesamiento de imágenes	9 de noviembre	11 de noviembre

6.4. Criterios de Aceptación

Para considerar de forma exitosa cada prueba realizada se deben obtener los resultados esperados, por cada uno de los desarrolladores.

Front-End: Carga de los elementos en los lugares esperados, movimiento entre las pantallas de la aplicación, funcionamiento de los elementos individualmente

Back-End: Correcta gestión de datos (almacenamiento, recuperación y actualización), respuestas consistentes a las solicitudes del front-end, manejo adecuado de errores, autenticación y autorización seguras, y un rendimiento eficiente bajo carga.

Modelo Predictivo: precisión adecuada en las predicciones sobre los datos de prueba (superior al 85%), correcta carga y ejecución del modelo sin errores, procesamiento eficiente y adecuado de las imágenes de entrada (incluyendo redimensionamiento y normalización), y tiempos de predicción dentro de límites aceptables para su uso en tiempo real o cercano al tiempo real.

7. Implementación:

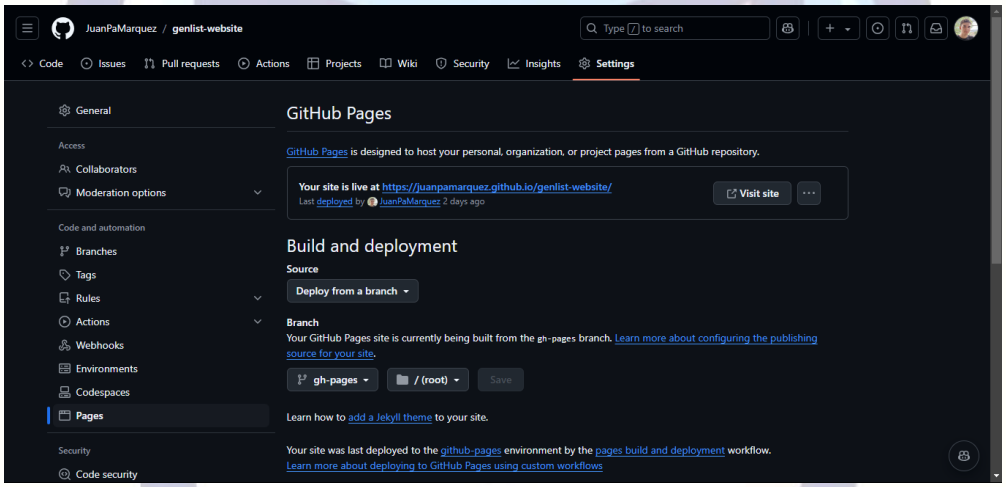
Para la parte de despliegue de nuestro sistema al ser una aplicación de escritorio creamos una página web donde los usuarios podrán descargar nuestra aplicación:

Donde pueden ver la descripción del proyecto:

Luego vemos la sección de quienes somos:

GenList

Este sitio se publicó en GitHub Pages:



En la configuración del archivo package.json de la página web se hizo esta configuración:

```
{
  "name": "genlist-website",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "homepage": "https://juanpamarquez.github.io/genlist-website",
  ▷ Depurar
  "scripts": {
    "dev": "vite",
    "build": "tsc -b && vite build",
    "lint": "eslint .",
    "preview": "vite preview",
    "predeploy": "npm run build",
    "deploy": "gh-pages -d dist"
  },
```

Y para el empaquetamiento de la aplicación usamos la librería de Python pyinstaller para crear nuestro archivo exe que los usuarios podrán descargar desde la página web explicada anteriormente.

Nuestro despliegue fue **directo** ya que nosotros cambiamos el archivo de descarga cada vez que haya una nueva actualización.

También usamos despliegue **Serverless** porque alojamos la página web de descarga en GitHub Pages donde no tenemos control del servidor, pero nos permite alojar el sitio web de forma gratuita.

8. Resultados

Creación de una aplicación con un alto valor para los docentes, buscando ayudar con el proceso del reporte de asistencias ante la universidad, optimizando los tiempos y minimizando los gastos.

9. Conclusiones:

- La identificación de herramientas y tecnologías viables fue crucial para garantizar la eficiencia y precisión de un sistema de reconocimiento facial. Estas herramientas permitieron el desarrollo de algoritmos complejos de manera eficiente y flexible. La elección se alineó con los requisitos específicos del proyecto y las habilidades del equipo de desarrollo.
- La revisión de documentos y publicaciones relevantes permitió identificar lenguajes de programación y tecnologías que han demostrado ser eficaces en el desarrollo de sistemas de reconocimiento facial. Lenguajes como Python y C++ son frecuentemente mencionados debido a su flexibilidad y amplia gama de bibliotecas compatibles como OpenCV, TensorFlow y Keras. Según nuestro criterio es recomendable seleccionar Python como el lenguaje principal debido a sus extensas bibliotecas y la facilidad para prototipar y ajustar algoritmos de aprendizaje automático.

- El desarrollo de un sistema de control de acceso basado en reconocimiento facial para un salón de clases es viable y puede mejorar significativamente la precisión y eficiencia de la supervisión. También es importante mencionar, el implementar medidas de seguridad para proteger los datos biométricos y garantizar la privacidad de los estudiantes.
- Además, realizar pruebas exhaustivas en diferentes condiciones de iluminación y ángulos faciales puede asegurar la robustez del sistema. Según los resultados vistos en el sistema de reconocimiento facial, los modelos pre-entrandos de OpenCV, OpenFace y FaceNet requieren mayor robustez a la hora de la detección, dichos modelos pueden ser más precisos al implementar redes neuronales más complejas y estructuradas usando herramientas como TensorFlow.

10. Recomendaciones:

- **Sugerencias para futuras mejoras.**

Migrar a herramientas de software para diseño web, para luego hacer uso de electrón y exportar la aplicación de escritorio de manera óptima en su funcionamiento y en su desarrollo.

- **Posibles ampliaciones de la aplicación.**

Hacer el entorno del funcionamiento para dispositivos móviles

Cambio de paleta de colores más amigables para el usuario.

Mejora del modelo de reconocimiento facial.

11. Anexos:

11.1. Resultados detallados de las pruebas.

11.1.1. ***Pruebas Front-End***

11.1.1.1. ***Prueba de pantalla login #1***

Para realizar el login se siguió el diseño establecido en la etapa de planeación.

Ilustración 1 Modelo Inicial del Login de GenList

Tipo de Prueba	Nombre de la prueba	Fecha de Inicio	Fecha de Finalización	Resultados
Prueba de Interfaz de Usuario	Prueba pantalla login #1	22/04/2024, 9:40:01 pm	22/04/2024 9:42:01 pm	Los resultados esperados respecto a la ubicación de elementos fue éxito.

Resultado prueba de pantalla login #1:

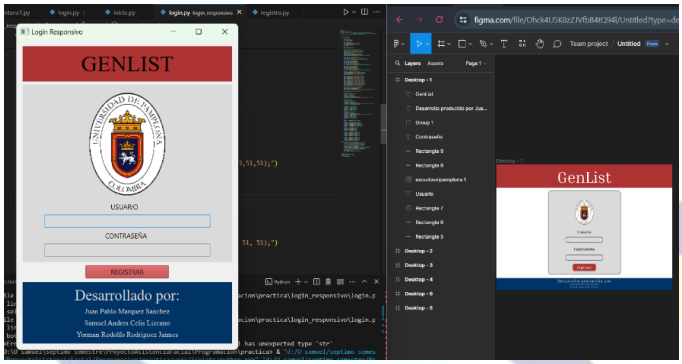


Ilustración 2 Resultado prueba de pantalla login #1

Realizada la primera prueba de Interfaz de Usuario se pudieron sacar conclusiones tales como:

- los colores no eran suficientemente amigables al usuario, no se lograba transmitir un espacio alegre e intuitivo.
- La aplicación necesitaba una identidad, el logo de la universidad no representaba esta idea.
- El tipo de fuente utilizado no era el más optimo.

Con estas conclusiones en mente se decidió realizar cambios en el diseño general de la aplicación estos cambios incluían: **Colores, Logo** y **tipo de fuente.**

*11.1.1.2. **Prueba de pantalla #2:***

Para realizar esta prueba se hicieron varios cambios al modelo, todo esto en base a los resultados obtenidos en la primera prueba, tales cambios fueron:

Ilustración 3 Propuesta cambio de Logo y colores

Primero se propuso cambiar los colores de tonalidad, el logo se cambió por el del grupo JUYSAM y el tipo de fuente se mantuvo, en la ilustración 3 podemos observar la comparación de los colores y el logo.

Por medio de reuniones se acordó probar otros colores los cuales los podemos apreciar en la ilustración 4 y 5.

Ilustración 4 Login con opción de Colores 1

Ilustración 5 Login con opción de Colores 2

Con estas opciones sobre la mesa se acordó la segunda prueba de interfaz de usuario para el login.

Tipo de Prueba	Nombre de la prueba	Fecha de Inicio	Fecha de Finalización	Resultados
Prueba de Interfaz de Usuario	Prueba pantalla login #2	24/04/2024, 11:33:55 am	24/04/2024 12:00:00 m	Los resultados de la prueba fueron positivos porque se lograron establecer los colores, el logo y tipo de fuente que se utilizaría para la aplicación.

Una vez terminada la prueba se llegó a un resultado final el cual no había sido planteado inicialmente, este resultado se puede observar en la ilustración 6.

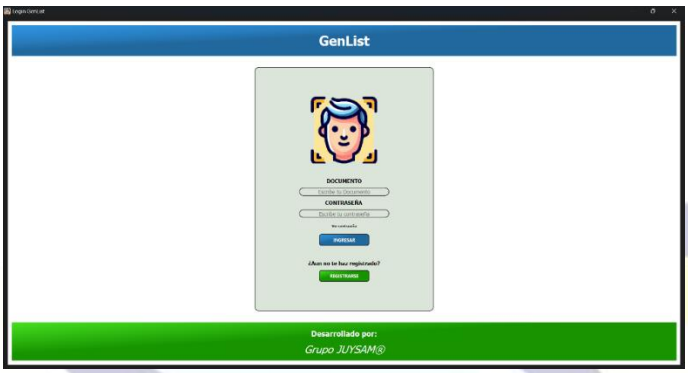


Ilustración 6 Login resultante de las pruebas

11.1.1.3. Prueba de la funcionalidad "Registrarse"

Para realizar la prueba unitaria de la funcionalidad "Registrarse" en la parte del Front-End, se realizó la navegación respectiva que debía de hacer el usuario para realizar el registro.

La primera ventana de la funcionalidad se puede apreciar en la ilustración 7, la cual contenía 3 campos a llenar por el usuario.

Ilustración 7 Funcionalidad "Registrarse"

Una vez terminada la prueba se pudo concluir con el equipo que se requeria mas campo de validaciones, por lo tanto, se agregó un espacio para confirmar la contraseña, esto fue probado en la segunda prueba de la funcionalidad.

La ventana resultante en base a los cambios basados en la prueba inicial es la observada en la ilustración 8.

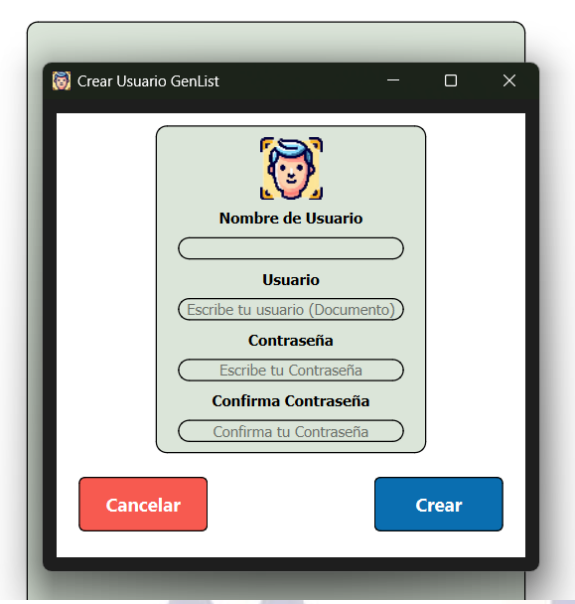


Ilustración 8 Ventana final de funcionalidad "Registrarse"

Tipo de Prueba	Nombre de la prueba	Fecha de Inicio	Fecha de Finalización	Resultados
Prueba de Unitaria	Funcionalidad “Registrarse”	21/10/2024, 2:00:00 am	23/10/2024 12:00:00 m	Se logro definir la interfaz final de la ventana de la funcionalidad “Registrarse”.

11.1.1.4. Prueba de Integración de la interfaz y el modelo de reconocimiento facial

Para realizar esta prueba se integró el código del modelo de reconocimiento facial a la interfaz de usuario, en la ilustración 9 podemos evidenciar la primera interacción que se realiza entre la interfaz de registro de estudiantes y el modelo de reconocimiento facial.

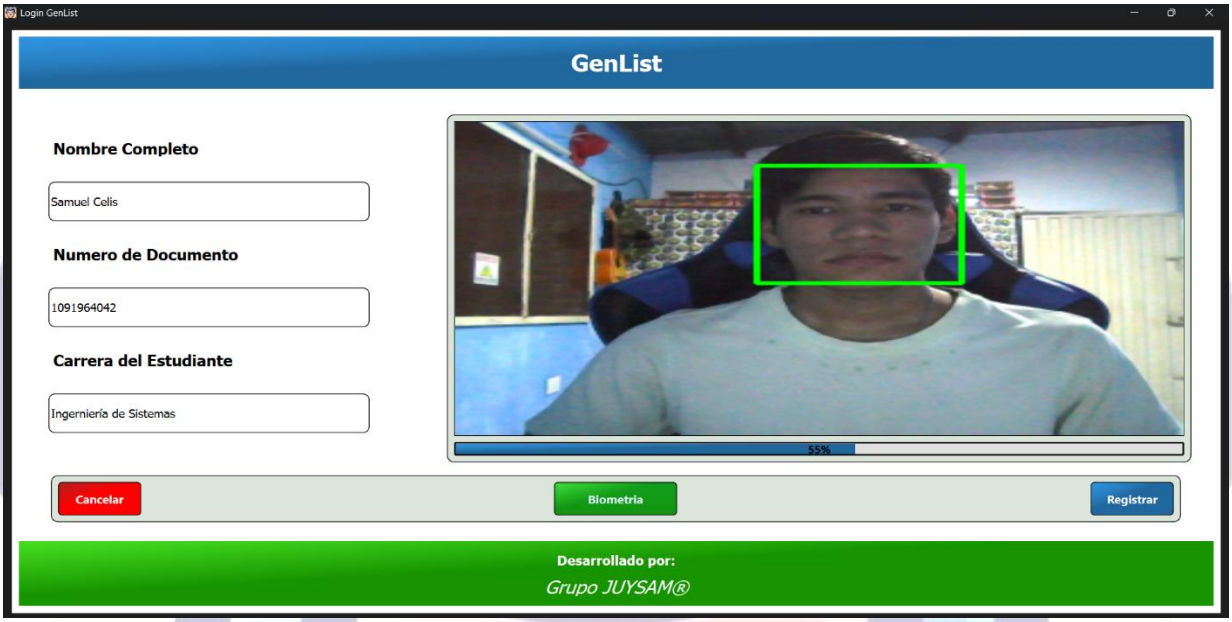


Ilustración 9 Prueba de Integración de Interfaz con Modelo de Reconocimiento

Como segunda prueba de integración se testeo la interfaz de toma de asistencia con la integración del modelo de reconocimiento facial, esto lo podemos observar en la ilustración 10.

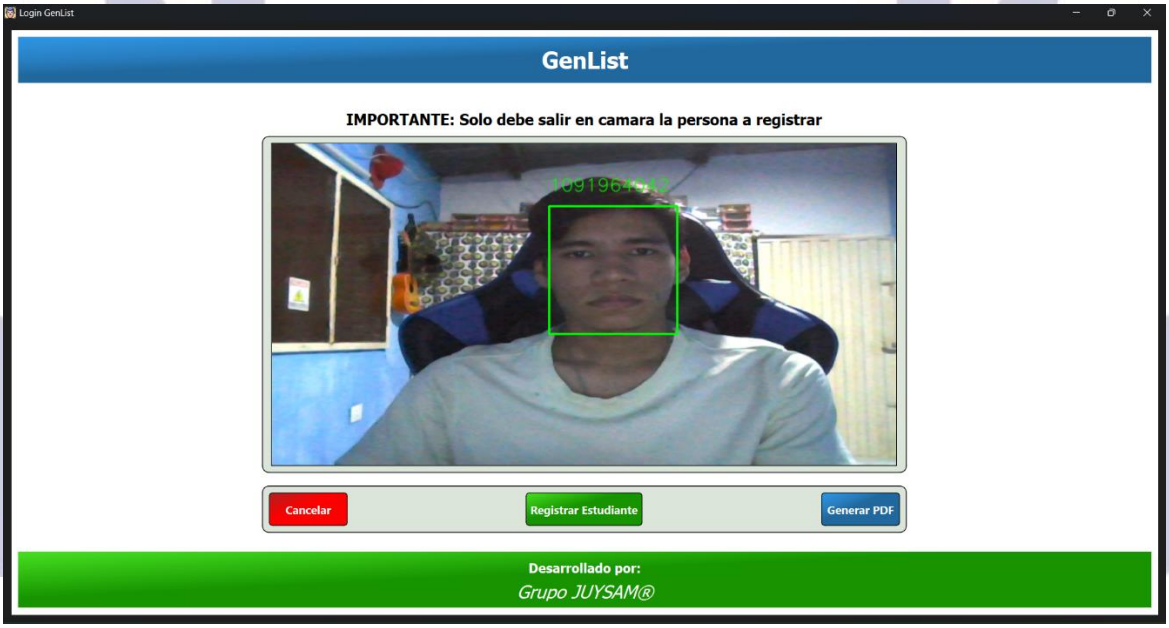


Ilustración 10 Prueba de Integración de Interfaz con Modelo de Reconocimiento II

Los resultados de estas pruebas fueron exitosos, en ambos casos la integración fue un éxito, el modelo con la interfaz trabajaron de forma adecuada cada uno realizando sus respectivas funciones.

Tipo de Prueba	Nombre de la prueba	Fecha de Inicio	Fecha de Finalización	Resultados
Prueba de Integración	Prueba de Integración de la interfaz y el modelo de reconocimiento facial	2/11/2024, 11:33:55 am	4/11/2024 12:00:00 m	La integración fue un éxito, los elementos trabajaron de la forma esperada.

11.1.1.5. Prueba de integración de la interfaz y la base de datos

En la realización de las pruebas de integración de la interfaz de usuario con la base de datos, se verifico la validez de la información mostrada a los usuarios después de ingresar a la aplicación y encontrarse en la parte de inicio. Esto lo podemos observar en la ilustración 11.

Ilustración 11 Prueba de integración Interfaz con Base de Datos

En la prueba de integración realizada los resultados fueron positivos, la información cargaba en los sitios solicitados y era acorde con los usuarios.

Tipo de Prueba	Nombre de la prueba	Fecha de Inicio	Fecha de Finalización	Resultados
Prueba de Integración	Prueba de integración de la interfaz y la base de datos	6/11/2024, 2:00:00 am	8/11/2024 12:00:00 m	Los resultados de la prueba fueron exitosos, la información es concordante con los usuarios.

11.1.2. Pruebas de Back-End

11.1.2.1. Prueba Unitaria Validar Credenciales

En esta prueba lo que se busca validar la funcionalidad del proceso de la validación de credenciales de profesores, para hacer esto, nos conectamos a la base de datos, enviamos los datos preestablecidos a la consulta a la base de datos, donde tenemos unos resultados esperados tanto en caso de ser una validación correcta para los datos de las credenciales del profesor y en su respectivo caso contrario:

```
8     @patch('src.components.conexcionBD.sqlite3.connect')
9     def test_validar_credenciales_profesor(self, mock_connect):
10        mock_conexion = MagicMock()
11        mock_cursor = MagicMock()
12        mock_connect.return_value = mock_conexion
13        mock_conexion.cursor.return_value = mock_cursor
14
15        # Simulamos que existe el documento del profesor y que la contraseña es válida
16        mock_cursor.fetchone.side_effect = [(1,), (1,)]
17
18        resultado = conexcionBD.validar_credenciales_profesor(
19            '123456', 'password123')
20        print(resultado)
21        self.assertTrue(resultado)
22
23        # Simulamos que no existe el documento del profesor
24        mock_cursor.fetchone.side_effect = [(0,), (0,)]
25        resultado = conexcionBD.validar_credenciales_profesor(
26            '654321', 'password123')
27        print(resultado)
28        self.assertFalse(resultado)
29    You, hace 3 meses • Arreglo de crasheo de aplicación y se agrego un…

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS  GITLENS

st/venv/Scripts/python.exe" "d:/U_samuel/septimo semestre/ProyectoAsistenciaFacial/Programacion/Gen
..True
False
.
```

Tipo de Prueba	Nombre de la prueba	Fecha de Inicio	Fecha de Finalización	Resultados
Prueba Unitaria	Prueba Unitaria Validar Credenciales	10/11/2024, 1:30:00 pm	13/11/2024 6:00:00 pm	Los resultados fueron exitosos ya que se verifico el optimo funcionamiento de las consultas

11.1.2.2. Prueba Unitaria Obtener Nombre de Profesor

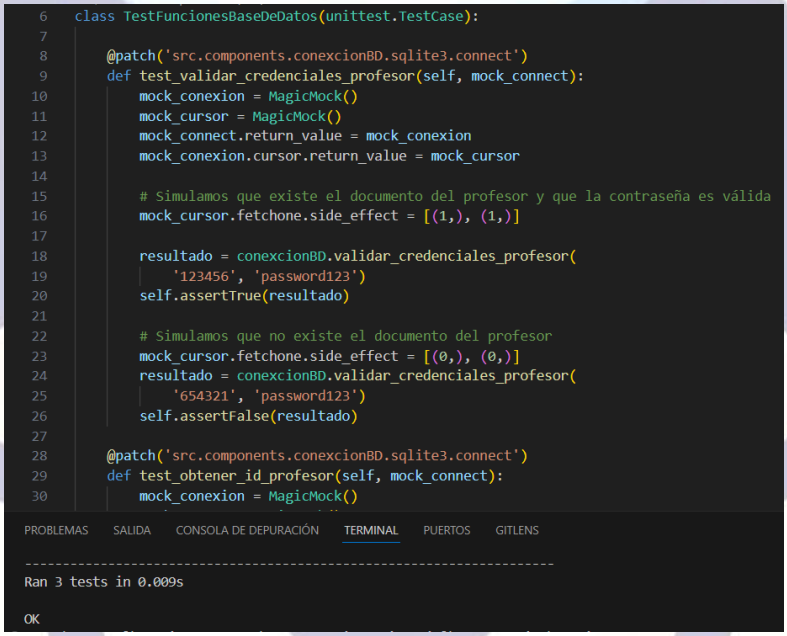
En esta prueba lo que se busca validar la funcionalidad del proceso de validación de existencia del nombre de un profesor si existe en la base de datos, para hacer esto, nos conectamos a la base de datos, enviamos los datos preestablecidos a la consulta a la base de datos, donde hacemos una consulta exitosa con un nombre existente en la base de y en su respectivo caso contrario de no existe:

```python
@patch('src.components.conexcionBD.sqlite3.connect')
    def test_obtener_nombre_profesor(self, mock_connect):
        mock_conexion = MagicMock()
        mock_cursor = MagicMock()
        mock_connect.return_value = mock_conexion
        mock_conexion.cursor.return_value = mock_cursor

        # Simulamos que se encuentra el nombre del profesor
        mock_cursor.fetchone.return_value = ('Juan Pérez',)
        resultado = conexcionBD.obtener_nombre_profesor(10)
        self.assertEqual(resultado, 'Juan Pérez')

        # Simulamos que no se encuentra el nombre del profesor
        mock_cursor.fetchone.return_value = None
        resultado = conexcionBD.obtener_nombre_profesor(20)
        self.assertIsNone(resultado)

```

Tipo de Prueba	Nombre de la prueba	Fecha de Inicio	Fecha de Finalización	Resultados
Prueba Unitaria	Prueba de la carga de imágenes en el modelo	14/11/2024, 2:30:00 pm	15/11/2024 11:25:30 am	Los resultados fueron exitosos ya que esta prueba permitió corregir errores de consultas

11.1.2.3. Prueba Unitaria Comprobación de Funciones

En esta prueba lo que se busca validar el óptimo funcionamiento de las funciones, para hacer esto, nos conectamos a la base de datos, ejecutamos en una función donde se ejecutan demás funciones ya preestablecidas para monitorear que no se generen errores al ejecutarse varias consultas a la base de datos, en este enviamos los posibles resultados de una función, ya sean tanto verdadero como falso el resultado:



Tipo de Prueba	Nombre de la prueba	Fecha de Inicio	Fecha de Finalización	Resultados
Prueba Unitaria	Prueba de la carga de imágenes en el modelo	24/11/2024, 10:00:00 am	27/11/2024 6:00:00 pm	Los resultados fueron exitosos ya que esta prueba permitió implementación de varias funciones

11.1.3. Pruebas del modelo predictivo

11.1.3.1. Prueba del modelo en la identificación

En esta prueba lo que se busca verificar es la precisión del modelo en la identificación de rostros, para hacer esto, se empezó cargándole unas 1000 fotos de cada persona para que aprenda sus rasgos faciales y así determinar que es la persona en la cámara:

En la ilustración # se puede observar la precisión del modelo identificando a el desarrollador Juan Marquez y cualquiera que no sea él, lo marcara como desconocido:

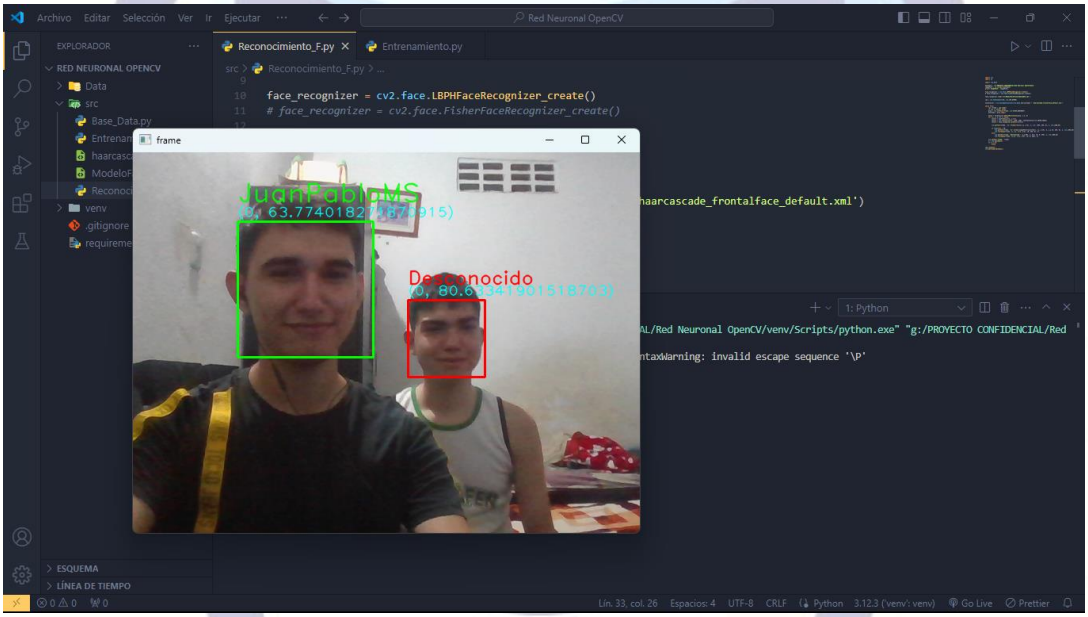


Ilustración 12, captura del modelo en la identificación de rostros

Se tomaron datos como el porcentaje de coincidencia de la persona para luego configurar la variable de aceptación:

Tipo de Prueba	Nombre de la prueba	Fecha de Inicio	Fecha de Finalización	Resultados
Prueba de la precisión del modelo	Prueba del modelo en la identificación	30/10/2024, 10:00:00 am	05/11/2024 12:00:00 pm	Los resultados al principio fueron variando, pero sirvieron para la configuración del modelo

11.1.3.2. Prueba de la carga de imágenes en el modelo

En esta prueba se realizaron procesos de optimización en la carga de imágenes, ya que al principio se necesitaban cargar 1000 imágenes por persona para que la identificara. Pero poco a poco ese numero se fue ajustando debido a que la precisión del modelo no bajaba mucho cuando se reducía el número de imágenes por persona. Logrando solo necesitar 300 fotos por persona, haciendo que el sistema fuera más eficiente.

En la ilustración # se puede observar dos personas identificadas por el modelo y en consola la variable de coincidencia:

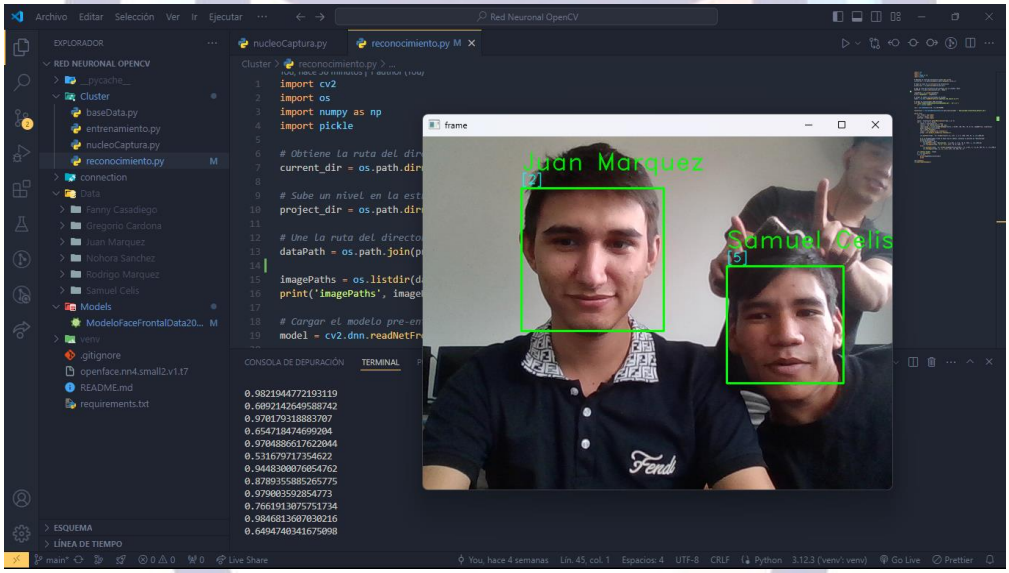


Ilustración 13, Captura de la carga de imágenes

Tipo de Prueba	Nombre de la prueba	Fecha de Inicio	Fecha de Finalización	Resultados
Prueba de la carga del modelo	Prueba de la carga de imágenes en el modelo	06/11/2024, 1:30:00 pm	08/11/2024 6:00:00 pm	Los resultados fueron exitosos ya que esta prueba permitió hacer mas eficiente el modelo

11.1.3.3. Prueba de registro de persona

En esta prueba se realizo un seguimiento al modelo a la hora de preprocesar las imágenes y generar los modelos entrenados:

En la ilustración # se puede ver como la aplicación registra a una persona para generar un modelo entrenado:

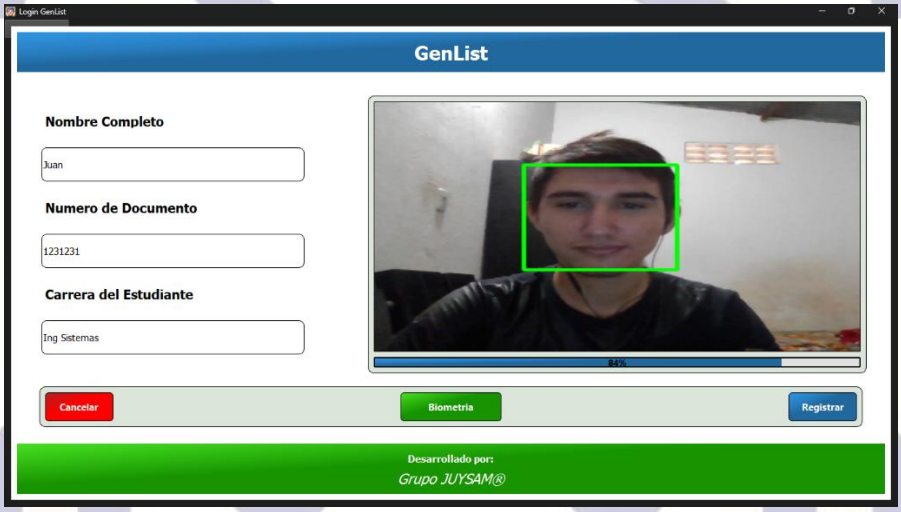


Ilustración 14, Captura del Registro de una persona

En la ilustración # se puede ver como la aplicación con el modelo predictivo entrenado identifica a la persona que recientemente registro

Ilustración 15, Sistema identificando una persona registrada

GenList

Tipo de Prueba	Nombre de la prueba	Fecha de Inicio	Fecha de Finalización	Resultados
Prueba del preprocesamiento de imágenes	Prueba de registro de persona	09/11/2024, 8:00:00 am	11/11/2024 5:00:00 pm	Los resultados fueron exitosos ya que el modelo logro identificar a las personas registradas