Taller GUI/Relaciones/Componentes/ Capas

Emmanuel Velez Lopez
Docente: Alejandro Rodas Vásquez
Universidad Tecnológica de Pereira

14 de mayo de 2024

Introducción

Una de las claves en este ejercicio es tener claridad los conceptos Arquitectura de Software, donde la aplicación del principio de *Modularidad* es fundamental. En este taller se desarrollará una arquitectura de Tres Capas. Sin embargo, esta arquitectura todavía presenta *un alto grado de acoplamiento* por no utilizar aun el concepto de interfaz.

1. Arquitectura de Software

Usted ha sido contratado para implementar un *Sistema de Información Hospitalaria* que se encuentra en su fase inicial. De modo que hasta el momento solo se utilizarán las tablas, *Hospital* y *Doctor*.

Desde una perspectiva de *Programación Orientada a Objetos* el *modelo* (Figura 1) se puede representar de la siguiente manera:

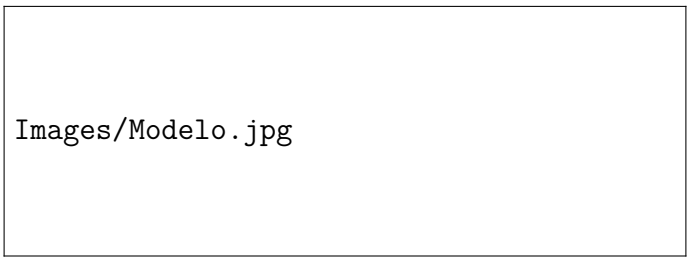


Figura 1: Modelo Sistema de Información Hospitalaria.

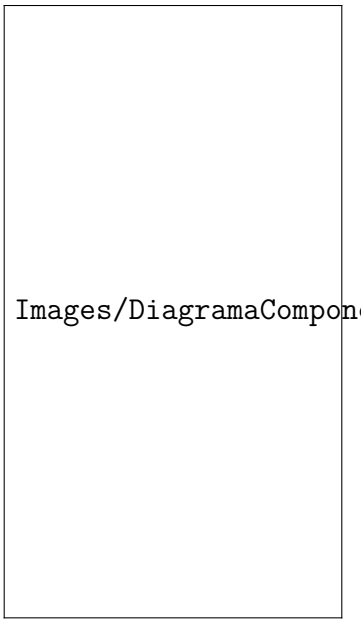


Figura 2: Arquitectura del Sistema de Información Hospitalaria.

2. Requerimientos Funcionales

Esta aplicación debe de:

- Usted debe de implementar el *Sistema de Información Hospitalaria* siguiendo la Arquitectura previamente expuesta (Sección 1).
- Seguir el *Estilo Arquitectónico en Capas* presentado en la Figura 2.
- Ser desarrollada en Python (aplicación de Escritorio) utilizando los conceptos de *Módulos y NameSpace*.
- Crear y almacenar un *Hospital* con su correspondiente planta de *Doctores*
- Implementar la función *search_by_dni()* para obtener información del médico y del hospital utilizando la identificación del médico. Esta información debe de ser mostrada en *tabla (grid)* en la interfaz gráfica.

2.1. Actividad de Investigación

¿Qué son los *Diagramas de Casos de Uso*?

2.2. Solución

- Los diagramas de casos de uso son una herramienta de modelado utilizada en el desarrollo de software para representar las interacciones entre un sistema y sus actores externos. Los casos de uso describen cómo los usuarios interactúan con el sistema para lograr ciertos objetivos. Estos diagramas son parte de la especificación de requisitos y ayudan a comprender los requisitos funcionales del sistema desde la perspectiva del usuario. En un diagrama de casos de uso, se representan los actores externos al sistema y los diferentes casos de uso que describen las acciones que los actores pueden realizar en el sistema. Los casos de uso se representan como elipses y los actores como figuras externas al sistema, como círculos o rectángulos. Las relaciones entre los actores y los casos de uso se muestran mediante líneas.

¿Para qué se usan?

2.3. Solución

- Los diagramas de casos de uso son útiles para:

1. Identificar los requisitos del sistema desde la perspectiva del usuario.
2. Visualizar las interacciones entre el sistema y los actores externos.
3. Ayudar en la comunicación entre los stakeholders del proyecto.

4. Servir como base para el diseño de la arquitectura del sistema.

   En resumen, los diagramas de casos de uso son una herramienta valiosa para comprender y especificar los requisitos funcionales de un sistema de software de una manera clara y comprensible para todos los involucrados en el proyecto.

3. ¿Cómo realizo la entrega?

a) Usted debe de entregar el código fuente (archivo comprimido).

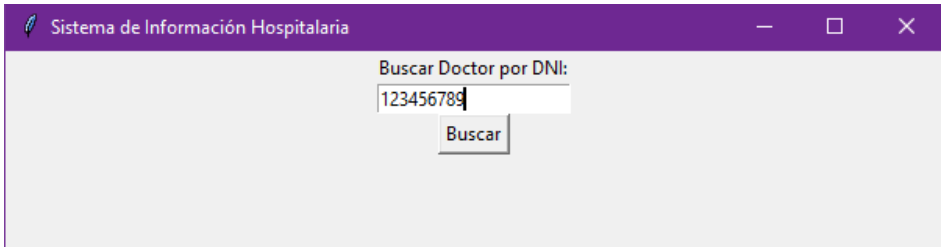
b) Pantallazos donde se corrobore el funcionamiento del sofware (consultas realizadas y resultado obtenido) y se compruebe que los Requerimientos Funcionales (Sección 2) han sido cumplidos.

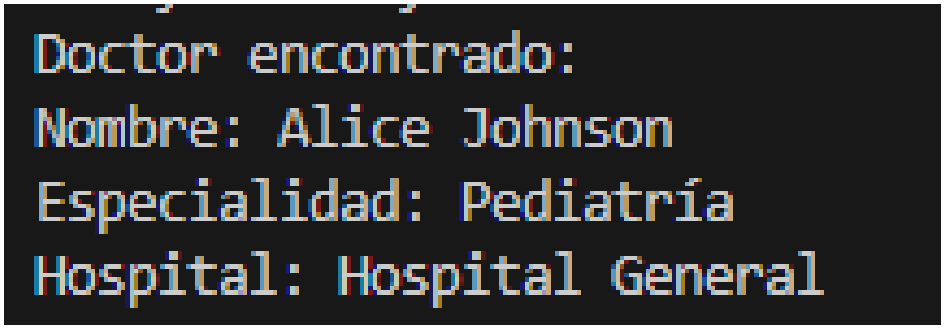
4. Evidencias

4.1. Repositorio de GitHub

PROGRMA MODELO

4.2. Interfaz Gráfica (GUI)





4.3. Componentes

hospital.py

```python
# PARCIAL_FINAL/model/hospital.py

class Hospital:
    def __init__(self, hospital_name):
        self.hospital_name = hospital_name
        self.doctors = []

    def add_doctor(self, doctor):
        self.doctors.append(doctor)
```

doctor.py

```python
# PARCIAL_FINAL/model/doctor.py

class Doctor:
    def __init__(self, doctor_name, speciality, dni):
        self.doctor_name = doctor_name
        self.speciality = speciality
        self.dni = dni
```

controller.py

```python
# PARCIAL_FINAL/controlador/controller.py

import sys
sys.path.append("PROGRAMACION\\PARCIAL_FINAL\\model")

print(sys.path)

from model.doctor import Doctor
from model.hospital import Hospital

class HospitalController:
    def __init__(self):
        self.hospital = None

    def create_hospital(self, hospital_name):
        self.hospital = Hospital(hospital_name)

    def add_doctor(self, doctor_name, speciality, dni):
        if self.hospital:
            doctor = Doctor(doctor_name, speciality, dni)
            self.hospital.add_doctor(doctor)
        else:
            print("Error: Debe crear un hospital primero.")

```

```
25      def search_by_dni(self, dni):
26          if self.hospital:
27              for doctor in self.hospital.doctors:
28                  if doctor.dni == dni:
29                      return doctor, self.hospital
30              return None, None
31          else:
32              print("Error: Debe crear un hospital primero.")
```