Arquitectura del Sistema

Se debe añadir al documento de requerimientos creado en el trabajo práctico anterior una sección que incluya un **diagrama de arquitectura preliminar** que permita asociar requerimientos y casos de usos con los sistemas, subsistemas y módulos identificados.

Por otro lado, se debe generar otro documento de **arquitectura** donde se presente:

- Un gráfico de arquitectura general para mostrar los componentes y sus relaciones con las interfaces externas.
- La explicación del **patrón de arquitectura** que fue usado y por qué, haciendo énfasis en cómo resuelve los requerimientos no funcionales.
- Diagramas UML de despliegue y de componentes.
- Opcionalmente se puede incluir un diagrama de contexto que incluya la relación de los sistemas y los subsistemas con las actividades del dominio del conocimiento de la aplicación.

Se deben definir los casos de **prueba de integración** que verifican la correcta interacción entre todos los componentes del sistema.

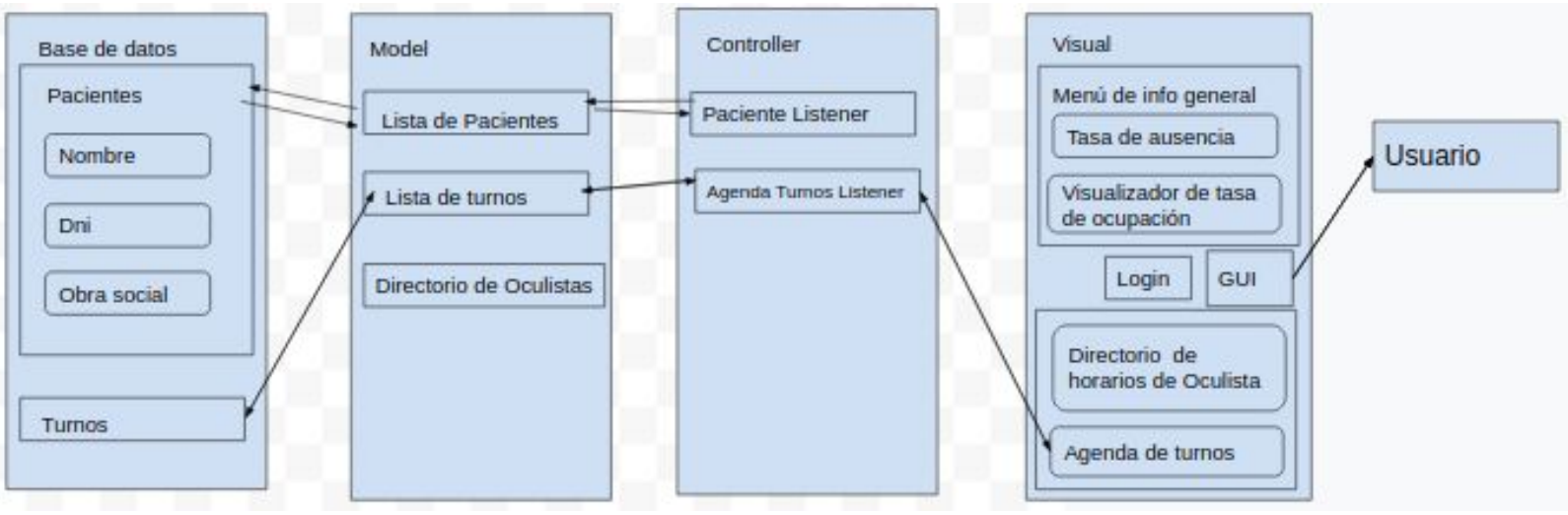
Arquitectura y Diseño del Sistema

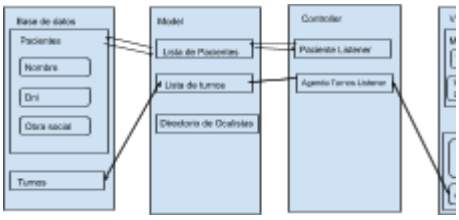
Introducción:

En el siguiente informe detallaremos el diseño e implementación de nuestro proyecto, así como también las pruebas realizadas sobre el sistema. Dentro del informe podremos ver dos partes principales, la relacionada a la Arquitectura del Sistema y la del Diseño del Sistema.

Desarrollo:

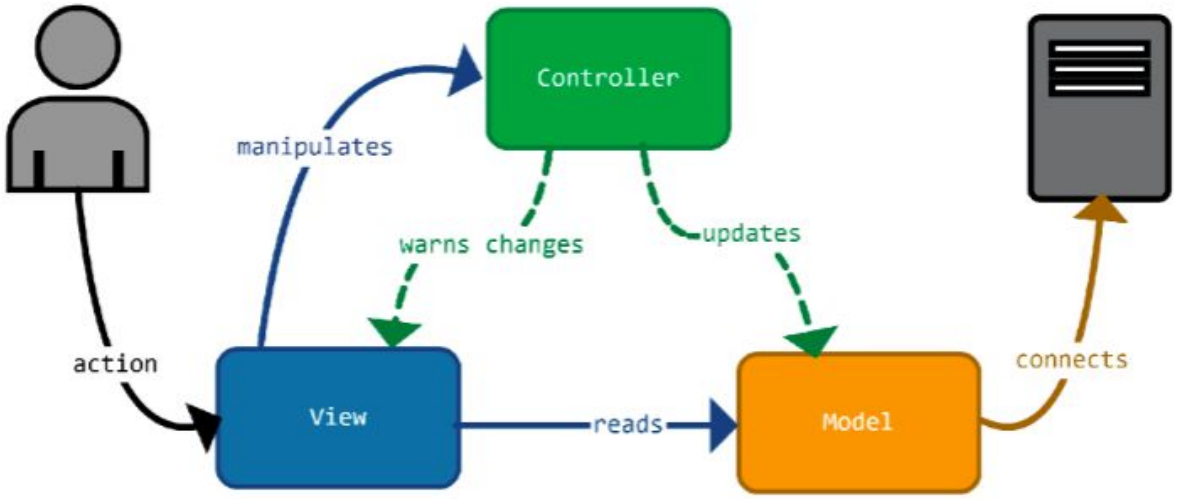
Diagrama de Arquitectura General:





(SI NO LES GUSTA ALGO DEL GRAFICO DE ARRIBA APRENTEN ACA PARA MODIFICARLO)

Arquitectura



En nuestro proyecto utilizamos como **patrón de arquitectura** el MVC (Model View Controller).
Éste consiste en tres partes, cada una con funciones específicas las cuales se interrelacionan para permitir el correcto funcionamiento del sistema.

El patrón elegido resuelve eficazmente los requerimientos no funcionales previstos en el planeamiento, ya que nos provee la posibilidad de tener una interfaz visual amigable con la persona que haga uso del programa, ésta es veloz e intuitiva y muestra algunos controles básicos para cumplir con los requerimientos del usuario.

Luego, haciendo foco en el modelo, tenemos todo lo relacionado a información y operaciones internas del programa; éste nos permite satisfacer el requerimiento de tener una base de datos ilimitada, ya que todos los datos e información son almacenados en esta parte. La base de datos puede ser configurada para que se almacene localmente o a través de internet, lo cual lo hace un sistema muy versátil. Al ser programado en Java las instrucciones son sencillas y muchas funcionalidades han sido simplificadas con el uso de librerías, así como también realizamos buenas prácticas de código aprendidas en la materia. Todo esto nos da como resultado un programa simple y rápido que está disponible cuando el usuario lo requiera; y permite al programador modificarlo y actualizarlo fácilmente debido a las prácticas implementadas.
Como último componente tenemos el controlador, encargado de interrelacionar el modelo y la visual, se encarga de generar una respuesta cuando el usuario interactúa con la interfaz. Nos provee un método ágil y más que suficiente para nuestro proyecto.

Diagramas UML de despliegue y de componentes:

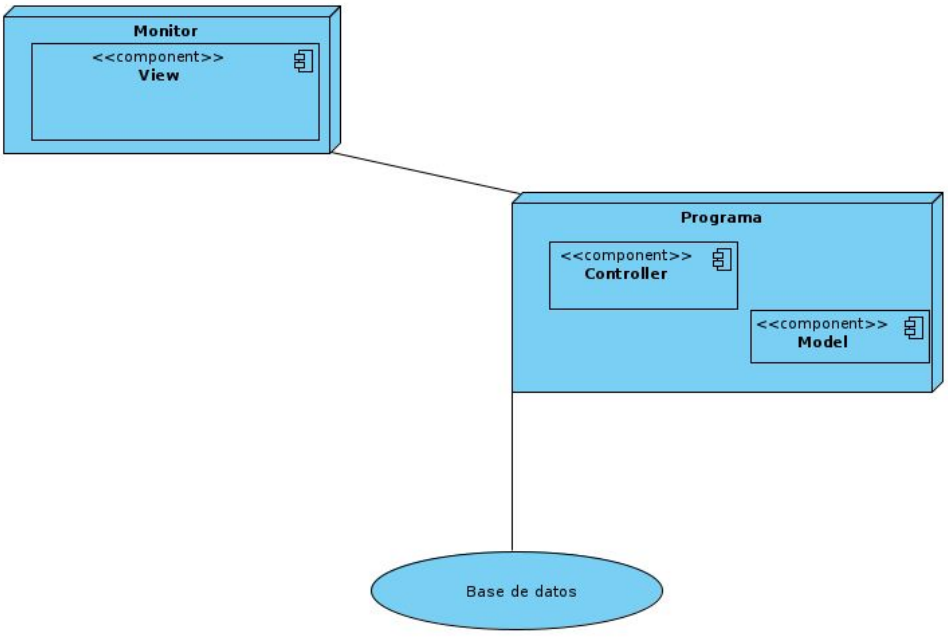


Diagrama de despliegue

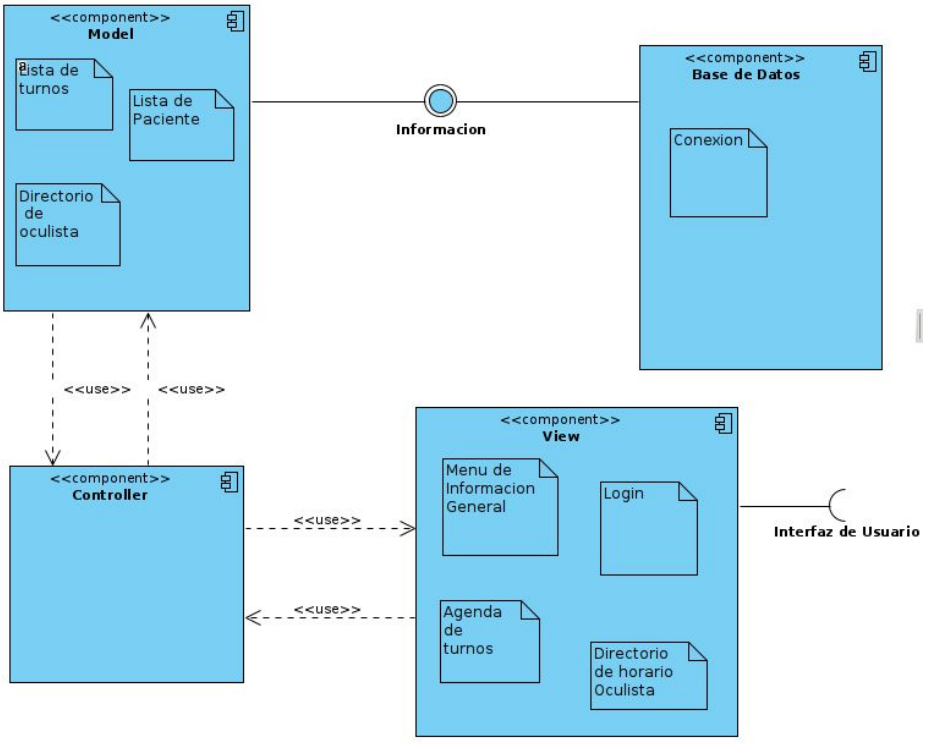


Diagrama de componentes

Base De Datos: Será el componente que se encargará de la gestión y almacenamiento de los datos necesarios en el sistema del consultorio. Este consta de datos de pacientes y de turnos, cada uno se almacenado de forma independiente.

View: Será un componente a través de la cual se crearán todas las vistas del programa y mediante el cual se le presentará una Interfaz Gráfica al usuario, quien interactuara con nuestra aplicación. La misma detectará sus acciones, a través de ActionListeners los cuales notificarán al controller de las medidas a tomar.

Model: Será el componente encargado en reunir toda la lógica y manejar toda la data del sistema.

Controller: Será el componente encargado de comportarse como intermediario entre el View y el Model. Responsable de la actualización de ambos, dispondrá de todos los ActionListeners utilizados en las vistas.

Casos de Prueba de Integración

●Caso I: Verificar integración entre Turno y el agregar un turno, comprobando la correcta actualización de la lista de turnos del usuario.

Prerrequisitos: agenda.existeTurno()==False que no haya un turno asignado.

Entrada: agenda.agregarTurno()

Salida esperada:En el caso de que el pre requisito se cumpla la salida esperada sería un agregarTurno exitoso, en caso contrario debería salta una excepcion diciendonos que ya hay un turno asignado en ese horario.

●Caso II: Verificar integración entre Turno y el borrar un turno, comprobando la correcta actualización de la lista de turnos del usuario.

Prerrequisitos: agenda.existeTurno()==True que haya un turno asignado.

Entrada: agenda.quitarTurno()

Salida esperada:En el caso de que el pre requisito se cumpla la salida esperada sería una entrada exitosa, en caso contrario debería salta una excepcion diciendonos que no hay un turno asignado en ese horario.

●Caso III: Verificar integración entre Turno y el cambiar turno, comprobando la

correcta actualización de la lista de turnos del usuario.

Prerrequisitos: agenda.existeTurno()==True que haya un turno asignado.
agendar.existeTurno()==False en el horario al que se va a cambiar.

Entrada: agenda.cambiarTurno()

Salida esperada:En el caso de que el pre requisito se cumpla la salida esperada sería una cambio exitoso, en caso contrario debería salta una excepcion diciendonos que no hay un turno asignado en ese horario.

●Caso IV: Verificar integración entre Paciente y Turno, comprobando que los datos expuestos por el Turno.comprobarPaciente() sean los correctos en un turno dado.

Prerrequisitos: agenda.existeTurno()==True que haya un turno asignado.

Entrada: Turno.comprobarPaciente()

Salida esperada:En el caso de que el pre requisito se cumpla la salida esperada sería una correlación de los datos del paciente.