Documento de Arquitectura

- Control de Versiones
    - Version 0.1.0 - Autores: Cavanagh Juan, Casabella Martin, Kleiner Matías - Fecha: 25/06/2017 - Descripción: primera versión del documento de arquitectura

1. Introducción

1.1 Propósito

El Documento de Arquitectura de Software presenta la arquitectura del sistema BITicket a través de diferentes vistas, cada una de las cuales ilustra un aspecto en particular del software desarrollado. Se pretende de esta forma que el documento proporcione al lector una visión global y comprensible del diseño general del software desarrollado. Proporciona una descripción de alto nivel de los objetivos de la arquitectura, los distintos casos de uso del Sistema y los estilos arquitectónicos. La estructura de este documento se basa en el modelo de vista de la arquitectura “4+1” de kruchten, el cual se describe en secciones posteriores.

1.2 Alcance

El alcance de este documento incluye al sistema en su totalidad. Esto es, la descripción de sus interfaces gráficas, bases de datos, así también como de las interfaces que permitan la comunicación entre los distintos componentes que forman parte de este proyecto, es decir aspectos de diseño que se consideran arquitectónicamente significativos y fundamentales.

1.3 Definiciones, Acrónimos y Abreviaciones

UML : Lenguaje de modelado unificado.
CSV : comma-separated values.
MVC : Patrón de arquitectura Model-View-Controller.

1.4 Objetivos y limitaciones

Se optara por diseñar el sistema según el presente documento para cumplir los siguientes objetivos:

- ❏ Poder cumplir con lo requerido en la materia Ingenieria en Software.
- ❏ Poder hacer un sistema modulable con el fin de, en un futuro, poder integrar las clases, librerías, métodos, etc, en algún otro sistema.
- ❏ Organizar y estructurar el diseño del sistema con el fin de que permita la correcta integración de los diferentes desarrollos que realiza el equipo.

Limitaciones

- ❏ Servidor: La base de datos será local. No se necesitará de algún protocolo de comunicación por internet.
- ❏ Almacenamiento de datos: Se utilizará una base de datos local (ficheros .csv).
- ❏ Rendimiento: El sistema debe responder a la base de datos en un tiempo pequeño ya que no requiere de conexión a internet.

2. Representación arquitectónica.

2.1 Vistas 4+1

Así como la construcción de una casa necesita diferentes perspectivas, un sistema de software también lo requiere.

Logramos de esta forma cierta solidez en el diseño, dejando fuera las ambigüedades.

El diseño del software no es responsabilidad de un solo desarrollador sino que es un acuerdo entre todos ellos y se basan en el presente documento para la realización del sistema.
Se opta utilizar UML como lenguaje de modelado.

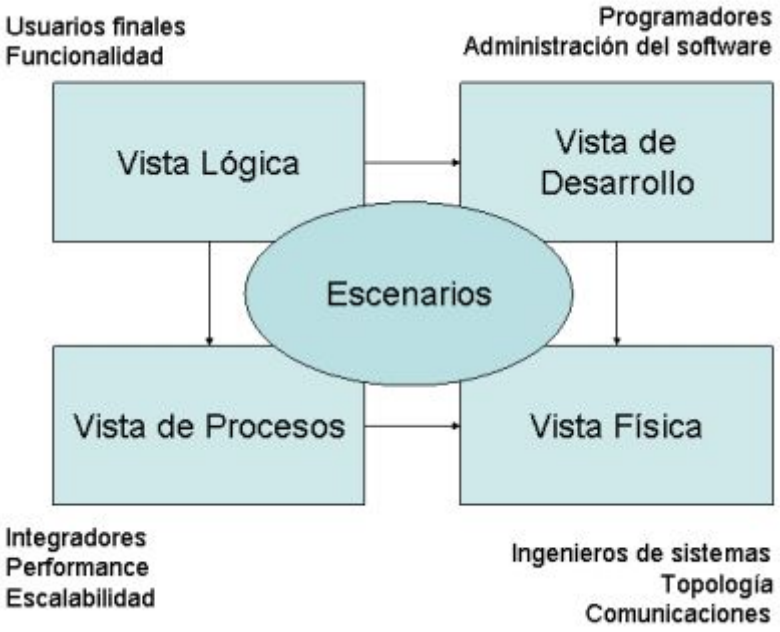


Diagrama básico de la vista 4+1

Vista Lógica: En esta vista se representa la funcionalidad que el sistema proporcionará a los usuarios finales. Es decir, se ha de representar lo que el sistema debe hacer, y las funciones y servicios que ofrece. Para completar la documentación de esta vista se pueden incluir los diagramas de clases, de comunicación o de secuencia de UML.

Para representar la perspectiva de esta vista se eligió el diagrama de clases, presentado en la siguiente figura.

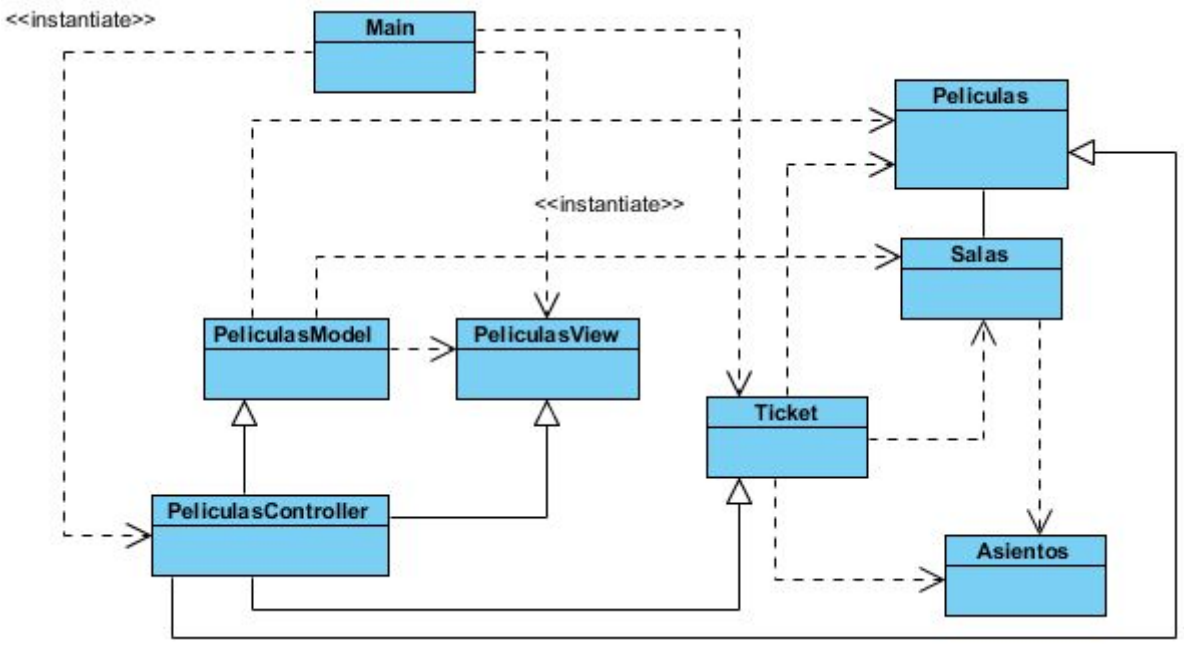
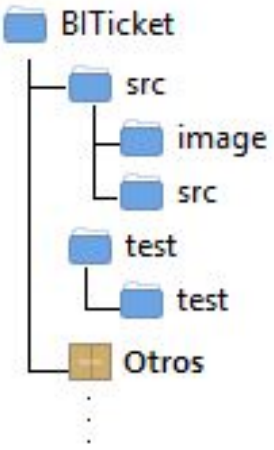


Diagrama de clase de la aplicación BITicket.



Esquema de directorios para el desarrollo

Vista de Desarrollo: En esta vista se muestra el sistema desde la perspectiva de un programador y se ocupa de la gestión del software; o en otras palabras, se va a mostrar cómo está dividido el sistema software en componentes y las dependencias que hay entre esos componentes. Para completar la documentación de esta vista se pueden incluir los diagramas de componentes y de paquetes de UML.

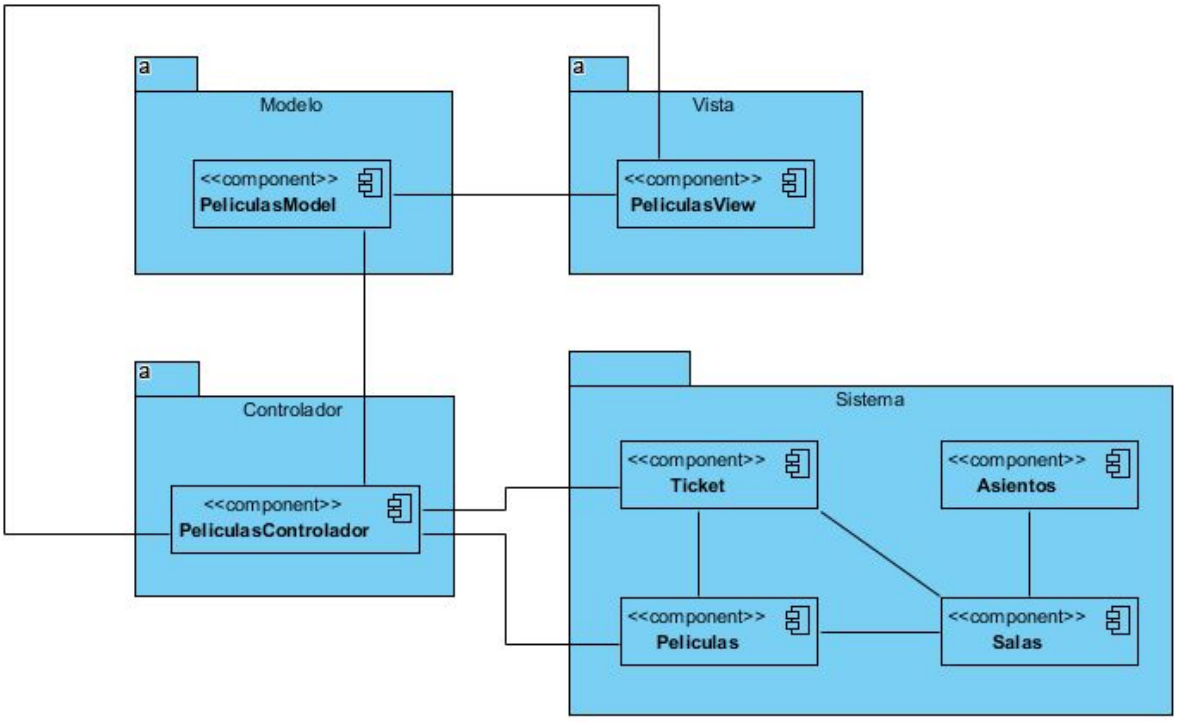
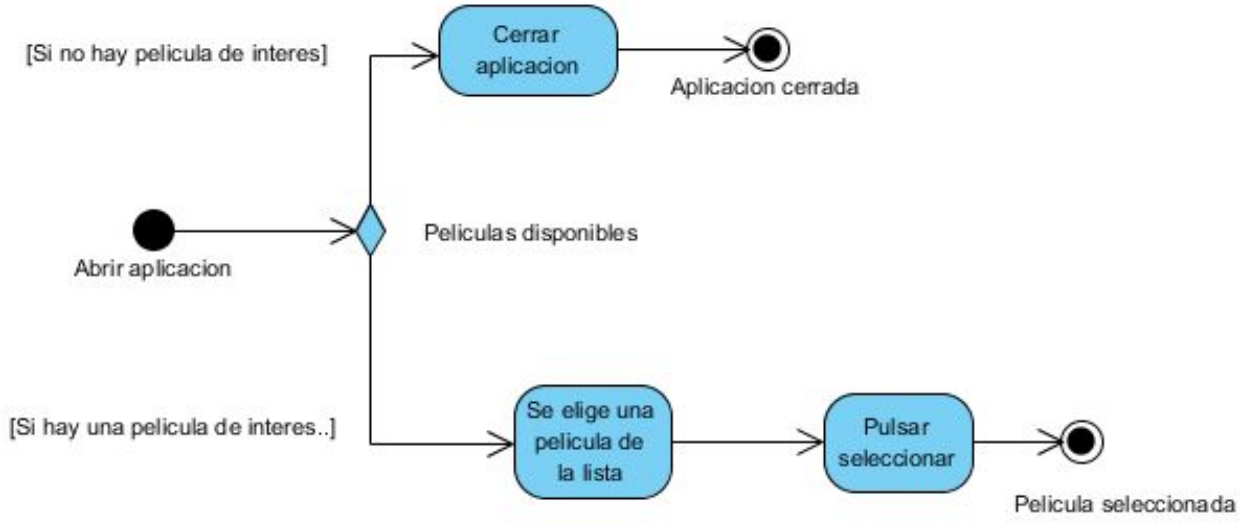
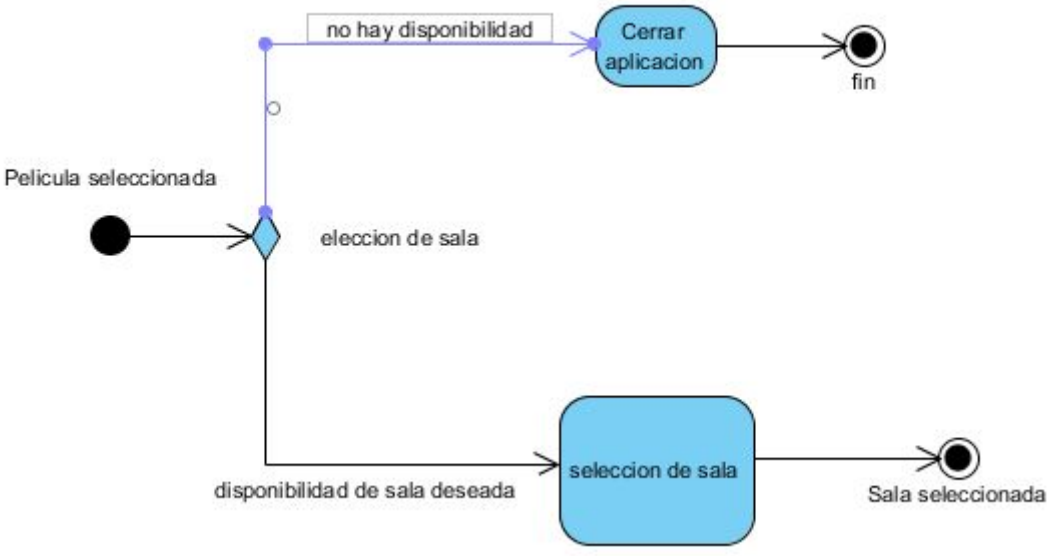


Diagrama de componentes UML para el sistema BITicket

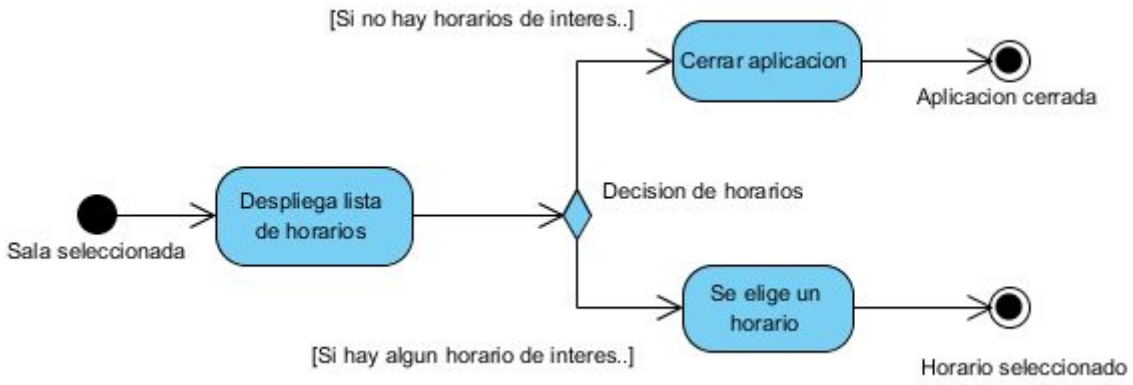
Vista de Procesos: En esta vista se muestran los procesos que hay en el sistema y la forma en la que se comunican estos procesos; es decir, se representa desde la perspectiva de un integrador de sistemas, el flujo de trabajo paso a paso de negocio y operacionales de los componentes que conforman el sistema. Para completar la documentación de esta vista se puede incluir el diagrama de actividad de UML.



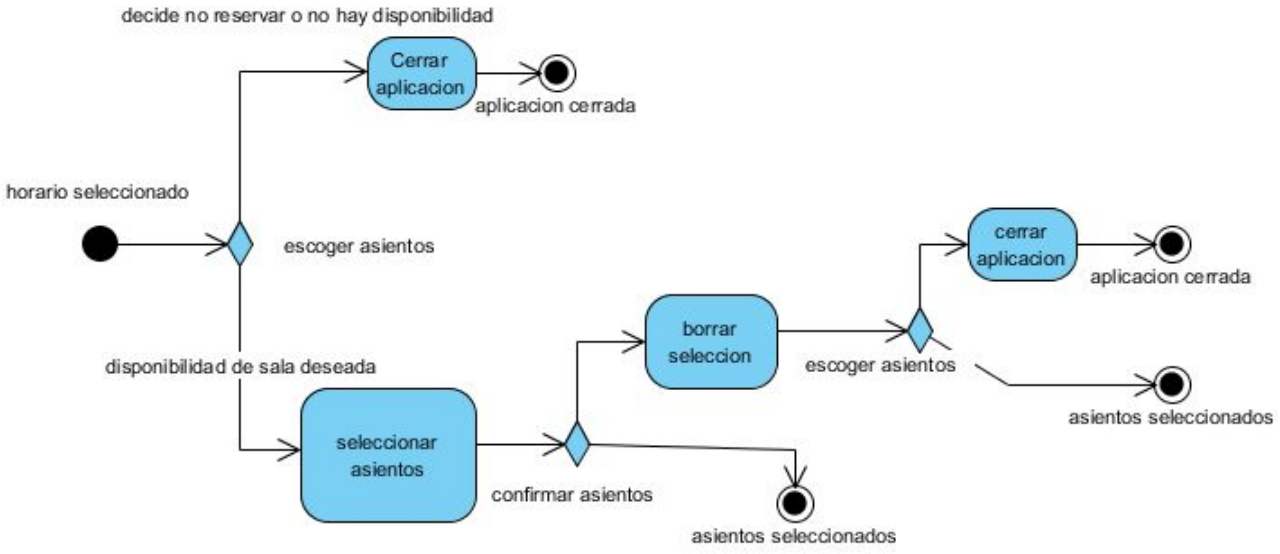
Selección de película



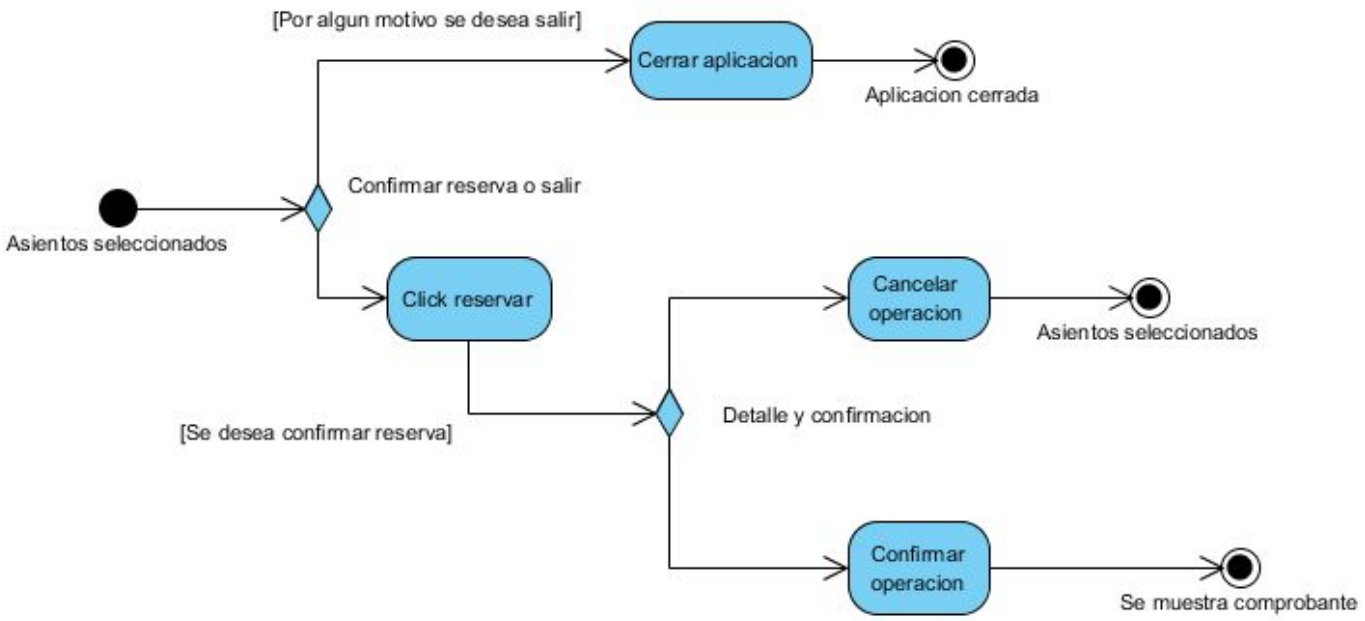
Selección de sala



Selección de horario

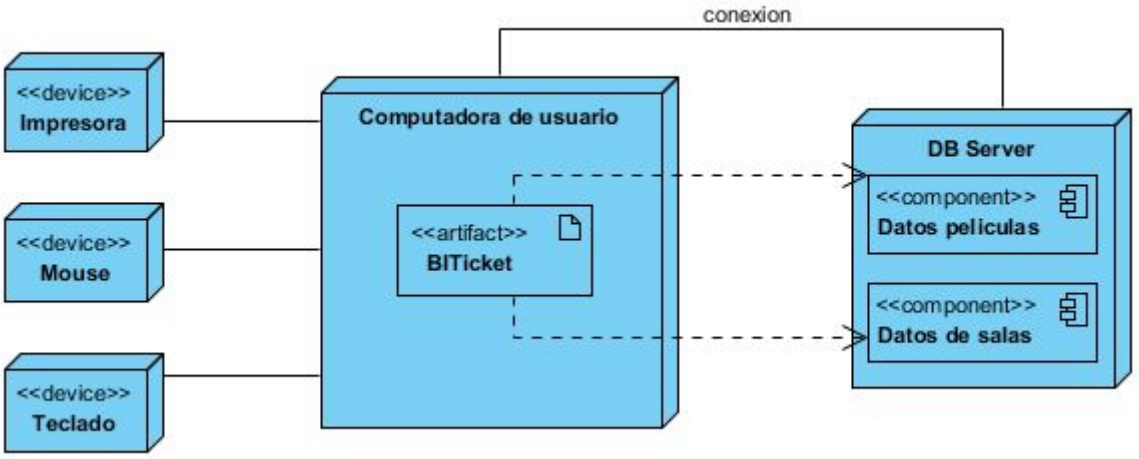


Selección de asientos

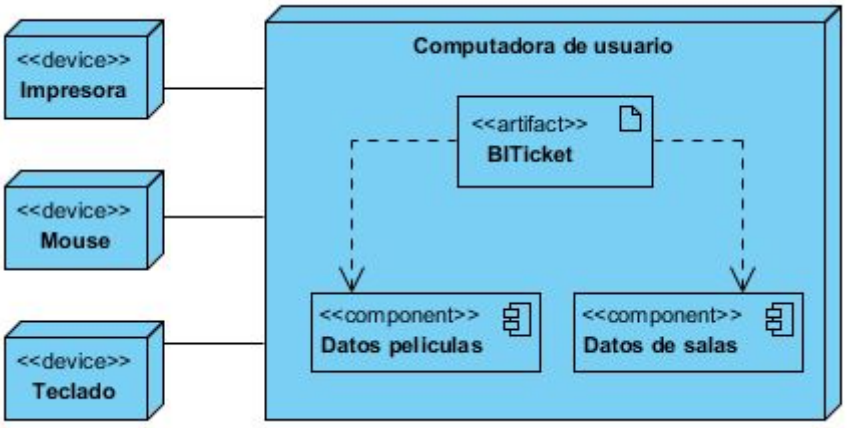


Confirmación de reserva

Vista Física: En esta vista se muestra desde la perspectiva de un ingeniero de sistemas todos los componentes físicos del sistema así como las conexiones físicas entre esos componentes que conforman la solución (incluyendo los servicios). Para completar la documentación de esta vista se puede incluir el diagrama de despliegue de UML.



1- Diagrama de despliegue UML para el sistema BITicket con servidor

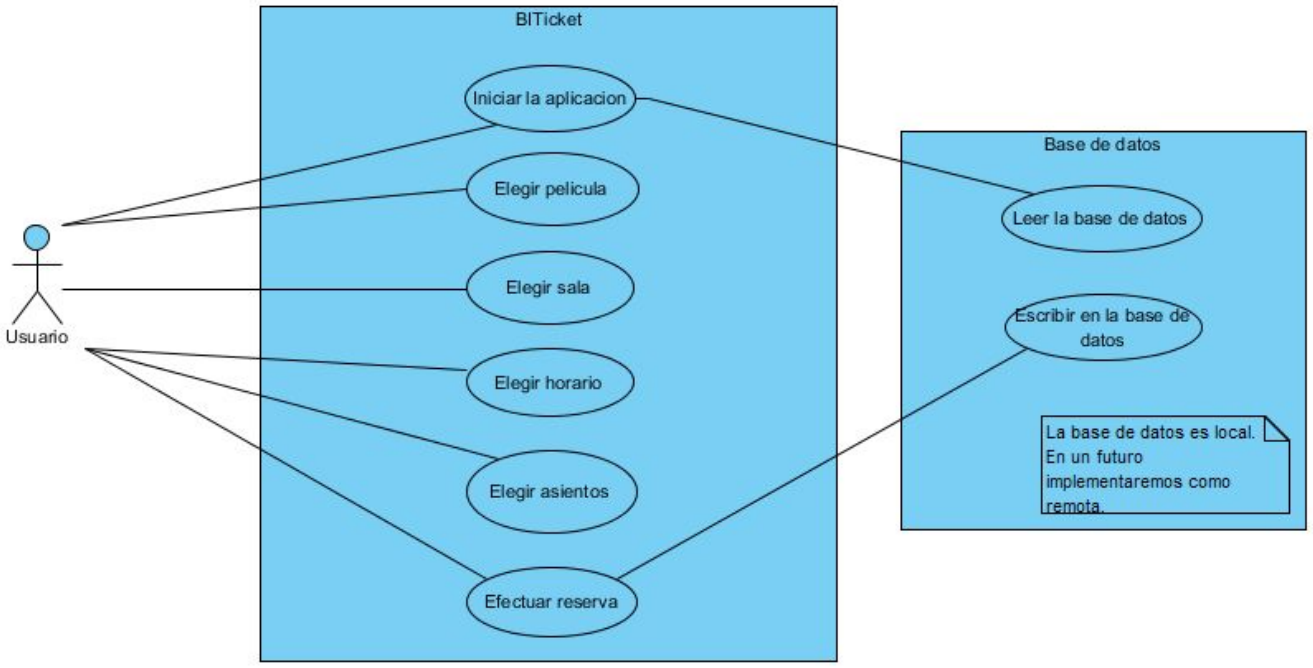


2- Diagrama de despliegue UML para el sistema BITicket sin servidor

Queda abierta la posibilidad de, en un futuro, portar la base de datos a un servidor remoto (ver imagen 1 del diagrama de despliegue).
Por el momento, la base de datos estará incluida dentro de la aplicación tal como se detalla en el diagrama de despliegue imagen 2.

“+1” Vista de Escenarios: Esta vista va a ser representada por los casos de uso software y va a tener la función de unir y relacionar las otras 4 vistas, esto quiere decir que desde un caso de uso podemos ver cómo se van ligando las otras 4 vistas, con lo que tendremos una trazabilidad de componentes, clases, equipos, paquetes, etc., para realizar cada caso de uso. Para completar la documentación de esta vista se pueden incluir el diagramas de casos de uso de UML.

Para la vista de escenarios elegimos la perspectiva que brinda el diagrama de casos de uso, mostrado en la siguiente figura:



Caso de uso de la aplicación BITicket

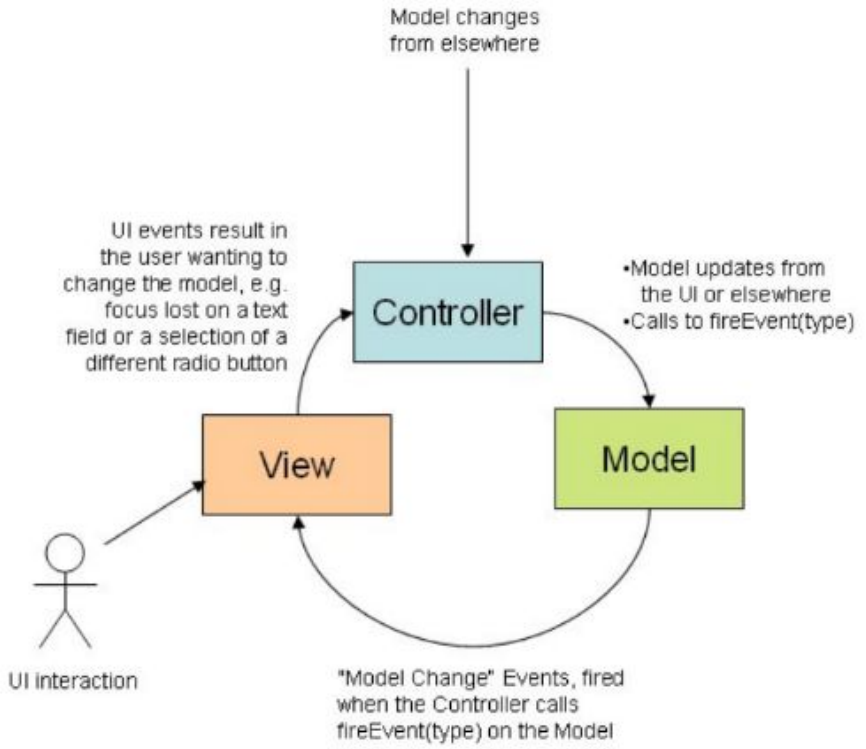
Cabe destacar que la aplicacion debera tener cierta forma de uso “secuencial” esto es, no se podrá efectuar una reserva sin haber elegido la película, la sala y algún horario disponible, entre otras limitaciones.

2.2 Patrón de Arquitectura

El patrón de arquitectura utilizado fue el MVC. El mismo, es un patrón compuesto por otros tres que son:

1. **Singleton**
2. **Observer**
3. **Strategy**

Gráficamente podemos explicarlo con el siguiente diagrama:



Patrón MVC.

El marco de trabajo MVC propuesto originalmente en la década de los 80 como una aproximación al diseño de GU que permitieron múltiples presentaciones de un objeto y estilos independientes de interacción de cada una de estas presentaciones.El marco MVC soporta la presentación de los datos de diferentes formas e interacciones independientes con cada una de estas presentaciones.
Cuando los datos se modifican a través de una de las presentaciones, el resto de las presentaciones son actualizadas. Los marcos de trabajo son a menudo instanciaciones de varios patrones. Por ejemplo, el marco **MVC** incluye el patrón **Observer**, **Singleton** y el patrón **Strategy** relacionado con la actualización del modelo, el patrón Composite y otros patrones.

Gracias a esta arquitectura ahorramos recursos que de lo contrario afectarían a nuestros requerimientos no funcionales.

Por ejemplo, al implementar el patrón **Observer**, nos aseguramos de que el modelo (Sujeto) informe a los observadores (Controlador y Vista) que hubo un cambio, favoreciendo así al requerimiento no funcional **Rendimiento**.

Otro ejemplo sería el uso del patrón **Singleton**, que gracias al mismo favorecemos al requerimiento no funcional **Seguridad** forzando así la unicidad de datos.

Por último, con **Strategy**, favorecemos la reutilización del código y la modularización del sistema.

Las diferentes pruebas y test del proyecto se encuentran detallados en el *Documento de Pruebas*