Metodologías de programación II

Profesora: Claudia Cappelletti

Trabajo Práctico Final: MiEntrada

Alumnos:

Cruz Guillermo.

Gauna Lucas Damián .

Bazán Gerónimo.

Índice

Objetivos

La cátedra de la Materia Metodologías de Programación II de la Universidad Nacional Arturo Jauretche ubicada en Av. Calchaquí 6200, Florencio Varela, Buenos Aires, Argentina. Pidió a sus alumnos que realicen un trabajo final que aporte contenidos a dicha materia. Por lo tanto, luego de evaluar ideas y sugerencias la profesora a cargo Claudia Cappelletti, aprobó la idea del grupo, la cual como trabajo eligió llevar a cabo un sistema de reserva de entradas. MiEntrada, es un software de gestión para reservas de ticket's de Cine que ha sido pensado y desarrollado con el propósito de poder mejorar las reservas de una función de cince, su optimización y modernización de los procesos(También se podría extender a otros negocios, teatros, circos, etc).

Instrumental

Trabajamos en una notebook HP 2570p elitebook que cuenta con una memoria RAM de 4 GB, un microprocesador Intel i5 y un disco rígido de 500 GB. En cuanto a Sistema Operativo utilizamos Windows 10, y en cuanto a aplicaciones para desarrollar el proyecto se utilizó Pharo versión 7.0. El lenguaje de programación fue Smalltalk.

Metodología e Implementación

Para empezar con desarrollo de la aplicación se debió instalar la versión de Pharo 7.0. Luego volcar las ideas conjuntamente en papél realizando los diagramas UML y de Secuencia de esta manera.

Diagrama de Secuencia

Un diagrama de secuencia es una forma de diagrama de interacción que muestra los objetos como líneas de vida a lo largo de la página y con sus interacciones en el tiempo representadas como mensajes dibujados como flechas desde la línea de vida origen hasta la línea de vida destino. Los diagramas de secuencia son buenos para mostrar qué objetos se comunican con qué otros objetos y qué mensajes disparan esas comunicaciones. Los diagramas de secuencia no están pensados para mostrar lógicas de procedimientos complejos.

Diagrama de Clases

Un diagrama de clases sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de agregación, ya que una clase es una descripción de conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones y semántica; mostrando un conjunto de elementos que son estáticos, como las clases y tipos junto con sus contenidos y relaciones. Un diagrama de clases esta compuesto por los siguientes elementos: Clase: atributos, métodos y visibilidad. Relaciones: Herencia, Composición, Agregación, Asociación y Uso.

Una vez tenido los diagramas con las clases y sus atributos más importantes, se prosiguió a desarrollar.

Implementación de la aplicación

|cine f1 f2 f3 f4 f5 f6 f7 s1 s2 s3 s4 p1 p2 p3 p4 p5 p6 c1 c2 c3 c4 col1 col2 col3 col4 dicc|

"aux"
col1 := OrderedCollection new.
col2 := OrderedCollection new.
col3 := OrderedCollection new.
col4 := OrderedCollection new.
dicc := Dictionary new.

Se crea un cine

cine := Cine crearCineNomb: 'NuestroCine'.

Se crean las funciones "las peliculas en cartelera".

f1 := Funcion crearFuncion: 'AVENGERS' horario: '10:00-12:00 hs'.

f2 := Funcion crearFuncion: 'SUPERMAN' horario: '12:00-14:00 hs'.

f3 := Funcion crearFuncion: 'KIMINONAWA' horario: '22:00-24:00 hs'.

f4 := Funcion crearFuncion: 'KILLBILL' horario: '16:30-18:00 hs'.

f5 := Funcion crearFuncion: 'MADAGASCAR' horario: '10:00-12:00 hs'.

f6 := Funcion crearFuncion: 'MATRIX' horario: '18:00-20:00 hs'.

f7 := Funcion crearFuncion: 'RAMBITO Y RAMBON' horario: '18:00-20:00 hs'.

Se crean los productos que estarán disponibles.

p1 := Producto crearProducto: 100 nombre:'Pochoclos dulces' precio:300.

p2 := Producto crearProducto: 101 nombre:'Paso de los Toros' precio:150.

p3 := Producto crearProducto: 102 nombre:'Nachos' precio: 200.

p4 := Producto crearProducto: 103 nombre:'pizza' precio: 550.

p5 := Producto crearProducto: 104 nombre:'Hamburguesa' precio: 420.

p6 := Producto crearProducto: 105 nombre:'Birringui' precio: 170.

Se crearán las salas con su capacidad correspondiente.

s1 := Sala crearSala: 1 capacidad: 50 funciones: f1.

s2 := Sala crearSala: 2 capacidad: 40 funciones: f2.

s3 := Sala crearSala: 3 capacidad: 30 funciones: f3.

s4 := Sala crearSala: 4 capacidad: 80 funciones: f4.

Se crean los clientes con sus id correspondientes.

c1 := Cliente crearCliente: 30111111 nombre:'Ale' codCliente:1.

c2 := Cliente crearCliente: 30222222 nombre:'menem' codCliente:2.

c3 := Cliente crearCliente: 30333333 nombre:'Gero' codCliente:3.

c4 := Cliente crearCliente: 30444444 nombre:'Lucas' codCliente:4.

Acá comenzarán a interactuar las colecciones.

Se agregan las salas y productos al cine creado.

- Productos

```
cine agregarUnProducto: p1.
cine agregarUnProducto: p2.
cine agregarUnProducto: p3.
cine agregarUnProducto: p4.
cine agregarUnProducto: p5.
cine agregarUnProducto: p6.
```

- Salas

```
cine agregarUnaSala: s1.
cine agregarUnaSala: s2.
cine agregarUnaSala: s3.
cine agregarUnaSala: s4.
```

Se agregan los clientes a las funciones creadas.

```
f1 agregarUnPublico: c1.
f2 agregarUnPublico: c4.
f3 agregarUnPublico: c4.
f4 agregarUnPublico: c4.
f3 agregarUnPublico: c2.
f2 agregarUnPublico: c1.
f2 agregarUnPublico: c1.
```

Acá se obtienen las funciones disponibles.

```
"OBTENEMOS LAS FUNCIONES"
col1  := cine obtenerSalas.
col2 := col1 collect: [ :sala | sala obtenerFunciones  ].
```

Se obtiene el nombre de las películas.

col3 := col2 collect: [:fun | fun verPeli].

Se obtiene un diccionario con clave-valor (Pelicula-Horario).

col2 do: [:fun|dicc at: (fun verPeli) put: (fun verHorario)].

Se pueden ver los clientes que reservaron entradas.

col4 := col2 collect: [:fun | fun Verpublico].

dicc inspect.

col1 inspect.

Como posible mejora a futuro se podría implementar una interfaz para la aplicación para que sea mas amigable ante el usuario/cliente.

Principales funciones

El sistema de MiEntrada tenía como funciones principales:

Reservar, verDisponibilidad, verFunciones.

Problemas encontrados

Mediante el desarrollo del proyecto, nos fuimos encontrando con diferentes problemas. Tanto como almacenar los módulos de los sprint en algún Repositorio Free, como el de errores de sintaxis para implementar la composición y por ende la aplicación.

Soluciones

Se investigó acerca de las nuevas herramientas que trabajan de manera repositorio, y también más a fondo el lenguaje Smalltalk para poder solucionarlo.

Metodología utilizada

Para llevar a cabo la gestión del proyecto, se utilizó la Metodología ágil Scrum. Se basa principalmente en idea de ejecutar un proyecto en entregas parciales y regulares del producto. El desarrollo del producto se realiza de forma incremental y evolutiva, lo que resulta ideal en entornos dinámicos y cambiantes. Scrum nos proporcionó un marco de trabajo que permitió al equipo auto-organizarse.

Los perfiles en el ámbito académico pueden ser intercambiados, pero en el ámbito profesional no es conveniente ya que cada uno cuenta con un propósito explícito.

Lo más importante fueron las reuniones, las cuales son el pilar del proceso.

Se realizaron al principio de cada Sprint para decidir que se va a realizar en él, asignando tiempo a cada una de ellas. Y son la base para la comunicación y lograr un equipo ágil y multifuncional, estas se desarrollaron tanto en el horario del ámbito académico, como vía acceso remoto mediante el uso de aplicaciones como TeamViewer y Skype.

Para llevar al pie de la letra esta metodología se utilizó la herramienta Trello.

Es una herramienta para la organización de tareas. Es ideal para la coordinación de equipos de trabajo y se basa en la metodología Kanban, la cual propone un sistema de uso colaborativo. En ella se fueron creando y definiendo, tanto las historias de usuario como los sprint's.

Mejoras en el futuro

Para posibles mejoras en el futuro se podrían poner en funcionamiento tanto patrones de diseño como el uso de framework's. Así mismo utilizar refactorización en algún caso que se decida estas metodologías lo permitirán.

Posibles patrones

Para el contexto de nuestro proyecto, se podrían implementar los siguientes patrones de diseño:

- **Observer:**

Observer es un patrón de diseño de software que define una dependencia del tipo uno a muchos entre objetos, de manera que cuando uno de los objetos cambia su estado, notifica este cambio a todos los dependientes. Se trata de un patrón de comportamiento por lo que está relacionado con algoritmos de funcionamiento y asignación de responsabilidades a clases y objetos.

El ***Observer*** se podría implementar para avisar a un listado de clientes que peliculas se estrenarán.

- **State:**

El patrón de diseño State se utiliza cuando el comportamiento de un objeto cambia dependiendo del estado del mismo.

El ***State*** se podría implementar para manejar el estado de la butaca, este podría congelar su estado así en el caso de que no se concrete la reserva, volvería a su estado anterior, que sería Disponible.

- **Front Controller** :

El patrón de diseño del controlador frontal se utiliza para proporcionar un mecanismo centralizado de manejo de solicitudes para que todas las solicitudes sean manejadas por un solo manejador. Este controlador puede realizar la autenticación / autorización / registro o seguimiento de la solicitud y luego pasar las solicitudes a los controladores correspondientes. Las siguientes son las entidades de este tipo de patrón de diseño.

- Controlador frontal: controlador único para todo tipo de solicitudes que llegan a la aplicación (ya sea en la web / en el escritorio).
- Despachador : el controlador frontal puede usar un objeto despachador que puede enviar la solicitud al controlador específico correspondiente.
- Vista : las vistas son el objeto para el que se realizan las solicitud.
-

El ***Controller,*** funcionaria como un mediador entre el cliente, el cine y la sala.

Posible Framework

Para el contexto de nuestro proyecto, se podría implementar el **Seaside**, es un framework del lenguaje de programación Smalltalk que se utiliza para el desarrollo web. El **Seaside** se utiliza para realizar el entorno gráfico entre el ***Cine*** y los ***Clientes.***

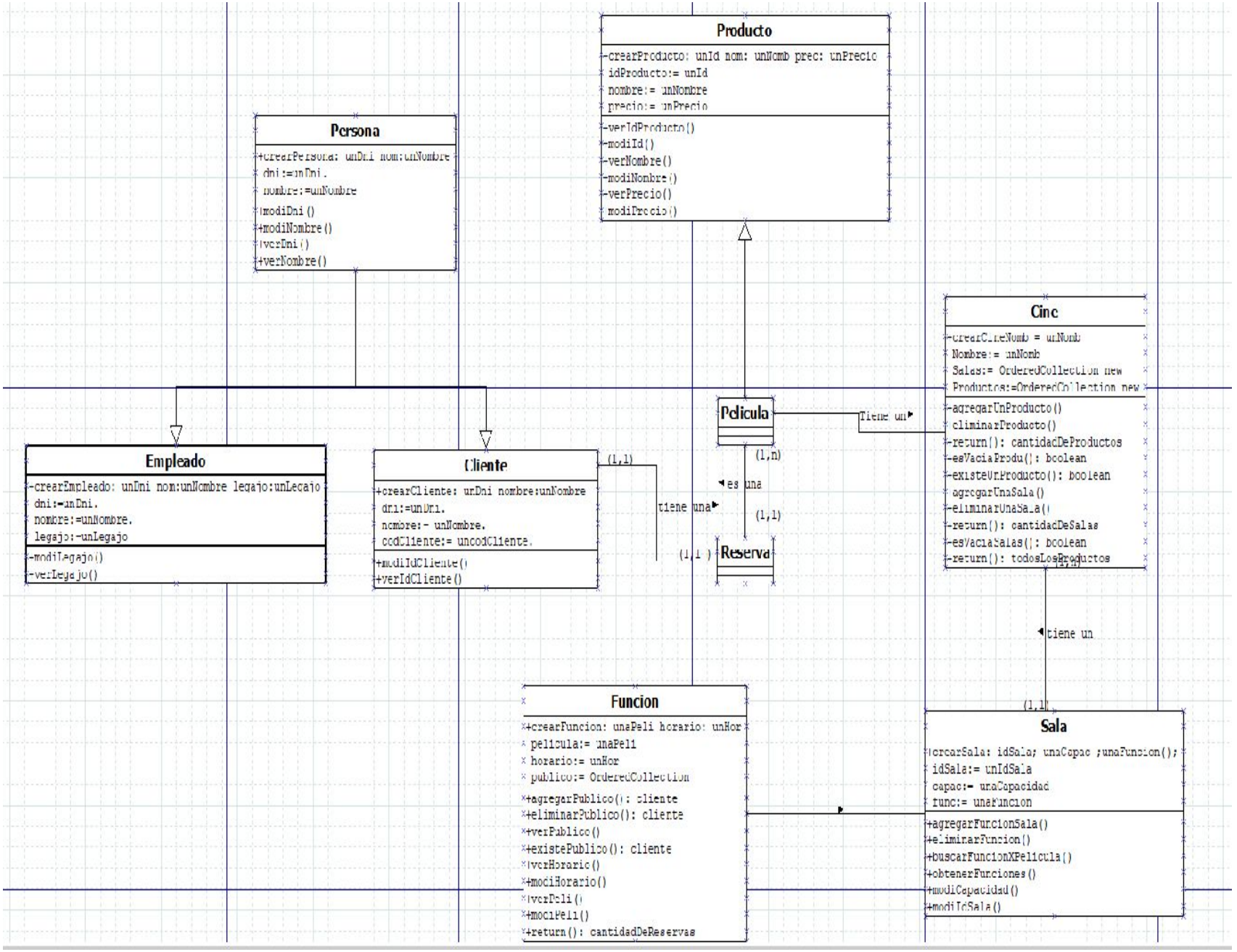
Repositorios

El repositorio que se utilizó para el desarrollo de aplicación colaborativo fue, GitHub. Es un sistema de gestión de proyectos y control de versiones de código, permite trabajar en colaboración con otras personas de todo el mundo, planificar proyectos y realizar un seguimiento del trabajo. Permite almacenar tanto en la nube como localmente.

Conclusión

Se llegó a los objetivos esperados. Realizar la aplicación en un lenguaje orientado a objeto puro como lo es Smalltalk, nos demostró que aplicando los contenidos vistos en clase cómo es que interactúan, los objetos de manera secuencial y didácticamente. También se aplicaron metodologías ágiles de trabajo, para llevarlo a cabo lo que ayudó bastante en la organización común del grupo.

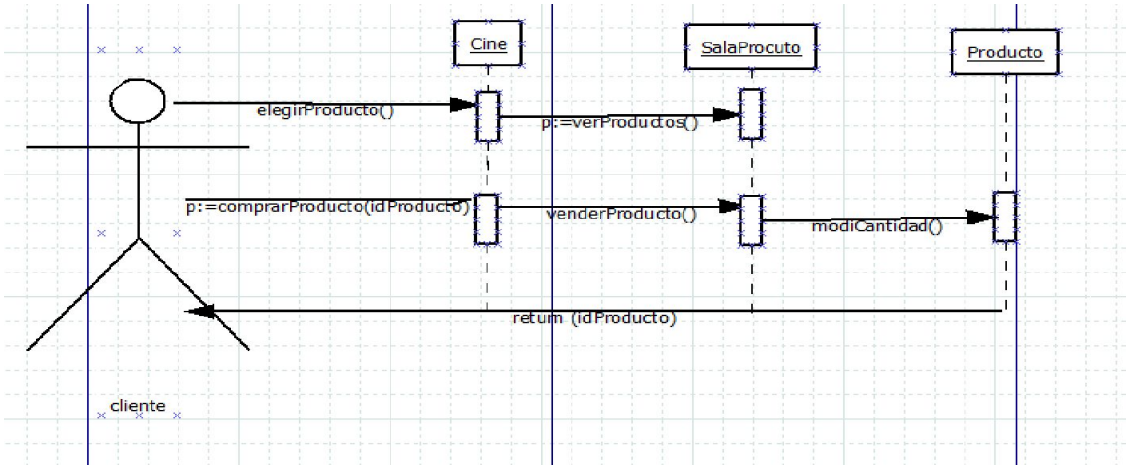
ANEXO 1


Producto
-crearProducto: unId nom: unNomb prec: unPrecio
idProducto:= unId
nombre:= unNombre
precio:= unPrecio
-verIdProducto()
-modiId()
-verNombre()
-modiNombre()
-verPrecio()
-modiPrecio()
Persona
+crearPersona: unDni nom:unNombre
dni:=unDni.
nombre:=unNombre
+modiDni()
+modiNombre()
+verDni()
+verNombre()
Cine
-crearCineNomb = unNomb
Nombre:= unNomb
Salas:= OrderedCollection new
Productos:=OrderedCollection new
-agregarUnProducto()
-eliminarProducto()
-return(): cantidadDeProductos
-esVaciaProdu(): boolean
-existeUnProducto(): boolean
-agregarUnaSala()
-eliminarUnaSala()
-return(): cantidadDeSalas
-esVaciaSalas(): boolean
-return(): todosLosProductos
Empleado
-crearEmpleado: unDni nom:unNombre legajo:unLegajo
dni:=unDni.
nombre:=unNombre.
legajo:=unLegajo
-modiLegajo()
-verLegajo()
Cliente
+crearCliente: unDni nombre:unNombre
dni:=unDni.
nombre:= unNombre.
codCliente:= uncodCliente.
+modiIdCliente()
+verIdCliente()
Pelicula
Reserva
Tiene un
(1,n)
es una
(1,1)
tiene una
(1,1)
(1,1)
tiene un
(1,1)
Funcion
+crearFuncion: unaPeli horario: unHor
pelicula:= unaPeli
horario:= unHor
publico:= OrderedCollection
+agregarPublico(): cliente
+eliminarPublico(): cliente
+verPublico()
+existePublico(): cliente
+verHorario()
+modiHorario()
+verPeli()
+modiPeli()
+return(): cantidadDeReservas
Sala
+crearSala: idSala; unaCapac ;unaFuncion();
idSala:= unIdSala
capac:= unaCapacidad
func:= unaFuncion
+agregarFuncionSala()
+eliminarFuncion()
+buscarFuncionXPelicula()
+obtenerFunciones()
+modiCapacidad()
+modiIdSala()

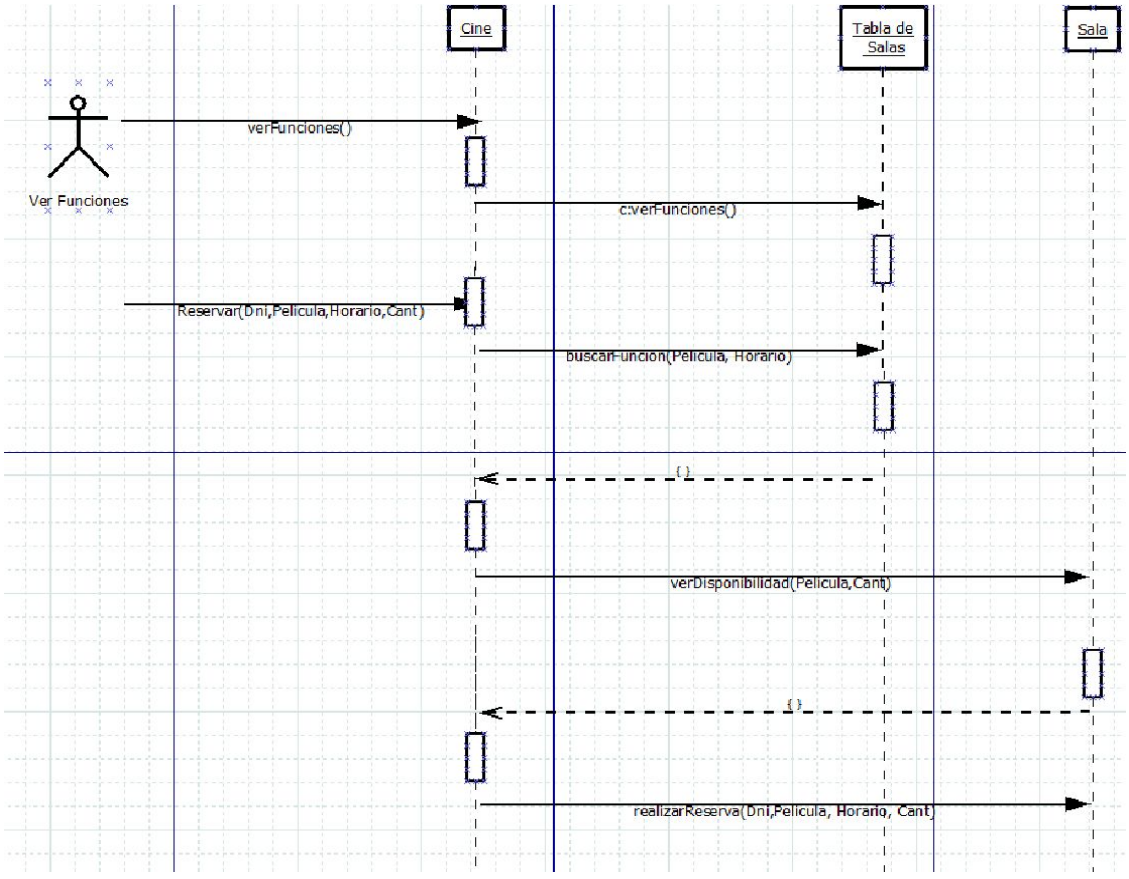
ANEXO 2

Diagramas de Secuencia:

- Producto

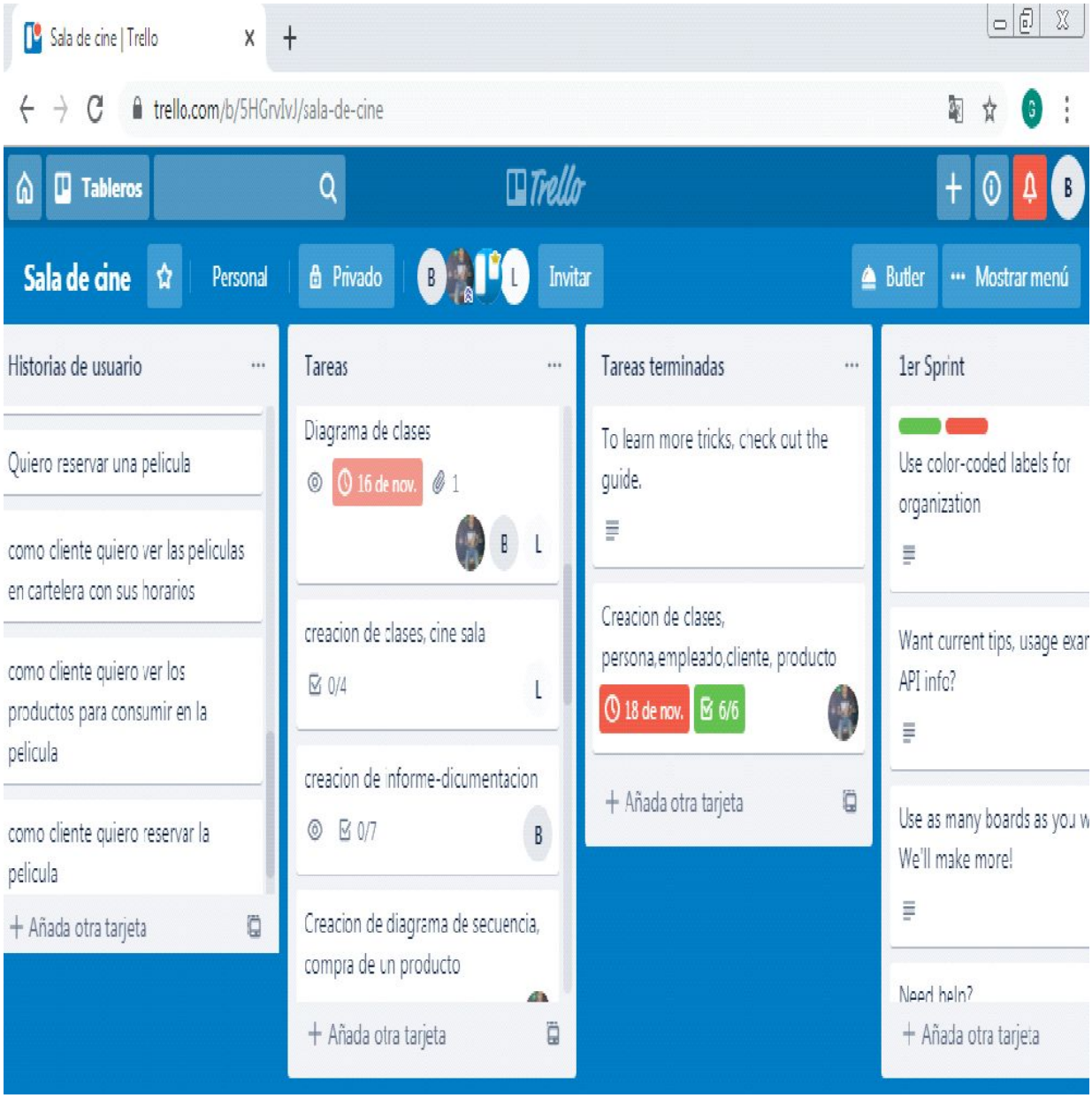


- Reserva de entrada



ANEXO 3

Administración de tareas: Sprint's e historias de usuarios.



ANEXO 4

Repositorio utilizado.