`Cine++`

Informe individual de Ingeniera de Software

Carlos Luis Aguila Fajardo

16 de Junio, 2021

- Equipo #3 del Profesor Jose L. Castañeda
- Username de Github: `kvothe99`

1 Introducción:

En el documento actual se describe las actividades realizadas por mí durante el desarrollo del proyecto `Cine++`, orientado al desarrollo de un entorno dedicado al negocio de un cine con distintas películas, horarios y salas, así como a la adminsitración del mismo.

2 Desarrollo y Actividades Específicas:

Views e interfaz:

Durante el proceso inicial del proyecto se decidió utilizar el patrón *Model-View-Controller* (MVC), por su facilidad para separar las actividades en distintos grupos relativamente desacoplados. Uno de los aspectos más importantes de este patrón de diseño son los Views, los cuales fundamentalmente rigen el aspecto que va a tener el proyecto para el usuario final, así como para los administradores del cine.

Para comenzar el desarrollo de los mismos fue necesario diseñar un primer prototipo de interfaz, se decidió que la misma debería tener una barra de navegación, con el nombre del proyecto visible que dirigiera al menú o página principal. Adicionalmente se consideró apropiado que existiera un diseño intuitivo para la selección de la sala, horario y asientos durante el proceso de compra de las entradas.

Layout común:

Se comenzó entonces diseñando la barra de navegación en el primer view `_Layout.cshtml`, el cual describe la estructura y aspecto en común de todos los views del proyecto, por ello se encuentra ubicado en el directorio `Views/Shared/`. La misma inicialmente contaba con acceso a las secciones **Admin**, **Catálogo**, **Registrarse** y **Estadísticas**. Esta última sección fue desplazada posteriormente para las opciones de administradores, puesto que nos percatamos que no era de interés que los clientes del cine visualizaran estos datos. Se procede entonces a desarrollar los views de administración de los modelos diseñados apropiados, es decir, los views correspondientes a las operaciones llamadas **CRUD**, que permitieran listar, añadir, editar y eliminar nuevas entidades de los modelos del proyecto, como **Filme**, **Socio**, **Sala**, **Horario**, entre otros; muchos de estos orientados a actividades asociadas al administrador del cine, aunque se utilizó la creación de nuevos socios para encapsular el proceso de subscripción de nuevos socios al cine, además de permitir que un administrador del cine añada o remueva dichos socios a su interés.

Dos de las actividades más complicadas de representar en el proyecto fueron las de Estadísticas y el proces de Compra de entradas, puesto que ambas requieren trabajo con formularios y una cuidadosa interconexión entre *queries* sobre los modelos y contextos de la base de datos, varios controladores y los

views apropiados, así como la validación de los formularios y el reporte en caso de errores en los datos provistos.

Estadísticas:

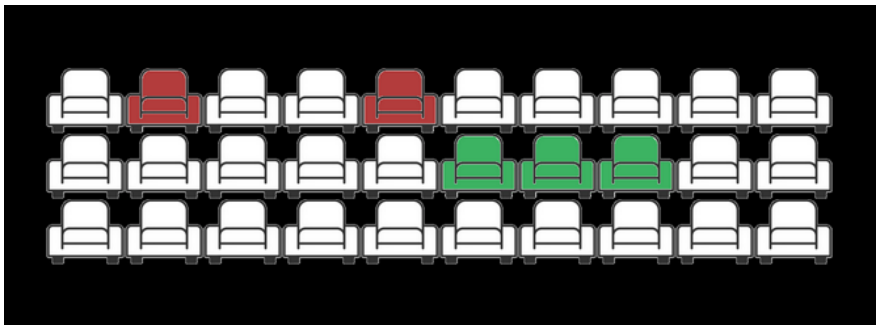
El proceso de Estadísticas fue relativamente simple de diseñar, pero algo complicado de implementar; se decidió mostrar las estadísticas más básicas (cantidad de ventas en el día, mes, año y en total) en forma de dashboard de la página de administrador, y manejar el resto de las estadísticas con un formulario. Dicho formulario tenía que ser capaz de mostrar un rango de fechas configurables cuando se seleccionaba la opción de Estadística por Período, pero debía ser capaz de ocultarlo cuando se seleccionaba otra opción. Esto se manejó con el script de *JavaScript* siguiente, encontrado en `Views/Estadisticas/Details.cshtml`:

```
dropdown.addEventListener('change', () => {
    // Si la opción es Período...
    if (dropdown.value !== '4')
        dates.style.display = "none";
    else
        dates.style.display = "flex"
});
```

Cualquiera de los datos seleccionados en dicho formulario son entonces procesados en el controlador de Estadísticas y pasados al view `.../Show.cshtml` utilizando el *view model* llamado `EstadisticasShowModel`, que dentro contiene una colección de pares de tipo `(string, int)`, correspondientes a la cantidad de entradas vendidas por grupo de datos obtenidos de la consulta sobre la base de datos.

Compra:

En caso de los views y controladores correspondientes a la compra de entradas, se necesitaba mostrar visualmente los asientos previamente seleccionados y los actualmente seleccionados por el comprador, de forma que se diseñó un script de *JavaScript*, junto con varias clases de *CSS* para representar los asientos de la sala seleccionada actualmente, de forma similar a la siguiente imagen:



Otro obstáculo a resolver durante la implementación de esta sección fue la necesidad de dinámicamente procesar el monto total de la cantidad de entradas, teniendo en cuenta si se aplica o no descuento así como si fue provista una tarjeta de crédito (por lo que se descontaría el monto en dólares) o un ID de Socio (entonces se procedería a descontar el monto en puntos de socio). Esto se logró implementando scripts de `JavaScript` en el frontend para ahorrar llamados innecesarios al backend del servicio, de forma que se envía información hacia el servidor solo cuando se desea submitir el formulario y validar los datos provistos.

Todos los scripts utilizados en esta sección se pueden ver en la sección final del view `Views/Compra/Comprar.cshtml`.

Adicionalmente, se diseñó el modelo adicional `CompraViewModel` para intercambiar datos entre el controlador y los views de Compra. En dicho modelo se utilizó *DataAnnotations* para validar las entradas provistas por el formulario, así como atributos de validación personalizados `PaymentValidationAttribute` y `SocioIdValidationAttribute` para validar los tipos de pago permitidos.

Controllers y Lógica:

Como fue mencionado anteriormente, parte de mis actividades durante el proyecto fueron orientadas a enlazar los controladores con los views diseñados. Por tanto tuve que realizar varias ediciones sobre los controladores existentes, así como diseñar la lógica de algunos de ellos; particularmente modifiqué en gran parte la arquitectura lógica del controlador de Estadísticas y Cancelación de Compra, e implementé gran parte del comportamiento del controlador de Compra.

Para la comunicación apropiada entre estos controladores mencionados y sus correspondientes views, fue necesario, como en parte mencioné anteriormente, diseñar los modelos `CompraViewModel`, `EstadisticasFormModel` y `EstadisticasShowModel`. Vale mencionar que estos no tienen nada que ver con los modelos de la base de datos diseñados por otro de mis compañeros, puesto que los utilizados aquí son *view models*, creados para comunicar views con controladores, y que no representan ningún tipo de datos en la BD.

Planificación:

Además de las actividades descritas, durante el desarrollo del proyecto mantuve al día las planificaciones acordadas, así como los retrasos y cambios en las actividades realizadas, todo ello representado en un proyecto de Gantt, el cual se puede observar en `Informes/gantt/`. Adicionalmente se pueden observar ahí la asignación de los miembros a las actividades, así como la distribución por sprints de las mismas.

Resumen:

I. Primer grupo de actividades (de cara al cliente):

- *View Models*:
  - `CompraViewModel.cs`
- Controladores:
  - `CancelacionCompraController.cs`
  - `CompraController.cs`
- Views:
  - `Home/Index.cshtml`
  - `CancelacionCompra/Index.cshtml`
  - `Compra`
    `/ Comprar.cshtml`
    `/ Details.cshtml`
    `/ Index.cshtml`
  - `Shared/_Layout.cshtml`

II. Segundo grupo de actividades (de cara al administrador):

- *View Models*:
  - `EstadisticasFormModel.cs`
- Controladores:
  - `AdminController.cs`
  - `EstadisticasController.cs`
- Views:
  - `Estadisticas`
    `/ Show.cshtml`
    `/ Details.cshtml`
    `/ Index.cshtml`
  - `Admin/Index.cshtml`
  - `Sala/*` y `Filme/Index.cshtml` (modificaciones a las propiedades mostradas)

III. Actividades adicionales:

- `Login.aspx` y `Register.aspx`; páginas creadas para la autenticación, al no ser views de Razor, hubo que hacerles un estilo nuevo basado en bootstrap sin las clases de *CSS* o la barra de navegación del layout.
- las planificaciones encontradas en `Informes/gantt/`.

Detalles Técnicos:

Los views del proyecto se encuentran escritos en `Razor`, un lenguaje integrado al framework `ASP.NET` que integra código imperativo de `C#` con lenguaje de markup `HTML`, y que adicionalmente permite la interacción con modelos pasados como parámetro desde los controladores correspondientes.

Para el aspecto visual se utilizó una mezcla de `Bootstrap v5.0.1` (una colección de librerías `CSS` gratuitamente distribuídas) y *CSS* personalizado; para funcionalidades adicionales, como el seleccionador de fechas en las estadísticas, se utilizó `jQuery`.