



### **Ejecutores:**

Roger Moreno Gutiérrez

Claudia Alvarez Martínez

Kevin Majim Ortega Alvarez

Yoan René Ramos Corrales

## Contents

I. Introducción: .....	2
II. Requerimientos Específicos:.....	4
III Funcionalidades del Producto: .....	7
IV. Enfoque Metodológico: .....	7
V. Arquitectura: .....	8
VI. Patrones de visualización y de datos:.....	13
VII. Modelo de Datos:.....	22
VIII. Diccionario de datos .....	23
IX. Esquemas de clases definidas .....	26
X. Opciones del sistema .....	29
XI. Salidas del sistema.....	34
XII. Análisis de modificaciones.....	37

### **I. Introducción:**

- Alcance del producto:

El producto va dirigido a la administración y comercialización de las entradas de Cine+ implementada sobre una plataforma web. Los usuarios del producto serán el personal que trabaja en el cine y clientes que se sientan atraídos por el séptimo arte.

- Descripción General:

Perspectiva y funciones del producto:

- a. Permitir la compra de entradas por parte de los clientes.
- b. Controlar la venta de entradas.
- c. Mostrar salas y horarios para la compra de entradas.
- d. Controlar las butacas disponibles y dar al cliente la posibilidad de elegir las.

- e. Tener en cuenta la aplicación de descuentos para calcular el precio final de la entrada.
- f. Permitir asociaciones de clientes al club Cine+ llenando formulario desde el sitio web o desde la taquilla, los que tendrán ciertos beneficios.
- g. Permitir exportación del comprobante de venta tras la compra por el sitio web.
- h. Permitir la cancelación de las compras hechas por la web.
- i. Los gerentes del cine podrán administrar listados de películas, horarios, consultar estadísticas.
- j. Mostrar listado de sugerencias de películas.

#### Características de los usuarios:

- a. Gerente: persona con conocimientos de cine y nivel medio superior capaz de organizar y gestionar Cine+.
- b. Taquilleros, personas con conocimientos elementales de uso y manejo de una computadora.
- c. Clientes, personas con conocimiento de cine o no, capaces de interactuar con un sitio sencillo, pero que a su vez le permita buscar información que satisfaga sus intereses (información acerca del sitio, programaciones de la semana...).

#### Restricciones Generales:

- a. Mostrar el logo del cine.
- b. Tener una página de contacto.

#### Resumen:

Una de las necesidades básicas de todo negocio que se encuentra en crecimiento; es su expansión hacia la red de redes, por lo cual el desarrollo de una página web es totalmente necesario. Cine+ será una página creada para garantizar que toda la venta de entradas de un(varios) cine(s) pueda

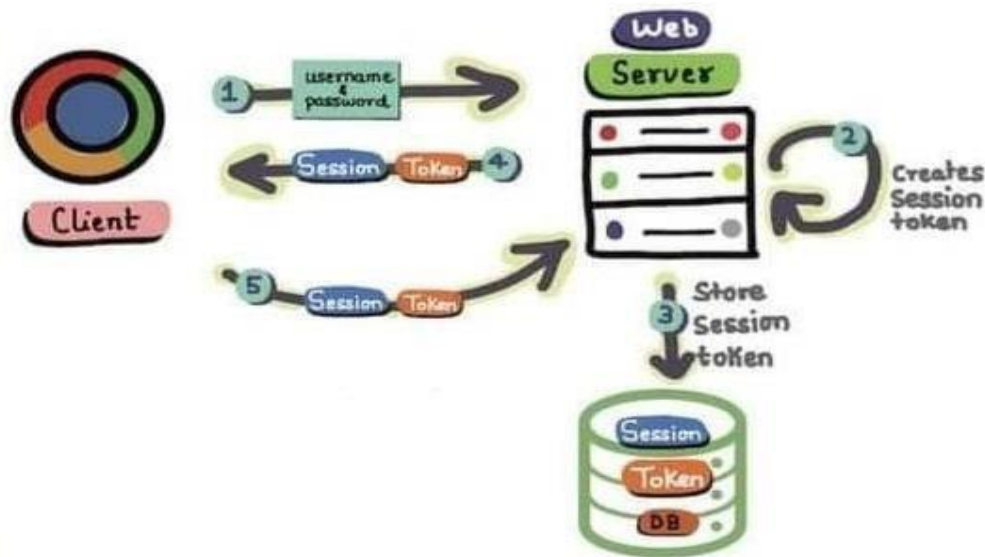
hacerse desde la comodidad del hogar, ahorrando tiempo y largas filas para la compra de las mismas. Se han planteado una serie de requerimientos específicos que debe cumplir la página a los cuales se les brinda solución con las funcionalidades de nuestro producto. Utilizando metodologías ágiles como crystal clear, la cual se vio totalmente efectiva para este tipo de trabajo, se logró una sólida organización. Luego si a estas características le sumamos la arquitectura N-Layered, obtenemos un producto lo suficientemente desglosado para su posterior mantenimiento.

**Palabras clave: Requerimientos específicos, Crystal Clear, N-Layered.**

## **II. Requerimientos Específicos:**

- **Requerimientos Funcionales:**
  - a. **Permitir la compra de entradas:** Los clientes deberán estar registrados previamente, dando a conocer ciertos datos de su información personal (documento de identidad, tarjeta de crédito, etc.), y estar autenticados.
  - b. **Controlar venta de entradas:** La venta de entradas deberá ser controlada de manera simultánea desde taquilla y la compra online.
  - c. **Seleccionar película, sala y horario:** Deberá visualizarse la lista de películas con la sala de proyección y sus horarios.
  - d. **Asignación de butaca:** Una vez hecha la selección de una programación, el cliente deberá seleccionar su asiento entre las butacas disponibles.
  - e. **Realizar pago:** El cliente en el momento del pago podrá aplicar alguno de los descuentos disponibles.
  - f. **Registrarse como socio del club Cine+:** Los clientes que deseen ser socios de Cine+ deberán llenar un formulario con datos personales (anteriormente mencionados) al que se le atribuirá un código único, el cual utilizará siempre que realice sus compras para participar en el programa de puntos del club. El cliente puede acreditarse directamente desde el sitio web cumpliendo el requisito de haber estado registrado, o a través de un vendedor de taquilla el cual rellenará el formulario.

- g. **Emitir recibo de ticket electrónico:** Tras realizar la compra de entrada, el cliente tendrá acceso a un comprobante con los datos de la sesión, el cual podrá exportar y deberá mostrar el día de su asistencia a Cine+.
  - h. **Cancelación de compras:** Los clientes de la web, tras realizar una compra tendrán la posibilidad de cancelar dicha compra, pero solo hasta dos horas antes del inicio de la sesión.
  - i. **Administración de Cine+ por el gerente:** El gerente de Cine+ podrá remover y añadir películas, manejar horarios, consultar estadísticas de venta según diversos criterios, etc. una vez esté autenticado.
  - j. **Listar sugerencias de películas:** Los usuarios podrán acceder a un listado de sugerencias de películas conformada por el gerente según varios criterios a su elección.
- **Requerimientos No Funcionales:**
    - Seguridad:
      - a. **Autenticación de usuarios:** Se contará con 3 tipos de usuarios: gerente, taquilleros y clientes. Estos deberán autenticarse para poder acceder a funciones específicas de los mismos mediante un nombre de usuario y contraseña.
      - b. **Creación de cuentas de usuarios:** Dentro del módulo administrativo el gerente será el encargado de crear las cuentas de acceso a los trabajadores del cine (taquilleros de taquilla).



#### Usabilidad:

De forma general, los usuarios del producto no deben tener conocimientos especializados, solamente un dominio básico para la interacción con un sitio web.

#### Requerimientos de Diseño e Implementación:

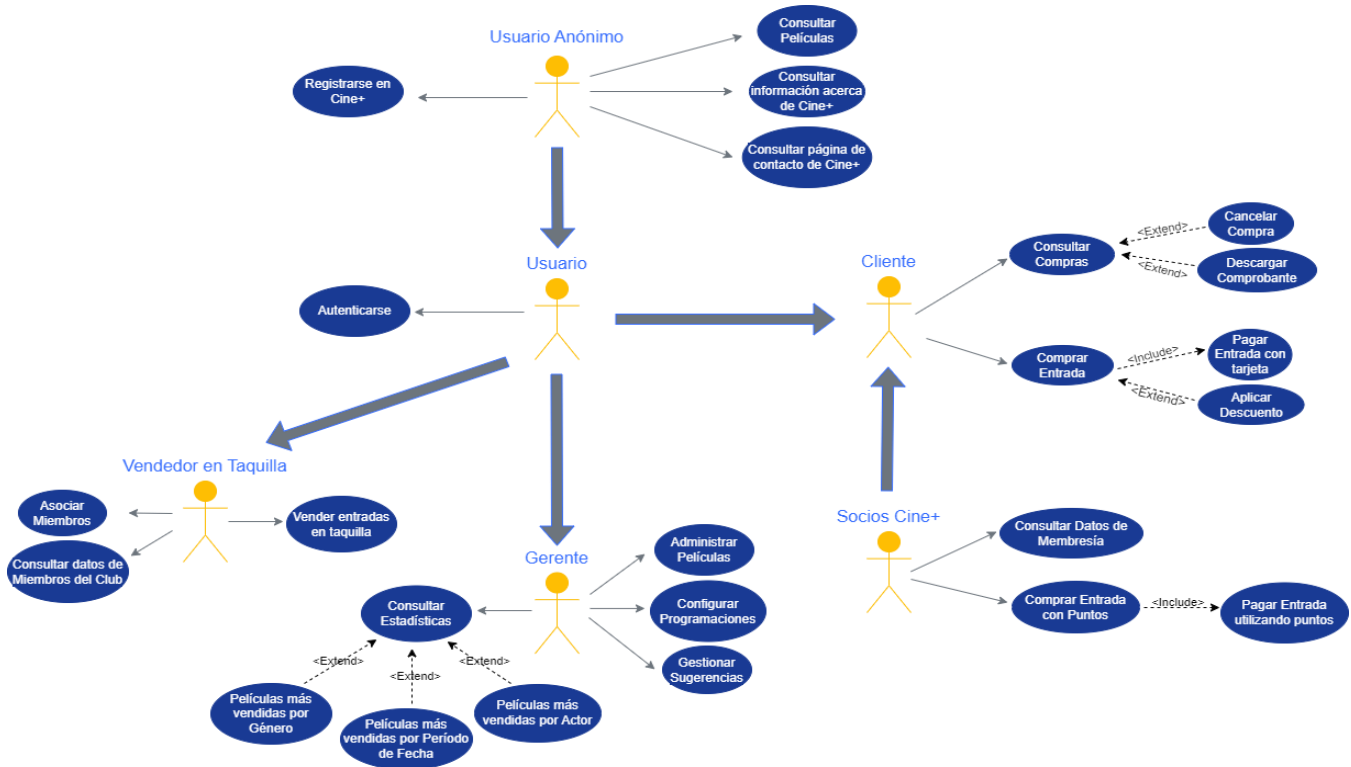
No se presentó ningún requerimiento de este tipo.

- **Requerimientos de Entorno:**

Nuestro cliente y su personal, dominan el Sistema Operativo Windows, así que la aplicación web estará dirigida más bien para este sistema operativo. Sin embargo, al ser un producto desarrollado con propósito web y dirigido en gran parte al público, será accesible desde cualquier navegador web moderno, es decir será multiplataforma gracias a ASP.NET en su versión NetCore. La base de datos se encontrará creada en MySQL y para el resto de las dependencias usaremos C#.

### III Funcionalidades del Producto:

#### Diagrama de casos de usos:



### IV. Enfoque Metodológico:

Para el desarrollo del proyecto, hemos tomado como enfoque metodológico Crystal Clear, fundamentalmente porque promueve la comunicación efectiva, la colaboración y la simplicidad en el proceso de desarrollo.

Como nuestro equipo de trabajo cuenta con personal con cierta experiencia, no es necesaria la búsqueda excesiva de información, con cada entrega que se le haga al cliente podemos ir obteniendo el conocimiento necesario para la creación del resto de funcionalidades. Además, nuestro equipo es bastante cercano por lo que podemos tener comunicación bastante a menudo, conocer los problemas que tiene el resto; este enfoque es muy importante en este tipo de metodologías. Para cubrir las deficiencias de este enfoque hemos diseñado los roles que van a seguir cada uno de los integrantes del equipo, por lo cual tendríamos un líder de proyecto y cada

participante del equipo sabe cuáles son las tareas y el rol que juega en la creación del proyecto; otra razón por la cual se ha decidido optar por esta metodología es la poca experiencia de los participantes en el trabajo en equipo. Aunque bien los participantes pueden tomar las decisiones que crean convenientes siempre y cuando cumplan las tareas asignadas.

## V. Arquitectura:

Para el desarrollo de nuestro proyecto nos hemos basado en la arquitectura N-Layered. Para ello adoptamos las capas de **presentación**, la de **lógica de negocios**, la de **seguridad** y la de **acceso a datos**.

### Capa de Presentación (Cliente) - React con TypeScript:

Dentro de la capa de presentación se observa toda la interacción del usuario con la página. Permitiéndole registrarse en el sistema, y una vez registrado permite su autenticación; para esto el cliente proporciona su nombre de usuario y contraseña, la capa de presentación se encarga de entregar estos datos a las capas siguientes para su manejo. En función del token generado, y el rol del usuario que inicia la sesión, se renderiza la interfaz de usuario correspondiente (de ellos se encarga la componente Switch.tsx).

#### Ejemplo de renderización del usuario anónimo

```
return (  
  <div>  
    {role === "unknown" && (  
      <Layout navLinks={UnknownNavLinks}>  
        <Routes>  
          {UnknownUserRoutes(setToken).map((route, index) => {  
            const { element, ...rest } = route;  
            return <Route key={index} {...rest} element={element} />;  
          })}  
        </Routes>  
        <Footer/>  
      </Layout>  
    )}  
  )
```



Cada rol de usuario tiene definida ciertas rutas a las que este tiene acceso, renderizando para cada caso las correspondientes componentes.

```
const UnknownUserRoutes = (tokenSetter: React.Dispatch<React.SetStateAction<string | null>>)=> [
  {
    index: true,
    element: <Home />
  },
  {
    path: '/log-in',
    element: <LoginPage tokenSetter={tokenSetter} />
  },
  {
    path: '/about-us',
    element: <AboutPage />
  },
  {
    path: 'sign-up',
    element: <SignUpPage />
  },
  ...
]
```

## Capa de Seguridad (Servidor) - C#:

La capa de seguridad se encarga de la encriptación de las contraseñas de los usuarios, y su posterior recuperación, además de la generación de un token único que se le entregara al cliente para ser enviado con cada solicitud. La capa de seguridad, ante la generación del token verifica primeramente que realmente esa es la contraseña de ese usuario. Esta capa está relacionada con la capa de presentación, ya que ante cada solicitud se verifica la expiración del token.

### Encriptación de Contraseña

```
string salt = BCryptNet.GenerateSalt();
string hashed_pass = BCryptNet.HashPassword(input.Password, salt);
```

### Verificación de Contraseña

```
private bool VerifyPassword(string hashedPassword, string salt, string password)
{
    string hashed_pass = BCryptNet.HashPassword(password, salt);
    if (hashedPassword == hashed_pass) return true;
    return false;
}
```

### Generación del Token

```
string GenerateJwtToken(User user, string role, string nick)
{
    var securityKey = new
        SymmetricSecurityKey(Encoding.UTF8.GetBytes(this.jwtSettings.securitykey));

    var credentials = new SigningCredentials(securityKey,
        SecurityAlgorithms.HmacSha384);

    var claims = new[]
    {
        new Claim(ClaimTypes.NameIdentifier, user.UserId.ToString()),
        new Claim(ClaimTypes.Role, role),
        new Claim("Nick", nick)
    };

    var token = new JwtSecurityToken(
        claims: claims,
        expires: DateTime.UtcNow.AddHours(2),
        signingCredentials: credentials
    );

    return new JwtSecurityTokenHandler().WriteToken(token);
}
```

### Capa de Lógica de Negocios (Servidor) - C#:

Esta capa es la encargada de manejar todas las solicitudes que realiza el cliente. Se basa en los controladores que se encuentran dentro de la carpeta 'Controllors'. Esta capa recibe las solicitudes que provienen directamente de la capa de presentación, las procesa (un ejemplo de procesamiento lo tenemos en el momento

que revisamos los nombres de usuario de cada uno de los usuarios registrados, para verificar que no coincida ninguno.) y luego le envía la respuesta al cliente. Aquí tenemos controladores que realizan operaciones CRUD para la configuración de Cine+ y otros encargados de realizar acciones o recuperar datos según la petición y necesidad del usuario.

#### Registro de Usuarios

```
[HttpPost]
public async Task<IActionResult> Register([FromBody] FormInput input)
{
    if (input == null) {return BadRequest("Invalid Data"); }

    // Verificar que no haya usuario registrado con mismo nick o documento de
    // Identidad

    if (_context.Users.Any(input.nick == nick) || _context.Clients.Any(input.DNI = DNI)
    {return BadRequest("Invalid Data"); }

    ...

    // Hashed password
    ...
    // Agregar usuario a la BD

    // Guardar cambios en la BD
    await _context.SaveChangesAsync();

    return Content("Valid Response.");
}
```

### Seleccionar Programaciones Disponibles

```
[HttpGet]
public async Task<IActionResult> GetAvailableProgramming()
{
    List<ProgrammingData> availableProgramming = new List<ProgrammingData>();

    DateTime time = DateTime.Now;

    // Seleccionar las programaciones que aún no han iniciado su sesión
    var programming = await _context.ScheduledMovies
        .Where(p => p.DateTimeId > time)
    ...

    // Quedarnos, de las programaciones disponibles, solo con las que tienen asientos
    libres
    ...

    return Ok(availableProgramming);
}
```

### Capa de Acceso a Datos (Servidor) - C# con Entity Framework:

En esta capa se encuentra nuestra base de datos con todos los datos almacenados, para realizar solicitudes la capa de lógica del negocio se encarga de enviar a través de el ORM Entity framework las peticiones.

En cada uno de los controladores tenemos una instancia de nuestra base datos para poder acceder a los datos necesarios mediante consultas.

### Establecer conexión con la base de datos

```
"ConnectionStrings": {
    "DefaultConnection": "server=localhost; database=db; user=root; password=Cc68594*"}

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder.EnableSensitiveDataLogging();
    var connectionString = Configuration.GetConnectionString("DefaultConnection");
    optionsBuilder.UseMySQL(connectionString, ServerVersion.AutoDetect(connectionString));
}
```

Instanciar nuestra base de datos

```
public class NameController : ControllerBase
{
    private readonly DataContext _context;

    public NameController(DataContext context)
    {
        _context = context;
    }
}
```

### Diagrama de la arquitectura N-Layer.



## **VI. Patrones de visualización y de datos:**

Por ejemplo, para cada uno de los siguientes puntos nos apoyaremos en la modelación de la funcionalidad película.

- Patrón de visualización “**MVC (Orientado a componentes)**”:

Al desarrollar el proyecto en ASP.NET Core, estamos empleando el patrón de presentación MVC, dado que este framework implementa la separación en tres capas que propone el patrón. Sin embargo, cuando decimos “Orientado a componentes” no nos referimos a un patrón de diseño específico, sino más bien a un enfoque general en el desarrollo de software que utiliza componentes como elementos fundamentales.

**View:** Se representa mediante las componentes de React. Esta biblioteca es la que se encarga de la lógica de renderizado en el navegador y la actualización de la interfaz de usuario.

El enfoque orientado a componentes nos da ventajas tales como la **reutilización**, un ejemplo es la componente *TextInput*, reutilizada en varios formularios.

```
function TextInput(props: Props){
  const [invalidFeedback, setInvalidFeedback] = useState("");
  const [tag, setTag] = useState("");

  const handleChange = (e: React.ChangeEvent) => {
    const newValue = (e.target as HTMLInputElement).value;

    // Control invalid feedback
    if (newValue.length === 0) {
      setInvalidFeedback("Campo obligatorio");
      setTag("is-invalid");
    } else setTag("");
    props.setValue(newValue);
  };

  return(
    <div className="form-group formgroup">
      <label htmlFor="titleInput">{props.name}</label>
      <input type="text" className={`form-control ${tag}`} id="titleInput" placeholder={props.placeholder}
      defaultValue={props.defaultValue} onChange={handleChange}/>
      <div className="invalid-feedback">{invalidFeedback}</div>
    </div>
  );
}
```

También nos brinda **mantenibilidad**, ya que, al estar separado en componentes, los cambios en una parte del sistema no deberían tener un impacto significativo en otras partes.

**Model:** En el servidor de ASP.NET Core se representa la lógica de negocio y los datos. Además, se tienen las clases de modelos que definen la estructura de los datos y la lógica asociada.

#### Modelo de nuestra base de datos PELÍCULA

```
public class Movie
{
    public Movie()
    {
        ActorsByFilms = new List<ActorByFilm>();
        GenresByFilms = new List<GenreByFilm>();
    }

    public int MovieId { get; set; }
    public string Title { get; set; }
    public int Year { get; set; }
    public string Country { get; set; }
    public string Director { get; set; }
    public int Duration { get; set; }

    public ICollection<ActorByFilm> ActorsByFilms { get; set; }
    public ICollection<GenreByFilm> GenresByFilms { get; set; }
    public ICollection<Likes> Likes { get; set; }
}
```

**Controller:** Los controladores se desarrollan en C# en la parte del servidor de ASP.NET Core y se encargan de manejar las diferentes peticiones o solicitudes HTTP que se hacen desde el navegador.

## Controlador de PELÍCULA

```
[ApiController]
public class MovieController : CRDController<Movie>
{
    private readonly DataContext _context;
    private readonly IMapper _mapper;

    public MovieController(DataContext context, IMapper mapper) : base(context)
    {
        _context = context;
        _mapper = mapper;
    }

    // operaciones CRD a realizar con Película
    [HttpGet]
    ...

    [HttpPost]
    ...

    [HttpDelete("{id}")]
    ...

    [HttpPut("{id}")]
    public async Task<IActionResult> UpdateMovie(int id, [FromBody] MovieInput updateMovie)
    {
        var movie = await _context.Movies.FindAsync(id);
        if (movie == null) { return NotFound(); }

        var deleteActors = await _context.ActorsByFilms
            .Where(m => m.MovieId == id).ToListAsync();

        _context.ActorsByFilms.RemoveRange(deleteActors);

        var deleteGenres = await _context.GenresByFilms
            .Where(m => m.MovieId == id).ToListAsync();

        _context.GenresByFilms.RemoveRange(deleteGenres);

        _mapper.Map(updateMovie, movie);
        await _context.SaveChangesAsync();
        return Ok();
    }
}
```



El uso de este patrón nos trae ventajas muy similares a las que brinda la arquitectura N-Layered, puesto que ofrece separación de preocupaciones, escalabilidad, y facilita la colaboración entre el equipo de trabajo.

- **Patrón de Diseño de Interfaz de Usuario Receptiva (Responsive UI).**

Aplicamos un diseño receptivo para garantizar que la interfaz de usuario sea accesible desde diferentes dispositivos y tamaños de pantalla, cumpliendo con el objetivo de que sea un sitio multiplataforma. Para ello utilizaremos CSS y técnicas de diseño responsivo para que las vistas web se adapten automáticamente a pantallas de diferentes tamaños, desde computadoras de escritorio hasta dispositivos móviles.

- **Patrones de Datos:**

- **Patrón Repository:** Utilizamos este patrón para separar la lógica de acceso a datos de la lógica de negocios, lo que facilita la interacción con la base de datos y asegura un código limpio y mantenible. Para cada entidad que representa una tabla en nuestra base de datos, hemos creado una clase de repositorio dedicada (controlador), la cual proporciona métodos para realizar operaciones comunes en esa entidad, como obtener, agregar, actualizar y eliminar registros. Los repositorios se comunicarán con la base de datos MySQL utilizando Entity Framework y LINQ para realizar consultas y operaciones CRUD.

Como las operaciones CRUD las realizamos para varias entidades decidimos crear una clase abstracta para el manejo de las operaciones CRD ya que Update seria específica para cada caso.

## Operaciones CRUD

```
namespace cineplus.CRDController;

public abstract class CRDController<T> : ControllerBase where T : class
{
    protected readonly DataContext _context;
    public CRDController(DataContext context)
    {
        _context = context;
    }

    public IQueryable<T> GetAll()
    {
        return _context.Set<T>();
    }

    public async Task Insert(T entity)
    {
        _context.Set<T>().Add(entity);
        await _context.SaveChangesAsync();
    }

    public async Task Delete(int id)
    {
        var entity = await _context.Set<T>().FindAsync(id);

        _context.Set<T>().Remove(entity);
        await _context.SaveChangesAsync();
    }
}
```

- **Patrón de Mapeo Objeto-Relacional (ORM):** La utilización de un ORM simplifica la manipulación de datos en la base de datos, reduciendo la necesidad de escribir consultas SQL directas. Nosotros utilizaremos Entity Framework, como ORM para mapear las entidades de la aplicación a las tablas de la base de datos MySQL. Esto facilita las operaciones CRUD y permite la manipulación de datos de manera más intuitiva.

- **Patrón de Transferencia de Datos(TO):** Es un patrón de diseño que se utiliza para transferir un conjunto de datos entre sistemas de manera eficiente y encapsulada. Este patrón se centra en la transferencia de datos y no tiene comportamiento, ya que su objetivo principal es representar una estructura de datos que se puede mover fácilmente entre las capas. En este contexto, la estructura de un objeto TO está compuesta por atributos que representan los datos a transferir. Esto resulta fundamental al transferir datos entre el cliente y el servidor en ambas direcciones.

En el caso específico de manejar datos relacionados con Películas, implementamos dos clases adicionales para gestionar los datos que recibimos del cliente y los que enviamos al cliente. En este escenario, empleamos los "DTO" (Objetos de Transferencia de Datos), que podemos considerar como una implementación específica del patrón TO.

Un ejemplo concreto de esta implementación se encuentra en el uso de AutoMapper en nuestro código. Hemos configurado un perfil de AutoMapper que define cómo mapear propiedades de un objeto de tipo `Movie` a otro objeto de tipo `MovieGet`. Este enfoque nos permite manejar de manera eficiente la transferencia de datos y garantizar que solo se envíen o reciban las propiedades necesarias, facilitando así la comunicación entre las capas de nuestra aplicación.

#### Configuración para el Mapeo entre Objetos

```
var mappingConfig = new MapperConfiguration(cfg =>
{
    cfg.AddProfile<MappingMovieProgramming>();
    ... // Añadir todas las clases encargadas del mapeo
});

var mapper = mappingConfig.CreateMapper();
builder.Services.AddSingleton(mapper);
```

```
public class MovieInput
{
    public int id { get; set; } = 0;
    public string title { get; set; }
    public int year { get; set; }
    public string country { get; set; }
    public string director { get; set; }
    public int duration { get; set; }
    public List<int> actors { get; set; }
    public List<int> genres { get; set; }
}

public class MovieGet
{
    public int id { get; set; }
    public string title { get; set; }
    public int year { get; set; }
    public string country { get; set; }
    public string director { get; set; }
    public int duration { get; set; }
    public List<ActorDto> actors { get; set; }
    public List<GenreDto> genres { get; set; }
}
```

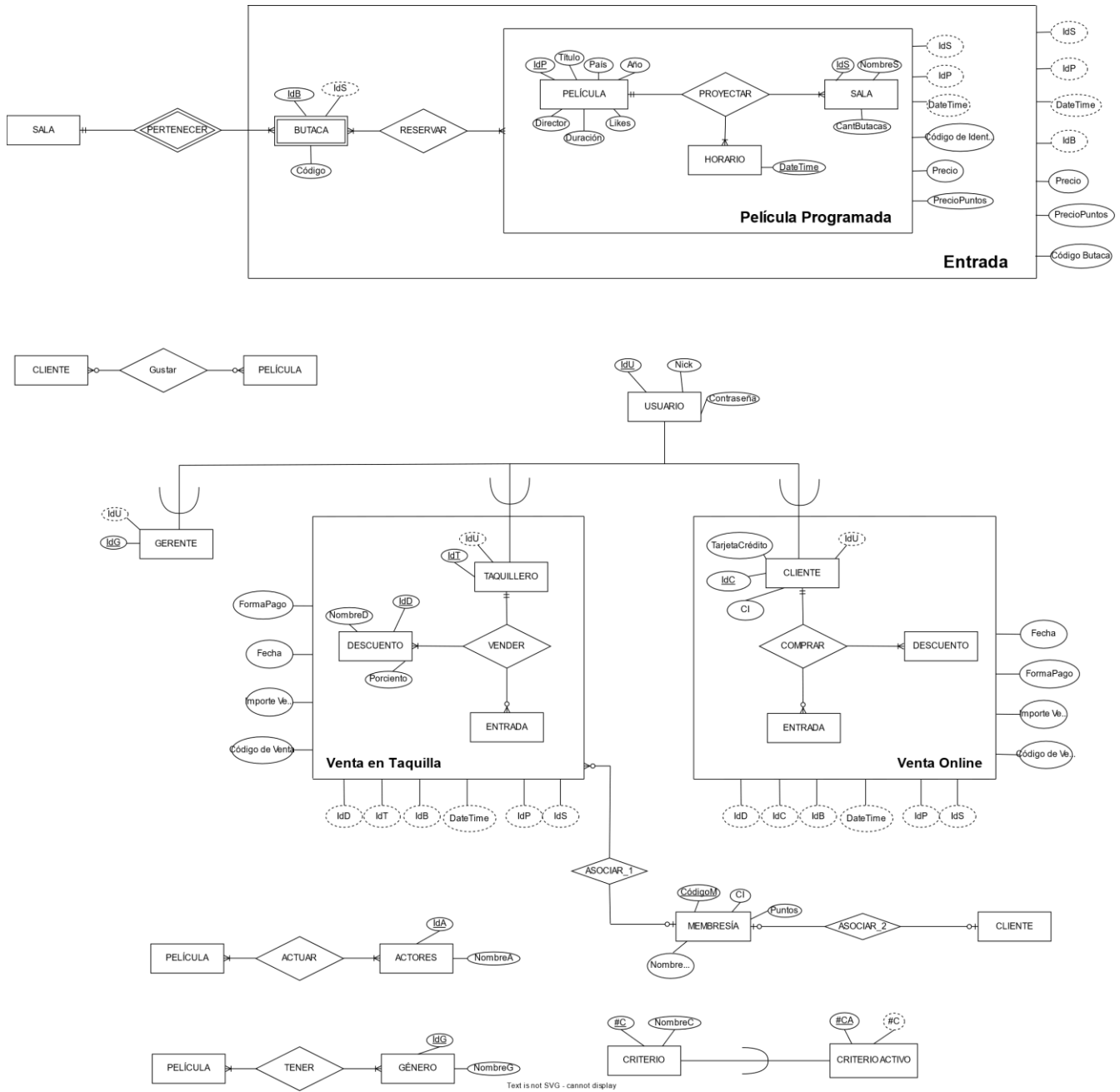
## Mapeo sobre la entidad PELÍCULA

```
CreateMap<MovieInput, Movie>()
    .ForMember(dest => dest.MovieId, opt => opt.Ignore())
    .ForMember(dest => dest.Title, opt => opt.MapFrom(src => src.title))
    .ForMember(dest => dest.Year, opt => opt.MapFrom(src => src.year))
    .ForMember(dest => dest.Country, opt => opt.MapFrom(src => src.country))
    .ForMember(dest => dest.Director, opt => opt.MapFrom(src => src.director))
    .ForMember(dest => dest.Duration, opt => opt.MapFrom(src => src.duration))
    .ForMember(dest => dest.actorsByFilms, opt => opt.Ignore())
    .ForMember(dest => dest.GenresByFilms, opt => opt.Ignore())
    .AfterMap((src, dest) =>
    {
        // Asigna los actores usando los IDs proporcionados en MovieInput
        dest.actorsByFilms = src.actors.Select(actorId => new ActorByFilm
            { ActorId = actorId, MovieId = dest.MovieId }).ToList();
    })
    .AfterMap((src, dest) =>
    {
        // Asigna los actores usando los IDs proporcionados en MovieInput
        dest.GenresByFilms = src.genres.Select(genreId => new GenreByFilm
            { GenreId = genreId, MovieId = dest.MovieId }).ToList();
    });

CreateMap<Movie, MovieGet>()
    .ForMember(dest => dest.id, opt => opt.MapFrom(src => src.MovieId))
    .ForMember(dest => dest.title, opt => opt.MapFrom(src => src.Title))
    .ForMember(dest => dest.year, opt => opt.MapFrom(src => src.Year))
    .ForMember(dest => dest.country, opt => opt.MapFrom(src => src.Country))
    .ForMember(dest => dest.director, opt => opt.MapFrom(src => src.Director))
    .ForMember(dest => dest.duration, opt => opt.MapFrom(src => src.Duration))
    .ForMember(dest => dest.actors, opt => opt.MapFrom(src =>
        src.actorsByFilms.Select(a => new ActorDto
            { id = a.Actor.ActorId, name = a.Actor.Name })))
    .ForMember(dest => dest.genres, opt => opt.MapFrom(src =>
        src.GenresByFilms.Select(g => new GenreDto
            { id = g.Genre.GenreId, name = g.Genre.Name })));
```

Este enfoque nos ayuda a mantener una estructura clara en la transferencia de datos y contribuye a la eficiencia y claridad en la comunicación entre el cliente y el servidor.

## VII. Modelo de Datos:



## VIII. Diccionario de datos

(Tabla) ActiveCriterion:

- (Columna) ActiveCriterionID: Entero
- (Columna) CriterionID: Entero {Llave foránea a la tabla Criterion}

(Tabla) Actor:

- (Columna) ActorId: Entero
- (Columna) Name: String {No se manejan restricciones con respecto al tamaño, puesto que se maneja en las capas anteriores}

(Tabla) ActorByFilm:

- (Columna) ActorID: Entero {Llave foránea a la tabla Actor}
- (Columna) MovieID: Entero {Llave foránea a la tabla Movie}

(Tabla) BoxOfficeSales:

- (Columna) TicketSellerID: Entero {Llave foránea a la tabla TicketSeller}
- (Columna) RoomID: Entero
- (Columna) MovieID: Entero
- (Columna) DateTimeID: DateTime
- (Columna) SeatID: Entero

{RoomID,MovieID,DateTimeID,SeatID: Llave foránea a la tabla Ticket}

- (Columna) DiscountID: Entero {Llave foránea a la tabla Discounts}
- (Columna) DateOfPurchase: DateTime
- (Columna) FinalPrice: Double
- (Columna) MemberCode: String {Puede ser null este valor, dado que puede que el cliente no se haya registrado}

(Tabla) Client:

- (Columna) ClientID: Entero
- (Columna) DNI: String
- (Columna) CreditCard: String
- (Columna) UserID: Entero {Llave foránea a la tabla User}

(Tabla) Criterion:

- (Columna) CriterionID: Entero
- (Columna) Name: String {Debe ser distinto de null}

(Tabla) Discount:

- (Columna) DiscountID: Entero
- (Columna) Concept: String {Debe Ser distinto de null}

- (Columna) Percent: Float
- (Tabla) Genre:
  - (Columna) GenreID: Entero
  - (Columna) Name: String
- (Tabla) GenreByFilm:
  - (Columna) GenreID: Entero {Llave foránea a la tabla Genre}
  - (Columna) MovieID: Entero {Llave foránea a la tabla Movie}
- (Tabla) JWTSettings:
  - (Columna) SecurityKey: String  
{Esta tabla se usa para guardar la llave secreta por si se desea cambiar}
- (Tabla) Likes:
  - (Columna) ClientID: Entero {Llave foránea a la tabla Client}
  - (Columna) MovieID: Entero {Llave foránea a la tabla Movie}
- (Tabla) Gerente:
  - (Columna) GerenteID: Entero
  - (Columna) UserID: Entero {Llave foránea a la tabla User}
- (Tabla) Membership:
  - (Columna) MembershipCode: String
  - (Columna) MemberDNI: String
  - (Columna) Points: Entero
  - (Columna) FullName: String
  - (Columna) ClientID: Entero {Llave foránea a la tabla Client, en caso de que sea un miembro de taquilla es null}
- (Tabla) Movie:
  - (Columna) MovieID: Entero
  - (Columna) Title: String
  - (Columna) Year: Entero
  - (Columna) Country: String
  - (Columna) Director: String
  - (Columna) Duration: Entero
- (Tabla) MovieProgramming:
  - (Columna) Identifier: Entero
  - (Columna) RoomID: Entero {Llave foránea a la tabla Room}
  - (Columna) MovieID: Entero {Llave foránea a la tabla Movie}
  - (Columna) DateTimeID: DateTime



- (Columna) Price: Double
- (Columna) PricePoints: Entero

(Tabla) OnlineSales:

- (Columna) ClientID: Entero {Llave foránea a la tabla Client}
- (Columna) RoomID: Entero
- (Columna) MovieID: Entero
- (Columna) DateTimeID: DateTime
- (Columna) SeatID: Entero

{RoomID,MovieID,DateTimeID,SeatID: Llave foránea a la tabla Ticket}

- (Columna) DiscountID: Entero {Llave foránea a la tabla Discount}
- (Columna) DateOfPurchase: DateTime
- (Columna) Transfer: Bool
- (Columna) FinalPrice: Double
- (Columna) SaleIdentifier: Guid

(Tabla) Room:

- (Columna) RoomID: Entero
- (Columna) Name: String
- (Columna) SeatsCount: Entero

(Tabla) Schedule:

- (Columna) DateTime: DateTime

(Tabla) Seat:

- (Columna) SeatID: Entero
- (Columna) RoomID: Entero {Llave foránea a la tabla Room}
- (Columna) Code: String

(Tabla) Ticket:

- (Columna) RoomID: Entero
- (Columna) MovieID: Entero
- (Columna) DateTimeID: DateTime

{RoomID,MovieID,DateTimeID: Llave foránea a la tabla MovieProgramming}

- (Columna) SeatID: Entero {Llave foránea a la tabla Seat}
- (Columna) Price: Double
- (Columna) PricePoints: Entero
- (Columna) Code: String

(Tabla) TicketSeller:

- (Columna) TicketSellerID: Entero

- (Columna) UserID: Entero {Llave foránea a la tabla User}

(Tabla) User:

- (Columna) UserID: Entero
- (Columna) Nick: String
- (Columna) Password: String
- (Columna) Salt: String

## IX. Esquema de clases definidas

Para el trabajo con la Base de Datos, utilizamos Entity Framework como ORM, gracias a las facilidades que brinda, entre estas se encuentra el uso de propiedades de navegación para la interrelación de clases, algunos ejemplos de estas propiedades de navegación son (Ver Fig. 9.1):

```
public class Client
{
    13 references
    public int ClientId { get; set; }

    6 references
    public string DNI { get; set; }

    4 references
    public string CreditCard { get; set; }

    13 references
    public int UserId { get; set; } // Propiedad para la clave foránea
    4 references
    public virtual User User { get; set; }
    2 references
    public virtual Membership Membership { get; set; }

    1 reference
    public ICollection<OnlineSales> Sales { get; set; }
    1 reference
    public ICollection<Likes> Likes { get; set; }
}
```

Fig. 9.1: Representación de la tabla Client en C#

Las propiedades de tipo:

- User
- Membership
- ICollection<OnlineSales>
- ICollection<Likes>

Las cuales se usan para modelar la relación que tiene cliente con dichas tablas:

- User y Membership indican que:
  - Por cada elemento de la tabla Client corresponde uno de la tabla User
  - Por cada elemento de la tabla Client corresponde uno de la tabla Member
- Luego los ICollection indican que:
  - Por cada elemento de la tabla Client corresponden muchos en la tabla OnlineSales
  - Por cada elemento de la tabla Client corresponden muchos en la tabla Likes

Veamos cómo se evidencia esta relación en las tablas User(Fig. 9.2) y Likes (Fig.9.3):

```
public class User
{
    10 references
    public int UserId { get; set; }
    10 references
    public string Nick { get; set; }
    7 references
    public string Password { get; set; }
    7 references
    public string Salt { get; set; }

    2 references
    public virtual Client Client { get; set; }
    2 references
    public virtual Manager Manager { get; set; }
    2 references
    public virtual TicketSeller TicketSeller { get; set; }
}
```

Fig. 9.2: Representación de clase User en C#

```
public class Likes
{
    4 references
    public int ClientId { get; set; }
    5 references
    public int MovieId { get; set; }
    1 reference
    public Client Client { get; set; }
    1 reference
    public Movie Movie { get; set; }
}
```

Fig. 9.3: Representación de la clase Likes en C#

Como podemos apreciar en ambos casos existe la propiedad de navegación hacia la tabla Client, lo que permite que la navegación sea bidireccional. En el caso de la tabla Likes (Fig. 9.3) solo guardamos un cliente dado que uno puede dar un like por una sola película.

También se pueden modelar estas relaciones desde la clase OnModelCreating (Fig. 9.4 y Fig.9.5), que provee el DbContext de EntityFramework, para especificar bien el mapeo con el que se va a crear la base de datos.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    // ----- Users -----

    modelBuilder.Entity<User>()
        .HasOne(u => u.Client) // Relación opcional: un usuario puede estar asociado con un cliente o no.
        .WithOne(c => c.User) // Configura la relación inversa en la clase Client.
        .HasForeignKey<Client>(c => c.UserId) // Clave foránea en la clase Client
        .IsRequired(false);
}
```

Fig. 9.4: Modelación de la tabla User en el context

```
modelBuilder.Entity<Likes>()
    .HasKey(x => new { x.ClientId, x.MovieId });

modelBuilder.Entity<Likes>()
    .HasOne(c => c.Client)
    .WithMany(x => x.Likes)
    .HasForeignKey(c => c.ClientId);

modelBuilder.Entity<Likes>()
    .HasOne(m => m.Movie)
    .WithMany(x => x.Likes)
    .HasForeignKey(m => m.MovieId);
```

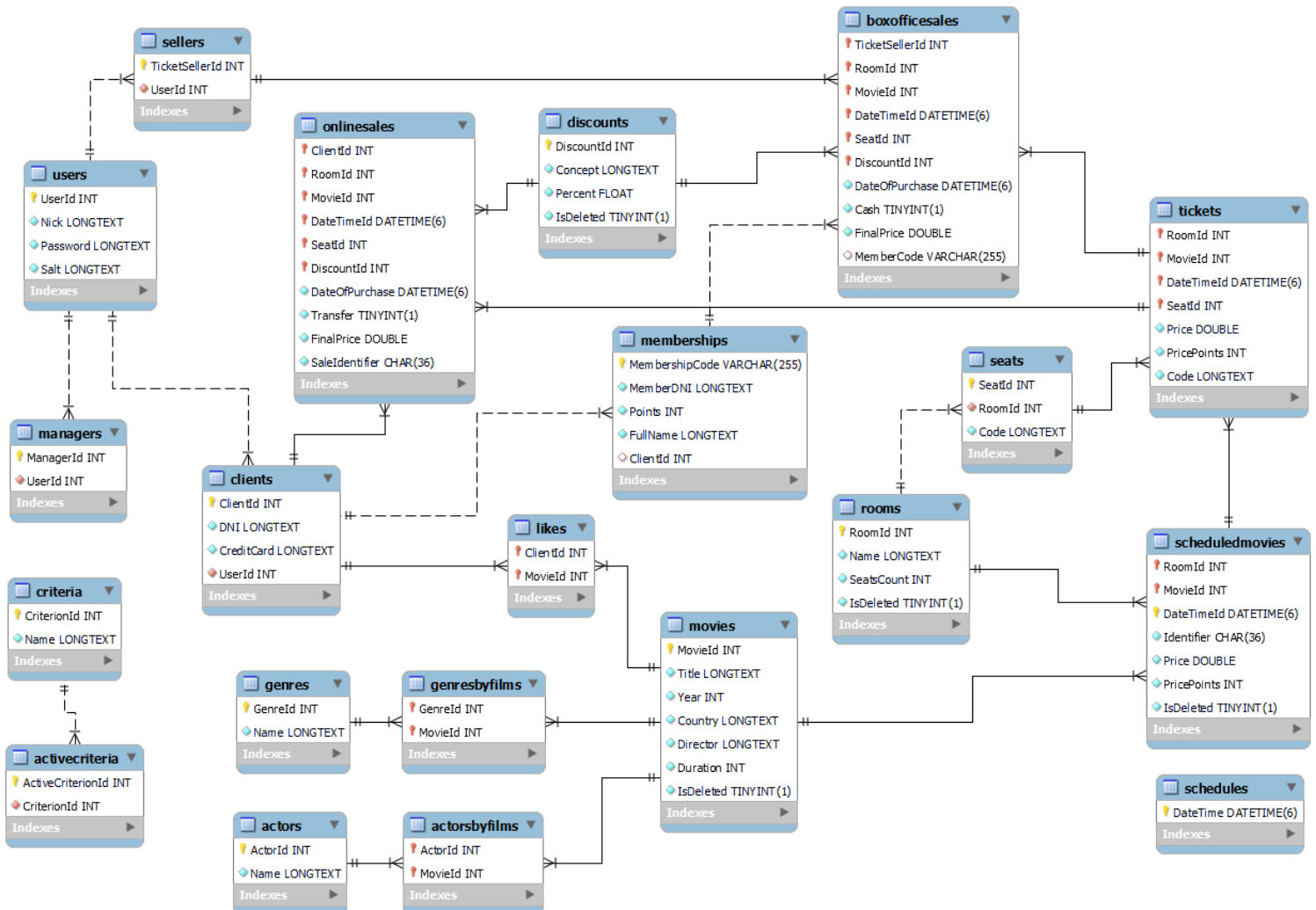
Fig. 9.5: Modelación de la tabla Like en el context

En la modelación de usuario (fig. 9.4) se especifica que le corresponde a un único cliente, o no, esto sucede porque un usuario puede ser también gerente.

En la modelación de la tabla Like se especifica que la llave, es una llave compuesta entre los id de los clientes y películas. Además, se especifica la relación entre los clientes con el ClientId como llave foránea.

Luego de esta manera se crean todas las relaciones entre las clases (Tablas) de nuestro proyecto. La selección de Entity Framework como ORM se debe a la gran versatilidad del mismo en el momento de modelar las clases, las interrelaciones entre ellas y las realizar consultas en la base de datos.

### Esquema de la base de datos



## X. Opciones del sistema

Nuestro sistema cumple las especificaciones y requerimientos planteados en la sección **Requerimientos Específicos** (página 4):

Permite el registro de clientes (Fig. 10.1) en la aplicación siempre y cuando especifiquen los campos mostrados

Luego, una vez registrados, los usuarios son capaces de iniciar sesión en la aplicación (Fig. 10.2), y serán dirigidos a la página de cliente la cual le permite otras funcionalidades, todas relacionadas con los clientes, más adelante veremos las relacionadas con el gerente y con los taquilleros

Registrarse

Nombre de Usuario

Introduce tu usuario

Campo obligatorio

Contraseña

Elige tu contraseña

Campo obligatorio

La contraseña debe tener entre 6-20 caracteres

Confirma tu contraseña

Documento de identidad

Introduce tu Documento de identidad

Tarjeta de crédito

Introduce tu número de tarjeta

Registrarse

Fig. 10.1: Registrarse como cliente

Iniciar sesión

Nombre de Usuario

Introduce tu usuario

Campo obligatorio

Contraseña

Introduce tu contraseña

Campo obligatorio

[¿Ya estás registrado?](#)

Iniciar sesión

Fig. 10.2: Iniciar sesión

Los Clientes registrados pueden ver todas las películas disponibles, separadas por criterios (cuyos criterios los maneja el gerente) (Fig. 10.3), si accede a la película, podrá observar las programaciones que están disponibles para dicha película, para luego comprar un ticket para esa programación (Fig. 10.4)

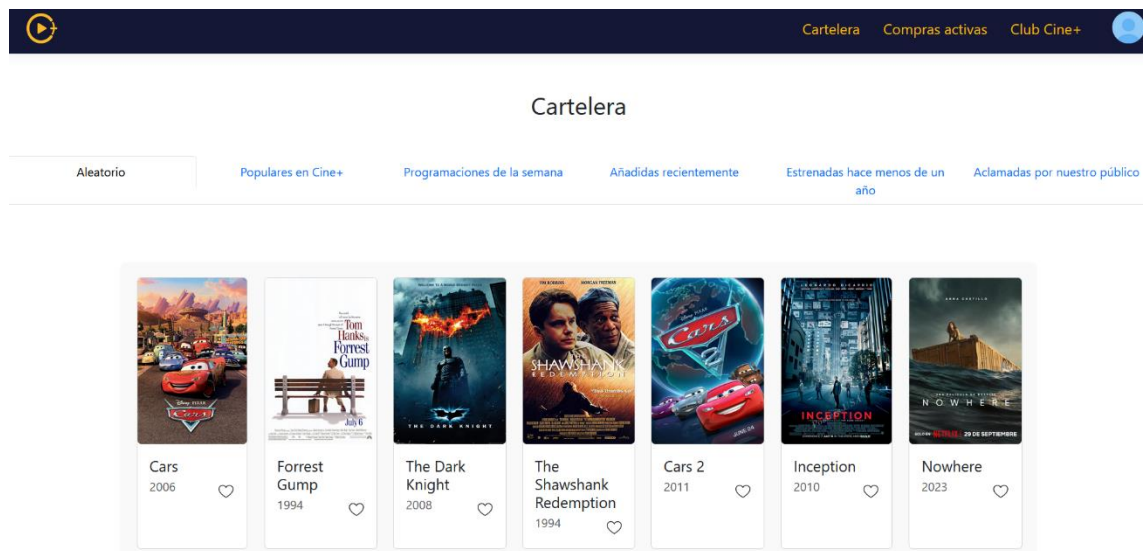


Fig. 10.3: Cartelera con películas disponibles

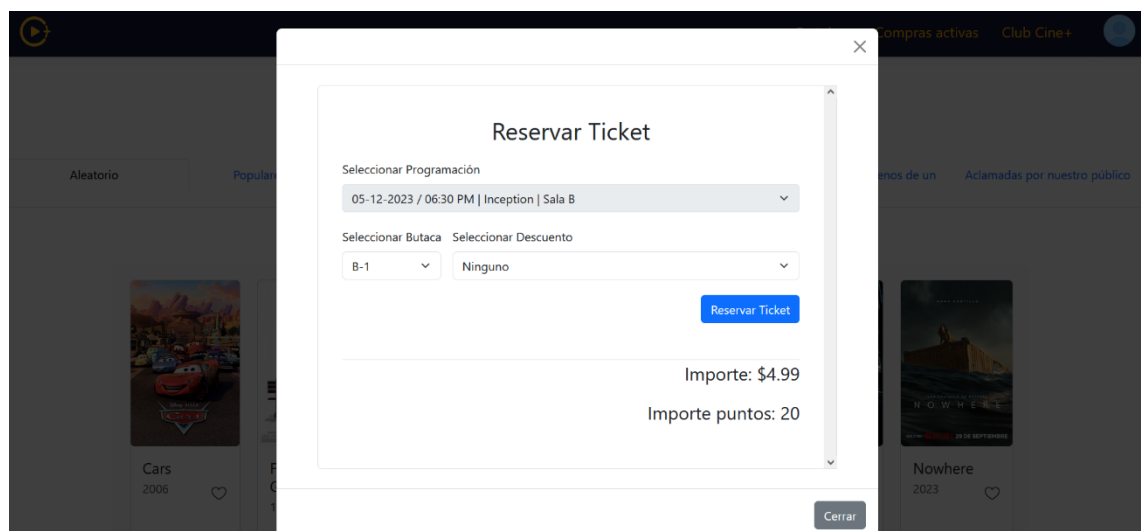


Fig. 10.4: Reservación de un ticket

Todos aquellos clientes que deseen pueden registrarse como socios, usando la la página de Club Cine+, o desde la propia taquilla.

Si un cliente lo desea puede cancelar todas sus compras (Fig. 10.5)



Pelicula	Horario	Sala	Butaca	Comprobante
The Shawshank Redemption	05-12-2023 / 09:30 PM	Sala A	A-1	<a href="#">Imprimir</a> <a href="#">Cancelar</a>
Cars	03-12-2023 / 10:43 PM	Sala VIP	VIP-1	<a href="#">Imprimir</a> <a href="#">Cancelar</a>

Fig. 10.5: Historial de reservaciones

Al inicio, los gerente y taquilleros tendrán una cuenta asociada, por lo tanto, al iniciar sesión con estas cuentas tienen acceso a sus páginas.

Si inicia sesión el gerente, se cargará su página la cual carga por defecto las estadísticas de ventas de las entradas (Fig. 10.6).

El gerente podrá realizar otras funciones como son añadir películas, realizar la programación de los horarios, añadir salas y definir la cantidad de butacas de las mismas, añadir actores o géneros.

El gerente también puede definir y administrar los descuentos que pueden aplicarse a los clientes, y manejar los criterios de sugerencia de las películas que se muestran. Todas estas opciones tienen la facilidad de edición para que puedan ser modificadas. (Fig. 10.7).





Fig. 10.6: Estadísticas y opciones del gerente

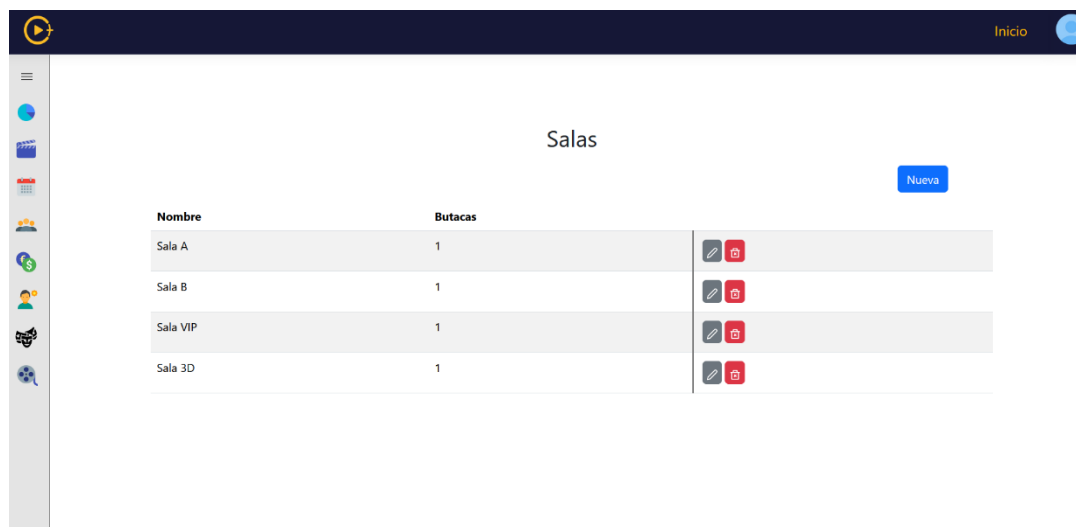


Fig. 10.7: Puede editar cada una de las opciones

En caso de que el que inicie sesión es el vendedor, este tendrá acceso a las películas programadas y podrá asociar clientes como miembros de Cine+ o verificar si un cliente es socio (Fig. 10.8).

Fig. 10.8: Opciones del vendedor

## XI. Salidas del sistema

Nuestro sistema posee un conjunto de salidas, la mayoría son mensajes de confirmación de que la solicitud ha Sido manejada, sin embargo, posee otro conjunto de salidas en las cuales se muestran los datos adquiridos, por tanto, nos centraremos más en las salidas que poseen un conjunto de datos.

Si accedemos como cliente, tendremos varias salidas, al comprar un ticket, se puede mostrar este ticket en la sección (Fig. 11.1), además el cliente puede solicitar su ticket para exportarlo como PDF, en caso de que desee imprimirlo (Fig. 11.2).

### Historial de compras activas

Pelicula	Horario	Sala	Butaca	Comprobante	
Conducta	08-12-2023 / 06:00 PM	Sala Plus+	PLUS-1	 Imprimir	 Cancelar
Fresa y Chocolate	11-12-2023 / 09:00 PM	Sala Clásica	CLAS-1	 Imprimir	 Cancelar
Cars	08-12-2023 / 06:00 PM	Sala 3D	3D-1	 Imprimir	 Cancelar

Fig. 11.6: Historial de compras para un cliente



Fig. 11.2: Comprobante de venta

El vendedor tendrá como salida en común al cliente la exportación del ticket. Además, podrá consultar si el cliente que está llegando es miembro o no, y como salida se le muestran los datos de este cliente (Fig. 11.3)

## Código de Membresía

Nombre: Kevin

Código de Miembro: CDCE7FFD

Puntos disponibles: 20

Cerrar

Fig. 11.3: Datos del miembro

El manager en cambio posee más salidas, dado que él puede manejar muchos más datos, como las películas que posee, las programaciones, luego si añade o modifica, se le mostrará en una tabla los cambios realizados (Fig. 11.4). Además, puede consultar estadísticas de venta, dónde se le mostrarán el usuario que más

ha comprado tickets, o si las películas cubanas son más vendidas que las películas extranjeras, estos datos se mostrarán cómo un gráfico de pastel, y una tabla para las películas más vendidas dado un filtro (Fig. 11.5)

### Películas

Nueva











Título	Año	País	Director	Actores	Géneros	Duración	
Inception	2010	USA	Christopher Nolan	Leonardo DiCaprio Tom Hardy Joseph Gordon-Levitt	Ciencia Ficción	148 min	 
The Shawshank Redemption	1994	USA	Frank Darabont	Leonardo DiCaprio Morgan Freeman	Drama	142 min	 
The Dark Knight	2008	USA	Christopher Nolan	Christian Bale Heath Ledger	Acción	152 min	 
Forrest Gump	1994	USA	Robert Zemeckis	Tom Hanks	Romance Aventura	142 min	 
Nowhere	2023	España	Albert Pintó	Anna Castillo	Drama Suspense	129 min	 

Fig. 11.4: Películas guardadas en el sistema



Fig. 11.5: Gráficos y tablas de las estadísticas

## XII. Análisis de modificaciones

El modelo inicial presentado ha experimentado algunas modificaciones, las cuales fueron necesarias para optimizar la funcionalidad de nuestro proyecto. A continuación, detallamos los cambios realizados:

- Tipo y representación de la llave primaria en la Tabla Horario:

Inicialmente, el tipo de dato en C# utilizado para representar fecha y hora en la tabla 'horario' era mediante string para cada una de esas propiedades. Decidimos cambiar la clave de la entidad al tipo 'DateTime', que es el tipo de representación más adecuado. Ahora, esta modificación se refleja en la clave primaria de 'Pelicula Programada', utilizando 'DateTime' en lugar de 'fecha' y 'horario'.

- Entidad Débil de Sala y Código de Butaca:

Transformamos la entidad 'butaca' en una entidad débil de 'sala' mediante una interrelación, permitiéndonos conocer las butacas específicas de cada sala. Adicionalmente, se incorporó una propiedad denominada 'código' para representar el número de butaca correspondiente. Este código se agregó a la entidad 'ticket', ya que estos tickets almacenan información crucial para la programación, incluyendo la relación de butaca asignada, necesaria para la presentación al cliente.

- Propiedad de Tipo GUID para Ventas Online y en Taquilla:

Se introdujo una propiedad de tipo GUID tanto para las ventas online como para las ventas en taquilla. Esto permite la generación de un código único con la función de identificar las ventas, facilitando su posterior visualización en el comprobante de compra que el cliente puede exportar.

- Propiedad 'IsDeleted' para Entidades Claves Foráneas:

Con el objetivo de evitar la pérdida de información y resultados incorrectos al realizar estadísticas, implementamos la propiedad 'IsDeleted' en las entidades 'Película', 'Sala', 'Película Programada' y 'Descuento'. Esta propiedad indica si la instancia de una de estas tablas ha sido eliminada por el gerente. En caso de que la instancia esté asignada como clave foránea en alguna de sus tablas relacionadas, no se elimina completamente de la base de datos, sino únicamente de cara al cliente.

- Tabla 'Likes' para Registrar Relación entre Películas y Clientes:

Se creó la tabla 'likes' para reflejar la interrelación entre películas y clientes, permitiendo así llevar un historial de las películas más gustadas por los clientes. Este recurso se ideó con la finalidad de proporcionar al gerente una perspectiva adicional al seleccionar los criterios para sugerir películas.