# PRÁCTICA
# DE
# PROGRAMACIÓN AVANZADA

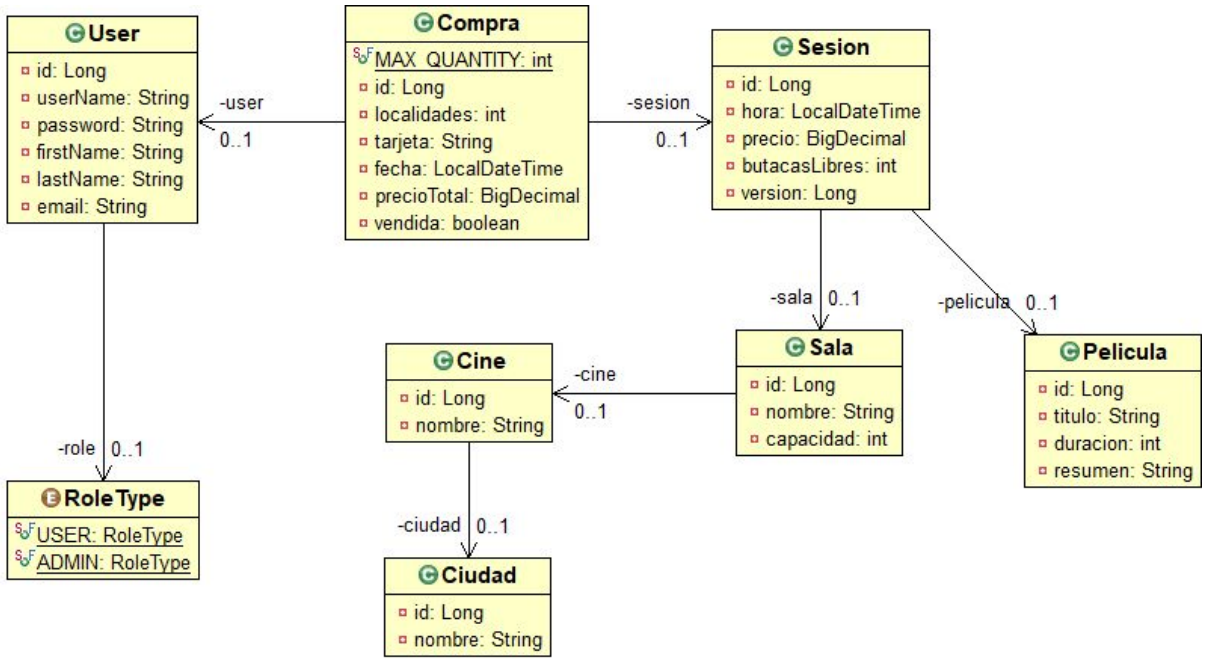
Ricardo Sánchez Arias - ricardo.sanchez1@udc.es

Laura Lestón Otero - laura.leston1@udc.es

Beltrán Aceves Gil - beltran.aceves@udc.es

# 1. Backend

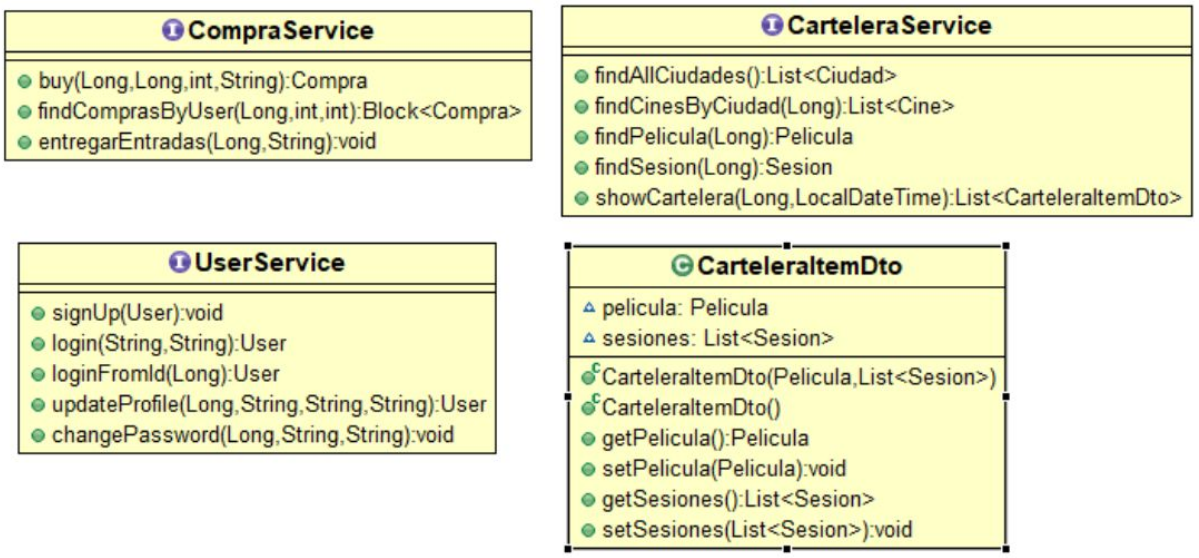
## 1.1 Capa de acceso a datos




- Aspectos a remarcar:

Tal y como están establecidas las relaciones, podemos navegar por toda la información necesaria dado un objeto Compra.
Los objetos de la clase Compra cuentan con el atributo vendida para facilitar el caso de uso **Entrega de las entradas de una compra.** Los objetos de la clase Sesion tienen el atributo version para hacer uso de la anotación @Version de JPA, que permite detectar modificaciones concurrentes y evita la pérdida de información durante actualizaciones.

# 1.2 Capa lógica de negocio




- Aspectos a remarcar:

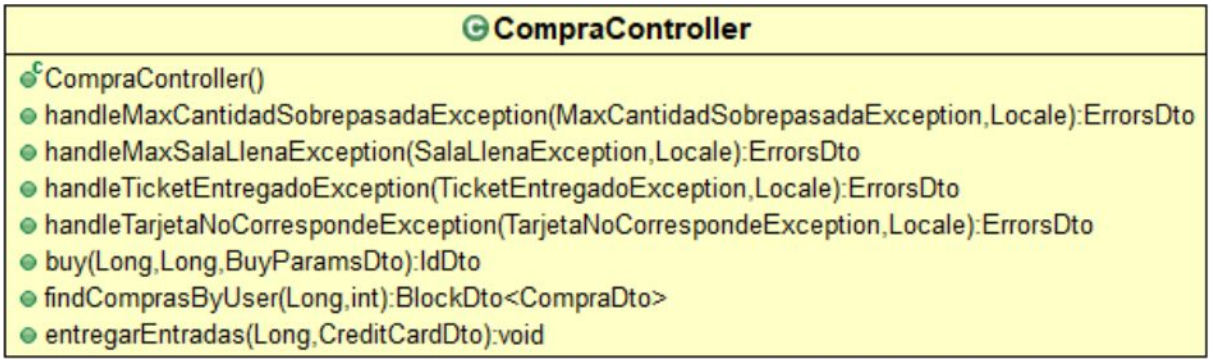
Cada fachada de servicio incluye únicamente funciones que representan casos de uso (login, showCartelera, etc), así como las Querys de JPA necesarias.

Creamos CarteleraItemDto para convertir la información sacada de las bases de datos en un estructura más apropiada para que la capa de servicio rest le devuelva al frontend.

# 1.3 Capa servicios REST

**Funciones:**

- **Compra Controller:**




**buy:**

**Firma:**

```
@PostMapping("sesiones/{sesionId}/buy")
public IdDto buy(
        @RequestAttribute Long userId, @PathVariable Long sesionId, @Validated
        @RequestBody BuyParamsDto params)
        throws InstanceNotFoundException, SalaLlenaException,
        MaxCantidadSobrepasadaException, PeliculaEmpezadaException;
```

**URL:** /sesiones/{sesionId}/buy

**Petición:** POST

**DTO input:** BuyParamsDto

**Códigos de respuesta HTTP:** 201 Created, 400 Bad Request, 404 Not Found

**findComprasByUser:**

**Firma:**

```
@GetMapping("historial")
public BlockDto<CompraDto> findComprasByUser(
        @RequestAttribute Long userId,
        @RequestParam(defaultValue = "0") int page);
```

**URL:** /historial

**Petición:** GET

**DTO output:** BlockDto //Añadirlo arriba

**Códigos de respuesta HTTP:** 200 Ok, 400 Bad Request, 404 Not Found

**entregarEntradas:**

**Firma:**

```
@PostMapping("{compraId}/entregarTicket")
    public BooleanDto entregarEntradas(
            @PathVariable Long compraId, @RequestBody CreditCardDto creditCard)
            throws InstanceNotFoundException, PeliculaEmpezadaException,
            TicketEntregadoException, TarjetaNoCorrespondeException;
```

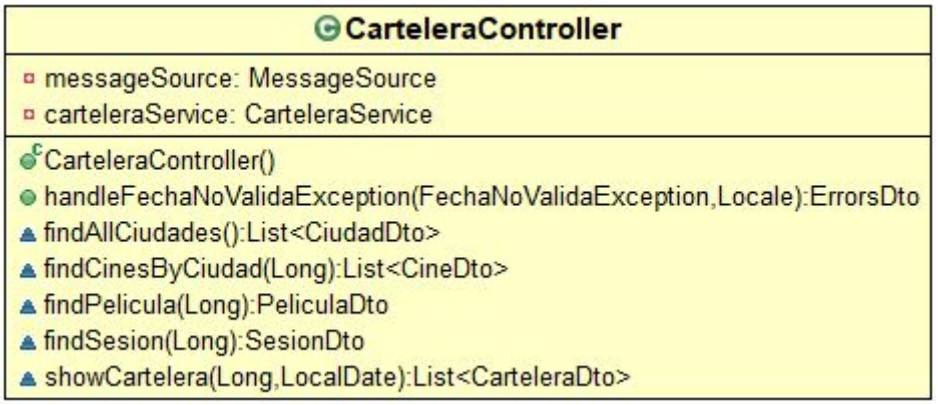
**URL:** /{compraId}/entregarTicket

**Petición:** POST

**DTO input:** CreditCardDto

**Códigos de respuesta HTTP:** 204 No Content, 400 Bad Request, 404 Not Found

- **Cartelera Controller:**




**findAllCiudades:**

**Firma:**

```
@GetMapping("/ciudades")
    List<CiudadDto> findAllCiudades();
```

**URL:** /ciudades

**Petición:** GET

**DTO output:** List<CiudadDto>

**Códigos de respuesta HTTP:** 200 Ok

**findCinesByCiudad:**

**Firma:**

```
@GetMapping("/ciudades/{id}/cines")
    List<CineDto> findCinesByCiudad(@PathVariable Long id);
```

**URL:** /ciudades/{id}/cines

**Petición:** GET

**DTO input:** List<CineDto>

**Códigos de respuesta HTTP:** 200 Ok

**findPelicula:**

**Firma:**

```
@GetMapping("/peliculas/{id}")
PeliculaDto findPelicula(
        @PathVariable Long id)
        throws InstanceNotFoundException;
```

**URL:** /peliculas/{id}

**Petición:** GET

**DTO output:** PeliculaDto

**Códigos de respuesta HTTP:** 200 Ok, 404 Not Found

**findSesion:**

**Firma:**

```
@GetMapping("/sesiones/{id}")
SesionDto findSesion(
        @PathVariable Long id)
        throws InstanceNotFoundException, PeliculaEmpezadaException;
```

**URL:** /sesiones/{id}

**Petición:** GET

**DTO output:** SesionDto

**Códigos de respuesta HTTP:** 200 Ok, 403 Forbidden, 404 Not Found

**showCartelera:**

**Firma:**

```
@GetMapping("/cines/{cineId}")
List<CarteleraDto> showCartelera(
        @PathVariable Long cineId, @RequestParam(required=false)
        DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate date)
        throws FechaNoValidaException;
```

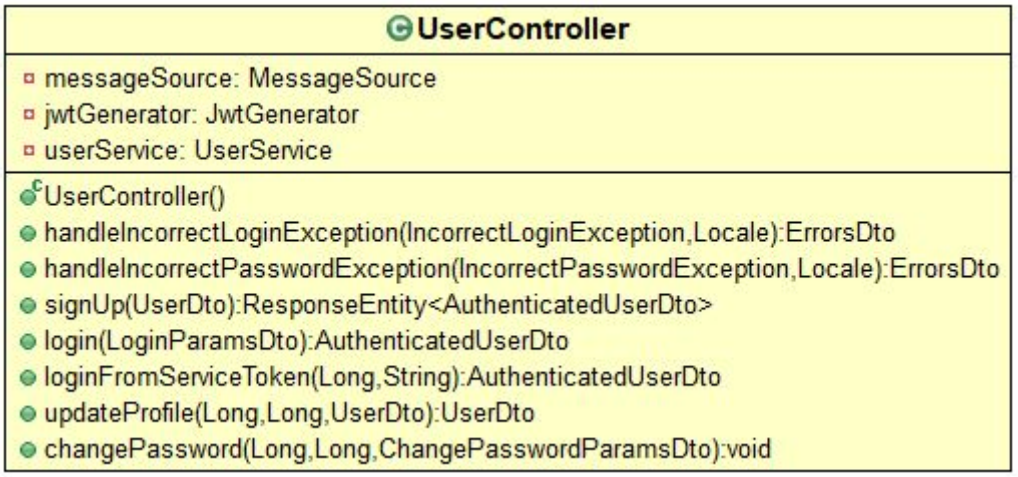
**URL:** /cines/{cineId}

**Petición:** GET

**DTO output:** List<CarteleraDto>

**Códigos de respuesta HTTP:** 200 Ok, 403 Forbidden

- **User Controller:**




**signUp**

**Firma:**

```
@PostMapping("/signUp")
public ResponseEntity<AuthenticatedUserDto> signUp(
        @Validated({ UserDto.AllValidations.class })
        @RequestBody UserDto userDto)
        throws DuplicateInstanceException;
```

**URL:** /signUp

**Petición: POST**

**DTO input:** UserDto

**Códigos de respuesta HTTP:** 201 Created, 400 Bad Request

**login**

**Firma:**

```
@PostMapping("/login")
public AuthenticatedUserDto login(
    @Validated @RequestBody LoginParamsDto params)
    throws IncorrectLoginException;
```

**URL:** /login

**Petición:** POST

**DTO input: LoginParamsDto**

**Códigos de respuesta HTTP:** 200 Ok, 400 Bad Request, 404 Not Found

**loginFromServiceToken**

**Firma:**

```
@PostMapping("/loginFromServiceToken")
	public AuthenticatedUserDto loginFromServiceToken(
		@RequestAttribute Long userId,
		@RequestAttribute String serviceToken)
		throws InstanceNotFoundException ;
```

**URL:** /loginFromServiceToken

**Petición:** POST

**DTO input:** none

**Códigos de respuesta HTTP:** 200 Ok, 400 Bad Request, 404 Not Found

**updateProfile**

**Firma:**

```
@PutMapping("/{id}")
	public UserDto updateProfile(
		@RequestAttribute Long userId, @PathVariable Long id,
		@Validated({ UserDto.UpdateValidations.class })
		@RequestBody UserDto userDto)
		throws InstanceNotFoundException, PermissionException;
```

**URL:** /{id}

**Petición:** PUT

**DTO input:** UserDto

**Códigos de respuesta HTTP:** 400 Bad Request, 404 Not Found, 204 No Content

**changePassword**

**Firma:**

```
@PostMapping("/{id}/changePassword")
@ResponseStatus(HttpStatus.NO_CONTENT)
	public void changePassword(
		@RequestAttribute Long userId, @PathVariable Long id,
		@Validated @RequestBody ChangePasswordParamsDto params)
		throws PermissionException, InstanceNotFoundException,
		IncorrectPasswordException;
```

**URL:** /{id}/changePassword

**Petición:** POST

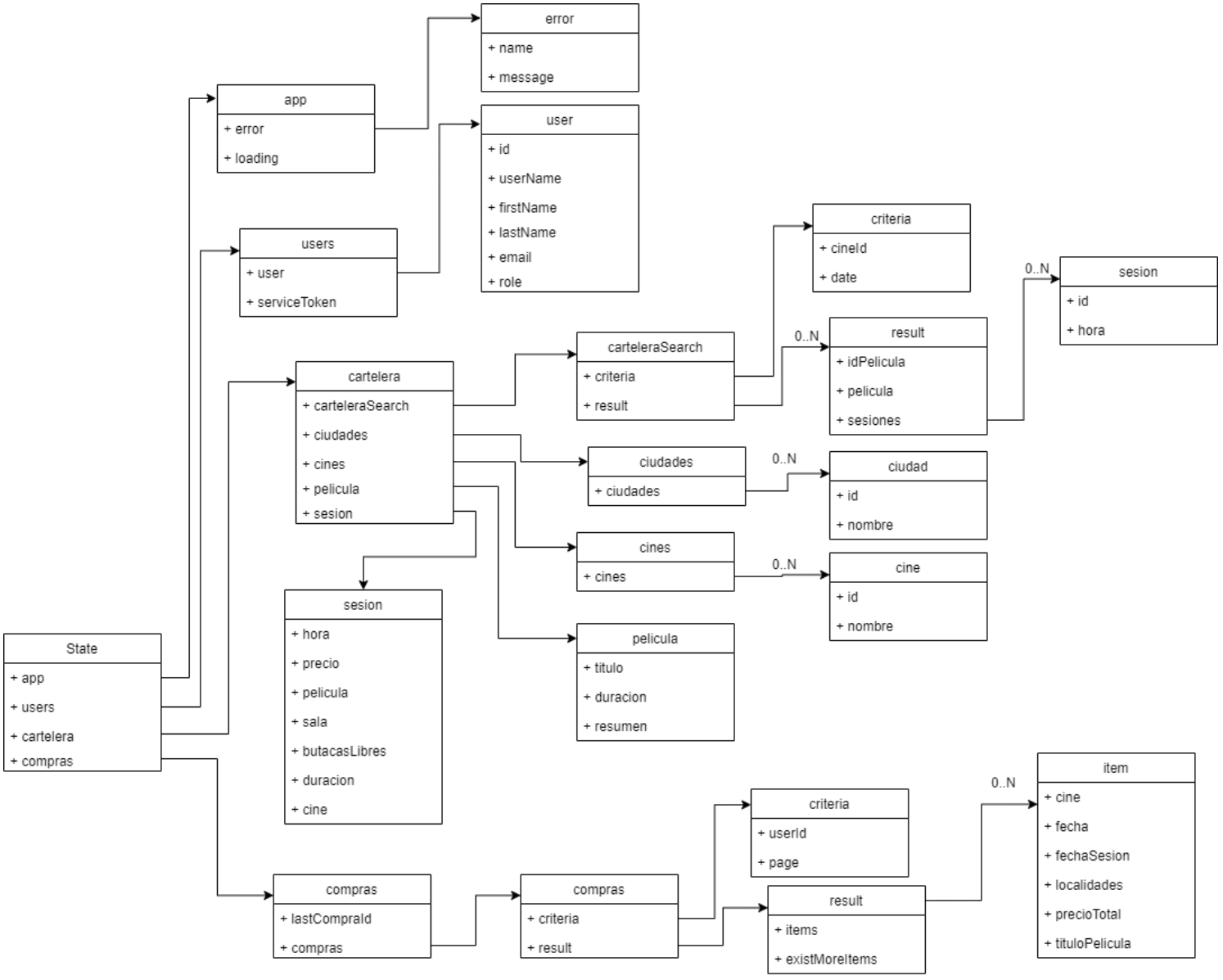
**DTO input:** ChangePasswordParamsDto

**Códigos de respuesta HTTP:** 400 Bad Request, 404 Not Found, 204 No Content

# 2. Frontend

## 2.1 Estructura de la aplicación

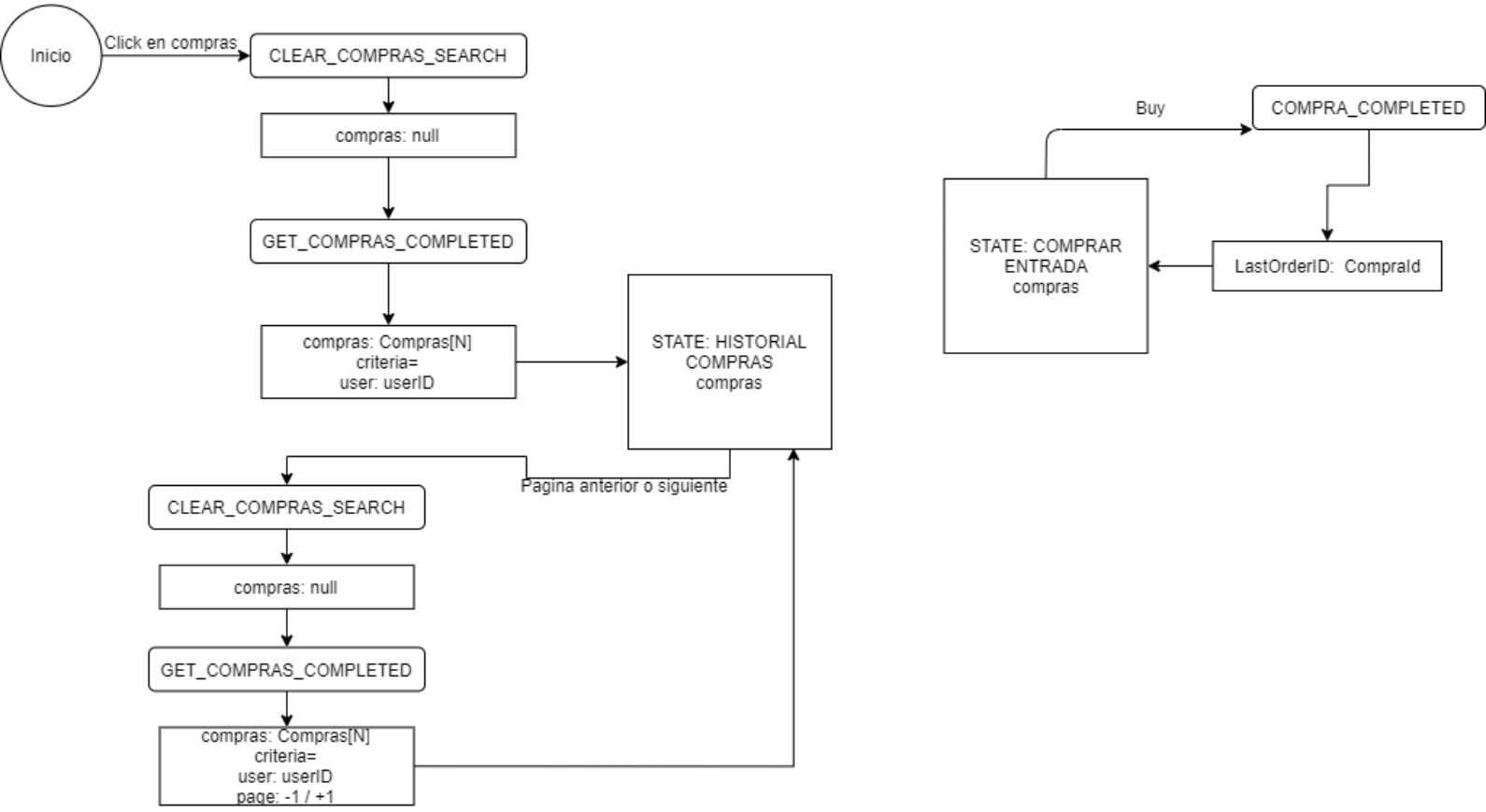
Los atributos de los diagramas de cada módulo referenciarán a estos objeto sin mostrar todos los detalles con objetivo de preservar la claridad de los mismos



# 2.2 Módulos

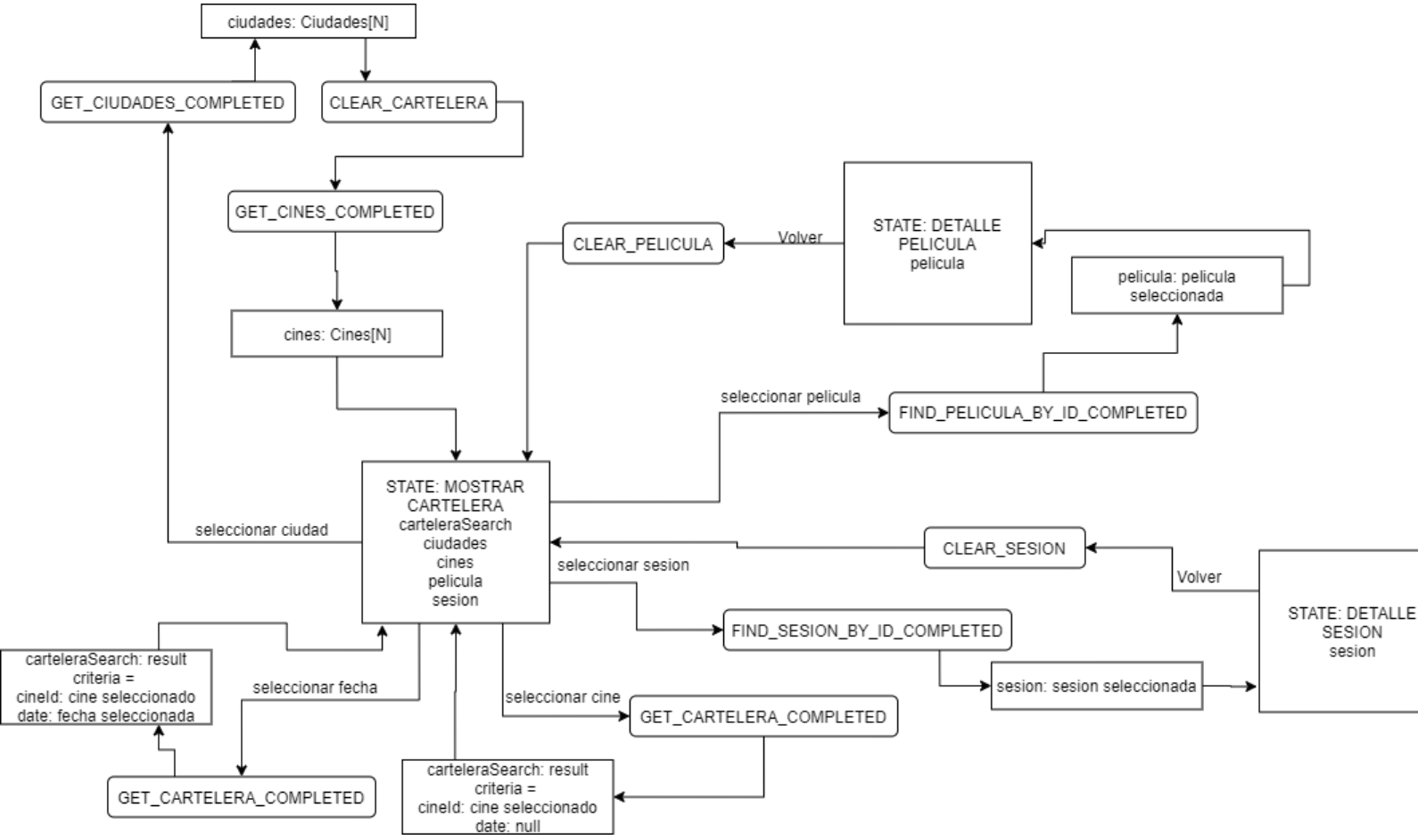
- **compras:**

Muestra el comportamiento, dado un usuario autenticado, de la navegación del historial de compras en /compras/find-orders-result y la compra de una entrada en /cartelera/sesion-detail/sesionId. Este módulo también gestiona la entrega de entradas dado un usuario ADMIN, pero como no cambia el estado interno de la aplicación si no que simplemente actualiza la base de datos del backend no está reflejado en el diagrama.



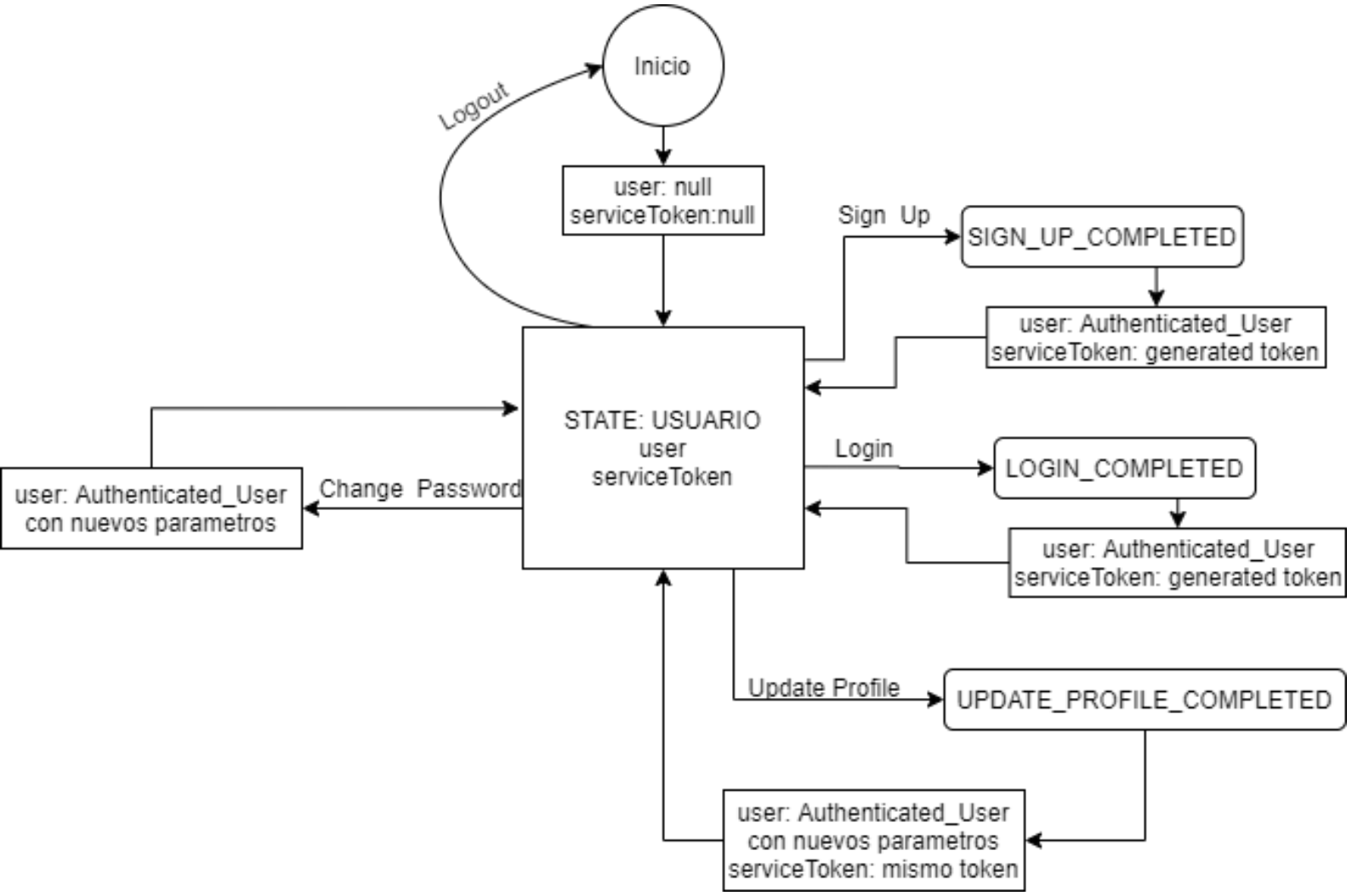
- **cartelera:**

Muestra el comportamiento de la página principal, visualización de la cartelera dado una ciudad y un cine, el detalle de una película en /cartelera/pelicula-details/peliculaId y el detalle de una sesión en /cartelera/sesion-detail/sesionId



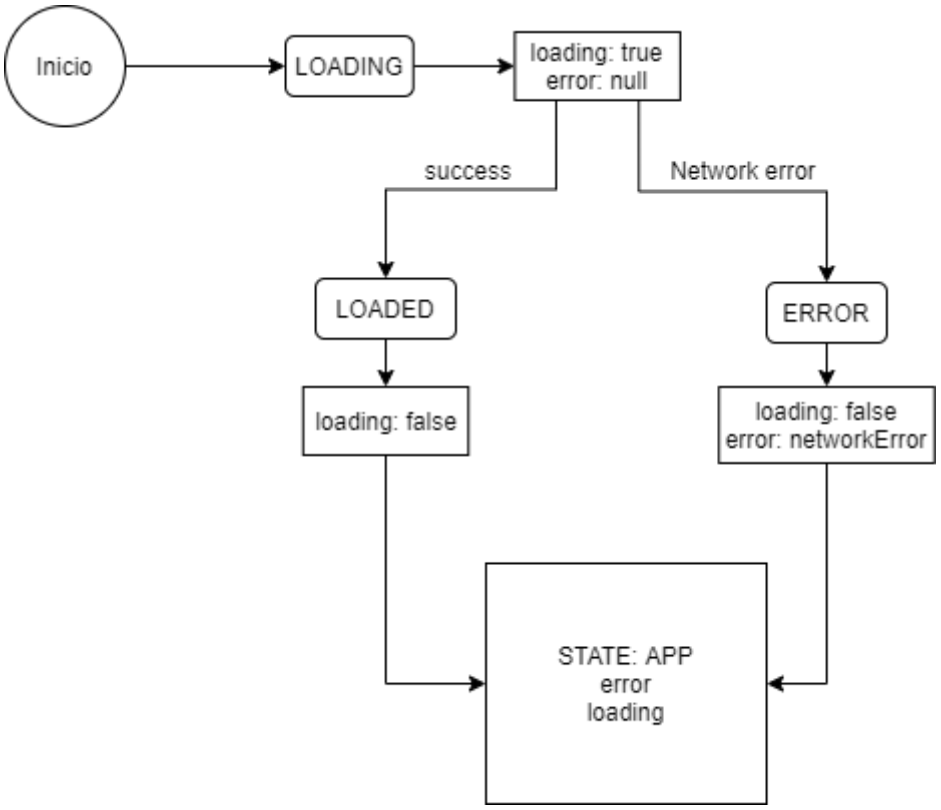
- **users:**

Muestra el comportamiento del inicio de sesión de usuarios en /users/login, registro en /users/signup, cambio de contraseña en /users/change-password y actualización de perfil en /users/update-profile.



- **app:**

Muestra el estado de la comunicación con el backend.



# 3. Trabajos Tutelados

- **Backend: BatchSize:** tanto para el caso de ver cartelera como para el de ver el historial de compras necesitamos navegara entidades que están contenidas en otras entidades.

  Decidimos colocar la anotación en las entidades de Pelicula, Sesion y Cine en el caso de visualizar histórico de compras, porque para crear los dtos necesitamos la fecha de la sesión y el título de la película.

  En el caso de la cartelera necesitamos instanciar películas y sesiones en un bucle for para crear los dtos que mandamos al servicio rest.

- **Frontend: pruebas unitarias:** se prueban 3 archivos.

  BuyForm.test: es el formulario para comprar entradas.

  - ❏ buy - success:
    el componente BuyForm recibe el id de la sesión que queremos comprar. Se crea una variable sesionId que se envía BuyForm. Se establece como estado inicial que haya 2 butacas libre y BuyForm comprueba que haya butacas disponibles para poder comprar.
  - ❏ buy - backend errors:
    En este caso se repiten las mismas órdenes y condiciones excepto que se fuerza al Backend a devolver un error para comprobar cómo se comporta el Frontend

  actions.test:
  - ❏ buy - success:
    se llama al buy de compraService, se fuerza una compra correcta utilizando el mismo método que en buy -success de BuyForm y se comprueban los campos que devuelve. Después se llama a actions.buy para corroborar que se realiza actions.buyCompleted.
  - ❏ buy - backend errors:
    se vuelven a repetir las mismas órdenes y condiciones excepto que se fuerza un error en el backend. Así, el test comprueba que bajo un error no se completa ninguna acción (ej. no se completa una compra si el servidor falla).

  reducer.test:
  - ❏ BUY_COMPLETED:
    da como estado inicial lastCompraId: null, fuerza una compra y comprueba que se actualiza correctamente ese valor, es decir, el de actions.