UNIVERSIDAD DE GUANAJUATO

DIVISIÓN DE INGENIERÍAS CAMPUS IRAPUATO SALAMANCA

SISTEMAS DE LA INFORMACIÓN

DR. JUAN CARLOS GÓMEZ CARRANZA

PRÁCTICA INDIVIDUAL

DIEGO EDUARDO ROSAS GONZÁLEZ

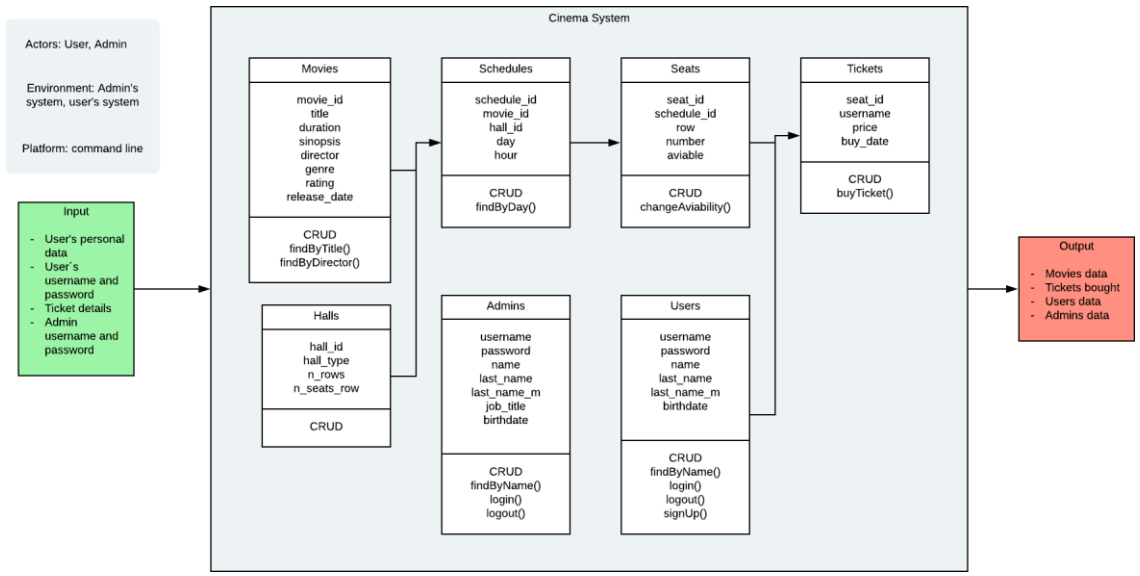
NUA: 908594

JUEVES, 28 DE MAYO DE 2020

OBJETIVO

Crear un programa escrito en Python que utilice el modelo-vista-controlador para gestionar la información de un cine. Este sistema tiene dos roles: administrador y usuario. El administrador puede gestionar todos los datos inclusive los de los otros administradores. Un usuario puede comprar un ticket para una película específica

DIAGRAMA MODULAR

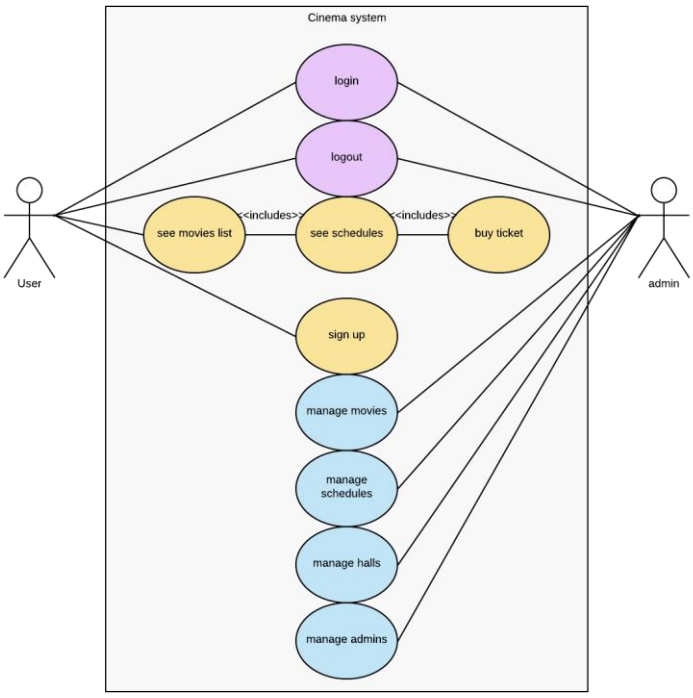


EXPLICACIÓN

Los actores que interactúan con nuestro sistema son dos. El administrador el cuál puede ser un trabajador del cine y un usuario el cuál necesita ser registrado previamente. La entrada de nuestro sistema es toda la información correspondiente a las películas, horarios y tickets. También de la información del administrador y el usuario. Se implementaron las tablas que se solicitaron. No se crearon tablas adicionales debido a que la gestión se basa principalmente en la venta del boleto y no tanto en la gestión de un catálogo de películas. Hago énfasis en que, sí se puede gestionar la información de las películas, solo no consideré necesario agregar una tabla de directores o géneros para éstas. Respaldé mi diseño en la página de Cinépolis la cual la búsqueda que existe es por título de película y el proceso de compra se hace más sencillo e intuitivo al no contar con demasiadas opciones.

Las tablas kernel de mi sistema son 'halls', 'movies', 'users' y 'admins'. 'Admins' no tiene conexión con ninguna tabla debido a que sus datos sirven para el inicio de sesión y control interno a diferencia del usuario que si "tiene" tickets. Con la película y la sala se crea uno o más horarios para éstos. Al crear un horario se crean automáticamente todos los asientos de la sala vacíos. No ligué directamente asientos a la tabla salas porque no es el mismo asiento por ejemplo A3 de la sala B-1 en un horario que en otro horario. Puede que en un horario esté ocupado ese asiento y en otro no. Por último, el usuario selecciona un asiento el cuál ya está ligado a un horario y sala específico para poder comprar un ticket.

DIAGRAMA DE CASOS DE USO



Action name: Buy ticket

Suppositions

- User have enought money
- Movie has released
- At least one hall is asigned to the movie

Pre-conditions

- User is logged in
- There are seats aviable

Start

- Select movie

Events flow

- S -> Display the movie information
- U -> Selects buy ticket option
- S -> Display aviable schedules
- U -> Select a schedule
- S -> Show a seat list
- U -> Select a seat
  - If seat is aviable
    - Then buy ticket
    - Show ticket information
  - If seat is not aviabe
    - Shows prompt with error
    - Ask for other seat

Postconditions

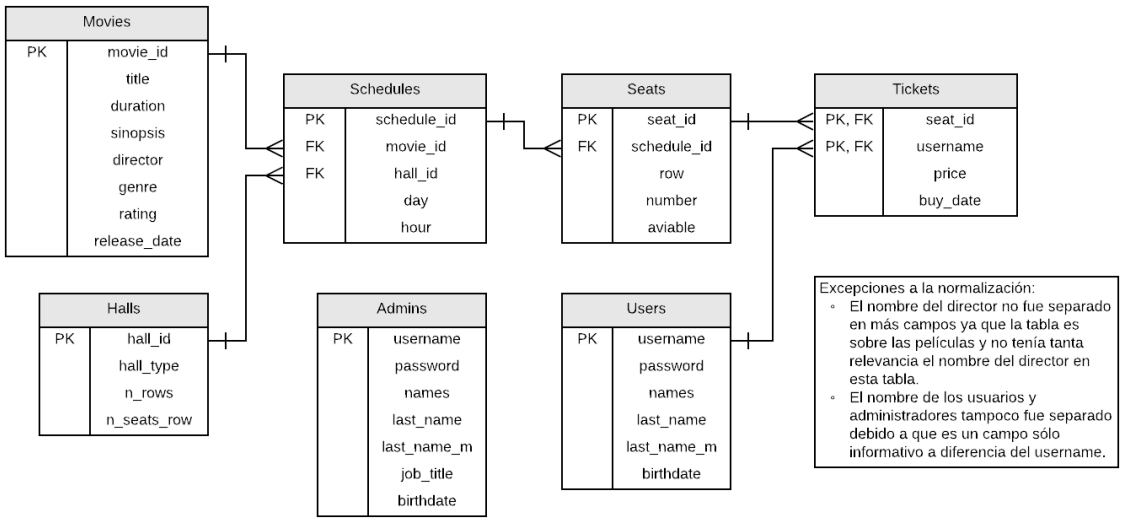
- Database updated
- Seat avilability changed

EXPLICACIÓN

Las acciones que tienen en común los dos actores son el inicio de sesión y cerrar sesión. Incluí el proceso para comprar un ticket el cuál requiere de unas acciones previas por parte del usuario. Hay un error con la acción sign up ya que le corresponde al administrador y no al usuario. Las siguientes acciones corresponden a un CRUD de toda la base de datos y que sólo puede hacer el administrador.

Describí el proceso más largo que hay en el diseño el cuál es comprar un ticket. La acción comienza desde que se selecciona una película debido a que no se puede comprar in ticket sin saber su identificador por lo cuál es necesario mostrar la lista de películas previamente. El procedimiento pasa por las vistas que corresponden a los modelos en el mismo orden en el cuál se diseñó en el diagrama modular. Al finalizar se actualiza el status del asiento para indicar que yá está ocupado y el boleto se agrega a la lista de boletos del usuario.

DIAGRAMA RELACIONAL



EXPLICACIÓN

Este diseño fue una traducción completa del diagrama modular así que los puntos correspondientes a su diseño ya fueron mencionados. En el diseño del código SQL se trataron de poner identificadores a las tablas debido a que, debido a las características de las tablas 'schedules' y 'seats' hacía que si se usaban los mismos campos para hacer una llave compuesta, al pasar a la siguiente tabla también se iba a hacer esto y daba como resultado una tabla en donde había más llaves foráneas que campos propios de la tabla. Por ejemplo, en 'schedules' cada horario se caracteriza por tener la combinación de película, sala, día y hora diferentes. Siempre hay tres campos que se pueden repetir y eso hubiera requerido que la llave compuesta comprendiera los 4 campos de la tabla.

CONCLUSIÓN

Se construyó un sistema que considero que está organizado gracias al uso del MVC el cuál me permitió dividir mi desarrollo en etapas fácilmente identificables. Otra ventaja fue que pude hacer pruebas de las funciones individualmente separando las funciones del modelo de la vista y el controlador. De hecho, cuando hubo más problemas en desarrollo fue cuando se agregaron funciones que no habían sido planeadas desde el inicio por lo cuál se tuvo que modificar los tres archivos y esto derivó en una más tardada identificación de los bugs que surgieron. Por otro lado, este sistema realiza la función para la cuál está diseñado y no tiene problemas graves como cierres inesperados o datos erróneos introducidos a la base de datos. El modelo transaccional de la base de datos ayudó mucho a esta parte al deshacer los cambios en la base de datos cuando había errores.