Instituto Tecnológico de Costa Rica
Área de Ingeniería en Computadores
Bases de Datos - CE3101

Profesor:
Marco Rivera Meneses

Documentación Externa

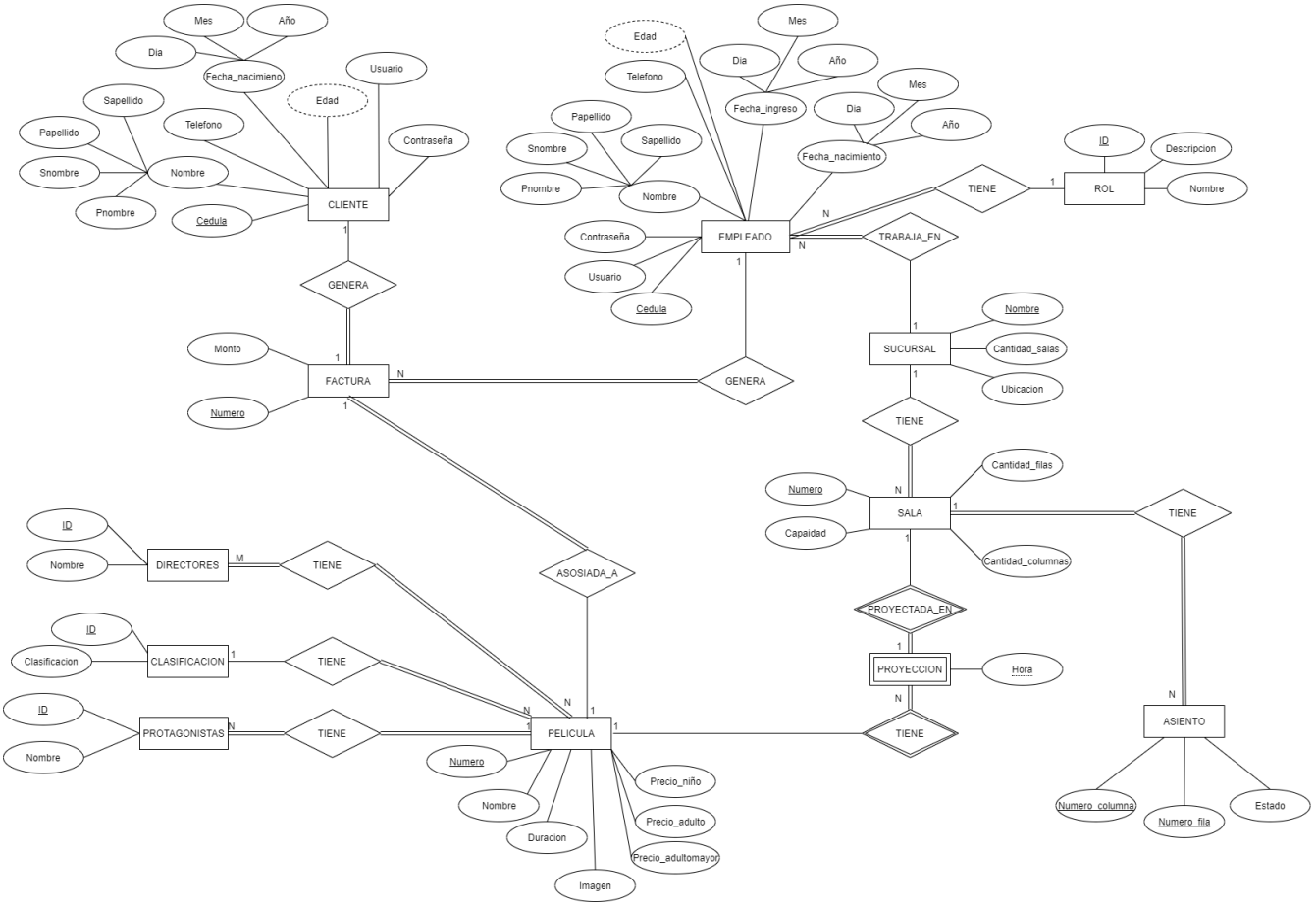
Estudiantes:
Brandon Gomez Gomez
Erick Madrigal Zavala
Marco Rivera Serrano
Kevin Rodriguez Lanuza

Grupo:
1

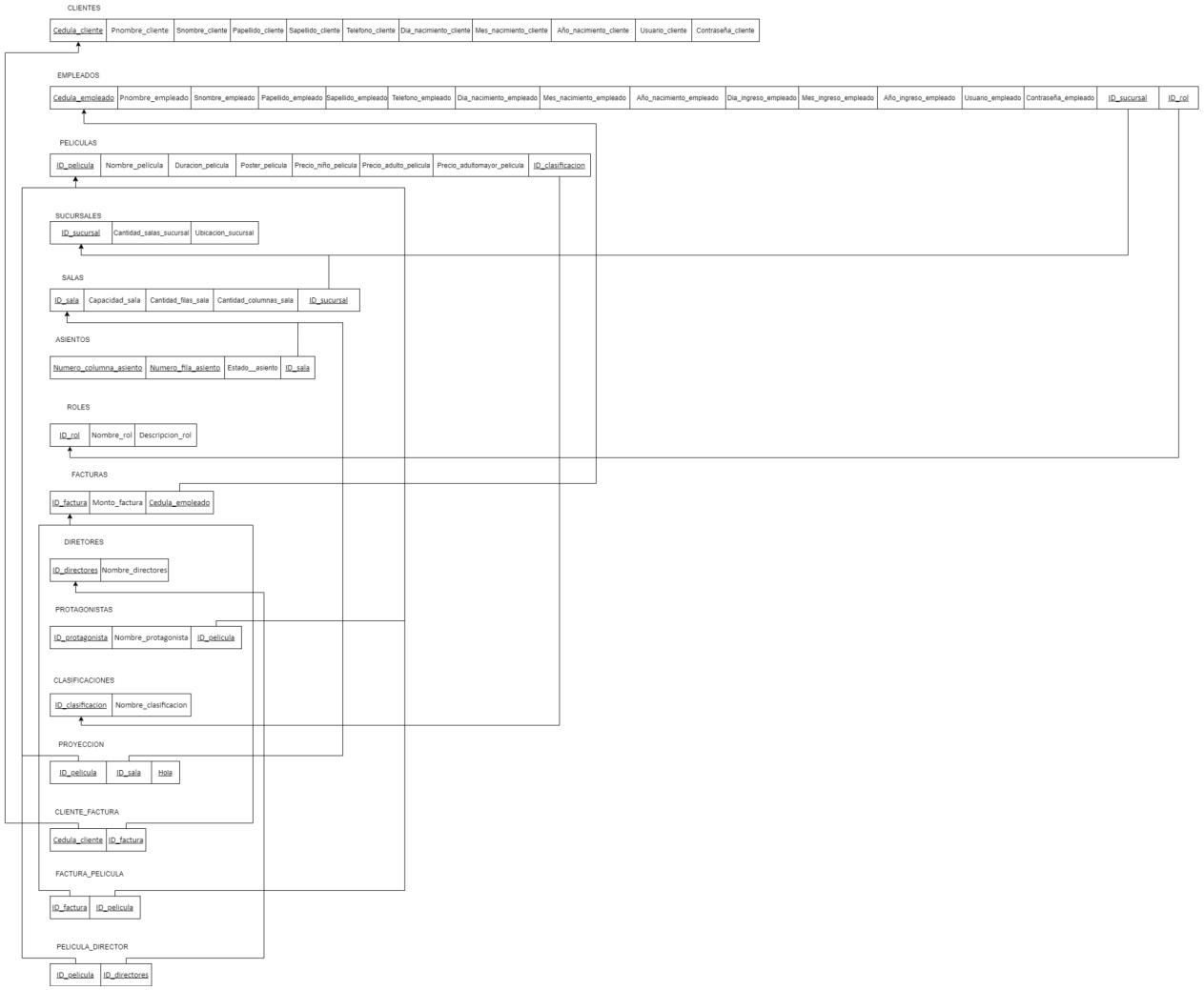
II Semestre

2021

Modelo Conceptual:



Modelo Relacional:



1. Mapeo de tipos de entidades fuertes

Para cada entidad fuerte construimos una relación que contenga los atributos simples, en caso de los atributos compuestos, solo ponemos los atributos simples que los componen.

Estas son las relaciones resultantes.

Relación resultante	Llave primaria	Atributos
CLIENTES	Cedula_cliente	Cedula_cliente Pnombre_cliente Snombre_cliente Papellido_cliente Sapellido_cliente Telefono_cliente Dia_nacieminto_cliente Mes_nacieminto_cliente Año_nacieminto_cliente Usuario_cliente Contraseña_cliente
EMPLEADOS	Cedula_empleado	Cedula_empleado

		Pnombre_empleado Snombre_empleado Papellido_empleado Sapellido_empleado Telefono_empleado Dia_nacieminto_empleado Mes_nacieminto_empleado Año_nacieminto_empleado Dia_ingreso_empleado Mes_ingreso_empleado Año_ingreso_empleado Usuario_empleado Contraseña_empleado
PELICULAS	ID_pelicula	ID_pelicula Nombre_pelicula Duracion_pelicula Poster_pelicula Precio_niño_pelicula Precio_adulto_pelicula Precio_adultomayor_pelicula
SUCURSALES	ID_sucursal	ID_sucursal
SALAS	ID_salas	ID_salas
ASIENTOS	-Numero_columna_asiento -Numero_fila_asiento	Numero_columna_asiento Numero_fila_asiento Estado__asiento
ROLES	ID_rol	ID_rol Nombre_rol Descripcion_rol
FACTURAS	ID_factura	ID_factura Monto_factura
PROTAGONISTAS	ID_protagonista	ID_protagonista Nombre_directores
DIRECTORES	ID_directores	ID_directores Nombre_protagonista
CLASIFICACIONES	ID_clasificacion	ID_clasificacion Nombre_clasificacion

2. Mapeo de tipos de entidades débiles

Para cada entidad debil construimos una relación que contenga los atributos simples, en caso de los atributos compuestos, solo ponemos los atributos simples que los componen. Esta es la relación resultante, para este caso en particular, según el modelo conceptual, solo tenemos una entidad débil, además se le agrega como llave foránea las llaves primarias de las entidades propietarias.

Relación	Llave parcial	Llave foránea	Atributos

PROYECCION	Hora	-ID_pelicula -ID_sala	ID_pelicula ID_sala Hora

3. Mapeo de tipos de asociaciones binarias 1:1

Primero identificamos las relaciones de este tipo, en nuestro caso solo hay 3: Cliente-Factura, Factura-pelicula y proyección-sala.

Utilizamos referencia cruzada, donde hacemos una nueva relación, incluimos como llave foránea las llaves primarias de las entidades relacionadas, y se suman los atributos propios de la relación.

Relación	Llaves foráneas	atributos
CLIENTE_FACTURA	Cedula_cliente ID_factura	Cedula_cliente ID_factura
FACTURA_PELICULA	ID_factura ID_pelicula	ID_factura ID_pelicula
Para el caso de Proyección-sala, al hacer una nueva relación, se pondría la llave primaria de sala y proyección, pero esta no tiene,sin embargo, consideramos que es suficiente con la relación PROYECCION del paso dos del mapeo.		

4. Mapeo de tipos de asociaciones binarias 1:N

Primero identificamos las relaciones de este tipo:

- Película-clasificación
- Protagonista-película
- Película-proyeccion
- Sucursal_sala
- Sucursal-empleado
- Empleado-rol
- Sala-Asiento
- Empleado-factura

Relaciones 1:N	
Lado N	Lado 1
Película	clasificación
Protagonista	película
sala	Sucursal
empleado	Sucursal
Empleado	rol
Asiento	Sala

factura	Empleado

Incluimos como llave foránea en el lado N de la relación, la llave primaria del lado 1, no creamos una nueva relación, solo se hace la agregación en la relación existente.

Relación resultante	Nueva llave foranea	Atributos
EMPLEADOS	ID_sucursal ID_rol	Cedula_empleado Pnombre_empleado Snombre_empleado Papellido_empleado Sapellido_empleado Telefono_empleado Dia_nacieminto_empleado Mes_nacieminto_empleado Año_nacieminto_empleado Dia_ingreso_empleado Mes_ingreso_empleado Año_ingreso_empleado Usuario_empleado Contraseña_empleado ID_sucursal ID_rol
PELICULAS	ID_clasificacion	ID_pelicula Nombre_pelicula Duracion_pelicula Poster_pelicula Precio_niño_pelicula Precio_adulto_pelicula Precio_adultomayor_pelicula ID_clasificacion
SALAS	ID_sucursal	ID_salas ID_sucursal
ASIENTOS	ID_salas	Numero_columna_asiento Numero_fila_asiento Estado__asiento ID_salas
FACTURAS	Cedula_empleado	ID_factura Monto_factura Cedula_empleado
PROTAGONISTAS	ID_pelicula	ID_protagonista Nombre_directores ID_pelicula

5. Mapeo de tipos de asociaciones binarias N:M

Para cada asociación de este tipo hacemos una relación intermedia, y la llave primaria de ambas relaciones asociadas componen la llave primaria de esta nueva relación.
Para nuestro modelo de entidad-relación, sólo tenemos una asociación de este tipo, que es entre películas y directores.

Haciendo referencia cruzada

Relación	Llaves foráneas	atributos
PELICULA_DIRECTOR	ID_pelicula ID_directores	ID_pelicula ID_directores

6. Mapeo de atributos multivaluados

Para nuestro modelo de entidad-relación no hay definidos atributos multivaluados.

Descripción de las estructuras de datos desarrolladas (Tablas)

Se tienen un total de doce entidades, las cuales son:

Bills:

- id_bill: es la llave primaria de la tabla y es la cual identifica a cada factura, es de tipo int.
- total_bill: es la cantidad total a pagar, es de tipo double point (conocido como float).
- id_employee_bill: llave foránea asociada al empleado, es de tipo int.
- id_client_bill: llave foránea asociada al cliente, es de tipo int.

Branches:

- id_branch: es la llave primaria de la tabla y es la cual identifica a cada sucursal, es de tipo int.
- name_branch: es el nombre de la sucursal, es de tipo text.
- cant_rooms_branch: es la cantidad de salas que tiene la sucursal, es de tipo int.
- address_branch: es la dirección de la sucursal, es de tipo text.

Classifications:

- id_classif: es la llave primaria de la tabla e identifica a cada clasificación, es de tipo int.
- classif: es la clasificación (adulto, niño adulto mayor u otras), es de tipo text.

Clients:

- id_client: es la llave primaria de la tabla e identifica a cada cliente y es su cédula, es de tipo int.
- first_name_client: es el primer nombre del cliente, es de tipo text.
- second_name_client: es el segundo nombre del cliente, es de tipo text.
- first_last_name_client: es el primer apellido del cliente, es de tipo text.
- second_last_name_client text: es el segundo apellido del cliente, es de tipo text.

- phone_client: es el número telefónico del cliente, es de tipo text.
- birth_date_client: es la fecha de nacimiento del cliente, es de tipo date.
- password_client: es la contraseña del cliente, es de tipo text.
- user_client: es el usuario del cliente, es de tipo text.

Directors:

- id_director: es la llave primaria de la tabla e identifica cada director, es de tipo int.
- name_director: es el nombre completo del director, es de tipo text.

Employees:

- id_employee: es la llave primaria de la tabla e identifica a cada empleado y es su cédula, es de tipo int.
- first_name_employee: es el primer nombre del empleado, es de tipo text.
- second_name_employee: es el segundo nombre del empleado, es de tipo text.
- first_last_name_employee: es el primer apellido del empleado, es de tipo text.
- second_last_name_employee text: es el segundo apellido del empleado, es de tipo text.
- phone_employee: es el número telefónico del empleado, es de tipo text.
- birth_date_employee: es la fecha de nacimiento del empleado, es de tipo date.
- admission_date_employee: es la fecha de ingreso del empleado, es de tipo date.
- password_employee: es la contraseña del empleado, es de tipo text.
- user_employee: es el usuario del empleado, es de tipo text.
- id_branch_employee: es el id de la sucursal en la cual trabaja el empleado, es de tipo int.
- id_rol_employee: es el id del rol el cual ejerce el empleado, es de tipo int.

Movies:

- id_movie: es la llave primaria de la tabla e identifica a cada película, es de tipo int.
- name_movie: es el nombre de la película, es de tipo text.
- duration_movie: es la duración de la película, es de tipo text.
- poster_movie: es el poster de la película, es de tipo text dado que es una url.
- price_elder_movie: precio de la película para adultos mayores, es de tipo int.
- price_adult_movie: es el precio de la película para adultos, es de tipo int.
- price_kid_movie: es el precio de la película para niños, es de tipo int.
- id_director_movie: es el id del director de la película, es de tipo int.
- id_classif_movie: es el id de la clasificación de la película, es de tipo int.
- id_protagonist_movie: es el id del protagonista de la película, es de tipo int.

Projections:

- id_projection: es la llave primaria de la tabla e identifica a cada proyección, es de tipo int.
- time_projection: es la hora a la cual se hará la proyección, es de tipo text.
- id_movie_projection: es el id película de la película la cual se proyectará.
- id_room_projection:

Protagonists:

- id_protagonist: es la llave primaria de la tabla e identifica a cada protagonista, es de tipo int.
- name_protagonist: es el nombre completo del protagonista, es de tipo text.

Roles:

- id_rol: es la llave primaria de la tabla e identifica a cada rol, es de tipo int.
- name_rol: es el nombre del rol, es de tipo text.
- description_rol: es la descripción del rol, es de tipo text.

Rooms:

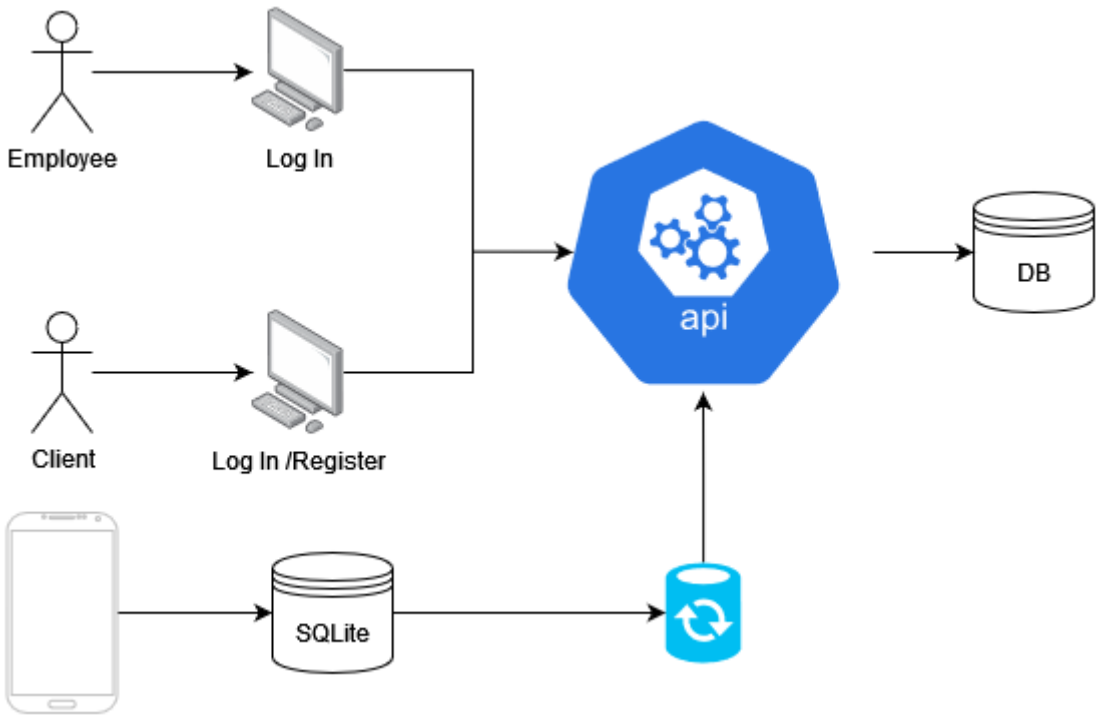
- id_room: es la llave primaria de la tabla e identifica a cada sala, es de tipo int.
- capacity_room: es la capacidad de la sala, es de tipo int.
- rows_room: es la cantidad de filas de la sala, es de tipo int.
- columns_room: es la cantidad de columnas de la sala, es de tipo int.
- id_branch_room: es el id de la sucursal en la cual se encuentra la sala, es de tipo int.

Seats:

- id_seat: es la llave primaria de la tabla e identifica a cada asiento, es de tipo int.
- row_seat: es la fila en la cual se encuentra el asiento, es de tipo int.
- column_seat: es la columna en la cual se encuentra el asiento, es de tipo int.
- state_seat: es el estado del asiento, es de tipo int.
- id_room_seat: es el id de la sala en la cual se encuentra el asiento, es de tipo int.

Descripción de la arquitectura desarrollada

Como se tiene previsto, se debe tener el API funcionando. Cuando un cliente o administrador utiliza la aplicación, la aplicación realiza la conexión que necesite al API y este utiliza los queries necesarios para obtener la información desde la base de datos. El siguiente diagrama explica la funcionalidad de la aplicación con sus componentes:



Problemas conocidos

A nivel de API:

- Al no poder utilizar triggers, no se pueden realizar los deletes si la tupla depende de otra tabla, dado que postgreSQL tira error 500 al front end. Sin embargo, no se encontró alguna manera de obtener dicha respuesta para manejar este error dentro del front end.

A nivel de la aplicación móvil:

- No se logró crear una conexión entre la base de datos de sqlite y postgree. Para intentar solucionarlo se buscó en varias fuentes pero todas estas sincronizaciones se realizaban de forma manual o con aplicaciones externas que no se podían adaptar a android studio.

A nivel de la aplicación web:

- No se logró generar una factura para el cliente que contuviera toda la información necesaria en ella debido a que no se podía obtener cierta información de los gets utilizando typescript, pero dicha información sí aparece en el API.
- No se logró visualizar asientos anteriormente reservados, debido a que el algoritmo tiene un alto nivel de complejidad para obtener cuales son utilizando typescript, dado a que intuitivamente, genera errores.
- No se logró editar películas dado que la carga del póster se realiza dinámicamente y para poder volverlo a cambiar genera problemas de eficiencia y es difícil. Sin embargo, su carga en la firebase es más sencillo.
- No se implementó editar protagonistas ni directores, dado que es algo innecesario. Porque para crear una película, ambas entidades ya deben existir en la base.

Problemas encontrados

A nivel de API:

- No se podía hacer inserción de si se recibían otro tipo de datos diferentes a los de la base. Para solucionar este error, se utilizaron diferentes queries buscando el id del o los strings que se reciben, luego, se confirma si estos queries contienen algo o están vacíos y luego, se realiza la inserción a la base de datos.
- No se podían obtener nombres de otras tablas, solo sus ids. Una manera de solucionar esto fue creando otro modelo que contenga toda la información que se quería mostrar. Finalmente, se realizaron queries entre las tablas que se quería obtener dicha información.

A nivel de la aplicación móvil:

- A la hora de entrar en la ventana de selecciones para la película no se podía cargar automáticamente el spinner ya que esto producía que la aplicación crasheara. Por la experiencia adquirida en trabajos anteriores se recurrió a poner un botón que hace la llamada a la función de carga del spinner para evitar el cierre de la aplicación.
- En sqlite no se puede crear fácilmente las llaves foráneas, para poder hacer dicho proceso se debe borrar la tabla inicial y luego crearla de nuevo con la columna con la llave foránea, para evitar este problema la base de datos se creó a partir de un script generado por el mismo sqlite para tener la certeza de que no habrá errores ya que las tablas se crean en el orden necesario para que siempre exista la tabla de donde se importa la llave foránea.

A nivel de la aplicación web:

- Para poder guardar imágenes y que no estuvieran en la base de datos, se utilizó una base de datos en la nube llamada firebase, en la cual se guarda dicha imagen y se obtiene su URL. Este mismo, es insertado en la cartelera inicial y en la página de proyecciones para que el usuario pueda ver en qué película quiera reservar los asientos.

Reuniones

- 20 de septiembre del 2021

Primera reunión del equipo. En este día, el grupo discutió el proyecto y se asignaron las tareas para cada miembro del equipo. Asimismo, se establecieron algunas de las rutas principales que son necesarias para el funcionamiento de la aplicación.

- 27 de septiembre del 2021

Segunda reunión del equipo. Para esta fecha se tienen adelantadas algunas plantillas para ser utilizadas en el desarrollo de la aplicación. Algunas de estas fueron, la base de datos, api y front end.

- 4 de octubre del 2021

Tercera reunión del equipo. Para este día, se lograron algunas conexiones con el front end para realizar algunos métodos básicos (get, post, update, delete) y establecer nuevos modelos según lo que necesite el front end.

- 7 de octubre del 2021

Reunión entre Kevin, Brandon y Marco. Resolución de algunos choques entre el front end y el back end. Se probaron más rutas compuestas de inserción utilizando datos desde el front end.

- 10 de octubre del 2021

Reunión entre Brandon y Marco. Pequeña reunión para agregar un modelo extra.

- 11 de octubre del 2021

Cuarta Reunión del equipo. En esta reunión se tomaron en cuenta todos aquellos puntos restantes para la solución del proyecto y trabajar en ellos.

- 14 de octubre del 2021

Reunión entre Marco y Brandon. En esta reunión se trabajaron las reservas de asientos para clientes. También se logró solucionar un error dentro del repositorio debido a un merge.

- 15 de octubre del 2021

Reunión entre Marco, Brandon y Kevin. Se establecieron nuevos modelos para ser mostrados en el front end utilizando queries avanzados de inner join dentro del api sin necesidad que el front end aplique alguna lógica.

- 17 de octubre del 2021

Entrega del proyecto

Actividades realizadas (Bitácora)

Brandon:

- 30/09/2021
  Creación de la página para gestión de clientes.
- 02/10/2021
  Creación de la página para gestión de salas.
- 04/10/2021
  Creación de la página para gestión de empleados.
- 05/10/2021
  Creación de la página para gestión de películas.
- 07/10/2021
  Creación de la página para gestión de roles.
- 09/10/2021
  Creación de la página para gestión de protagonistas y directores.
- 10/10/2021
  Creación de la página principal del cliente.
- 12/10/2021
  Creación de la página que muestra la cartelera, y que permite escoger si ver la película..

- 13/10/2021
  Creación de la página para reservar asientos..
- 14/10/2021
  Comunicación entre el front end, de la vista administrador, con la el back end
- 16/10/2021
  Comunicación entre el front end, de la vista cliente, con la el back end

Erick:

- 28/9/2021.
  Creación de la sección principal donde se realiza la transacción
- 2/10/2021
  Adición de los spinners para la selección de la sala, pelicula y horario
- 6/10/2021
  Creación de lógica para los spinners
- 10/10/2021
  Creación de la base de datos en sqlite en base a la base de datos de postgres
- 12/10/2021
  Cambio de la interfaz para añadir imágenes de las salas
- 13/10/2021
  Implementación de la base de datos en la aplicación
- 14/10/2021
  Añadido de variables específicas para la interfaz, cambios de la base de datos de sqlite
- 15/10/2021
  Creación de la función que crea la factura en la base de datos y actualización de los asientos en la misma
- 16/10/2021
  Inserción de datos a una base de datos extra para poder realizar pruebas de la aplicación

Marco:

- 28/09/21
  Creación de un API básico (CRUD) para cliente, empleado, sucursal, película y roles.
- 01/10/21
  Creación de un API básico (CRUD) para directores, clasificación, protagonistas, salas, asientos, proyecciones y facturas.
  También se solucionó el problema de CORS al tratar de utilizar métodos de edición en el API.
- 05/10/21
  Creación de las llaves foráneas y adición de estas mismas en la base de datos
- 09/10/21
  Creación de métodos para poder trabajar las llaves foráneas desde el API
- 13/10/21
  Se añadieron verificaciones dentro del api por si alguno de los queries no contenía alguna tupla y evitar errores en el front end
- 17/10/21
  Se agregaron pequeños cambios para mejorar la visualización en el front end

Kevin:

- 29/09/21

Creación de una página de registro.

- 04/10/21
  Creación de una página para registrar sucursales
- 10/10/21
  Creación de una página para registrar nuevas proyecciones
- 13/10/21
  Creación de una página de login
- 16/10/21
  Arreglos a la página de login para que tenga comunicación con el back end.

Conclusiones

Las bases de datos son muy versátiles para el manejo de información masiva. Además, lograr las relaciones entre ellas permite mantener un control adecuado de la información. Sin embargo, manejar dicha información necesita su cuidado dado que borrar información es sencillo si se tiene el query adecuado.

Lograr que la interfaz de usuario sea amigable con este mismo, permite que el cliente o el administrador no se pierda o confunda a la hora de navegar por la aplicación. Esto permite una experiencia agradable y que este pueda manipular la información con mucha facilidad. Sin embargo, si a este se le presenta una aplicación en la cual no pueda entender con facilidad dónde se encuentran las páginas o información, la experiencia no será gratificante y podría manipular la información erróneamente.

La implementación de aplicaciones para dispositivos móviles permite que los usuarios tengan formas de acceso al servicio que se desea brindar ya que el usuario no depende de un ordenador para poder acceder al contenido de la misma y puede estar seguro de que dicha aplicación funciona de una mejor manera que una página web en su dispositivo

Recomendaciones

Tener bastante práctica con respecto a edición y confiscación de páginas web utilizando Angular, Bootstrap, CSS y HTML, dado que la curva de aprendizaje es bastante complicada y requiere un manejo complejo de archivos para saber exactamente donde se puede encontrar un error de ejecución. Además, se requiere un conocimiento previo avanzado para lograr una interfaz amigable para el usuario sin que este pierda o no sepa qué hacer.

Tener una comunicación constante con el front end para tener en cuenta todos las rutas para el API, qué tipos de parámetros se requieren o qué tipo de respuesta se espera. Por otra parte, si existen varios métodos, pero con diferentes parámetros de entrada, se necesita saber cual ruta debe ser llamada para evitar choques con la base de datos.

Bibliografía

Developers, G. (n.d.). *Android Developers*. obtenido de: https://developer.android.com/docs?hl=es-419

Fazt Code. (17 de mayo de 2019). *PostgreSQL Instalación en Windows 10* [Archivo de video]. Youtube. https://www.youtube.com/watch?v=cHGaDfzJyY4

The Coder Cave esp. (12 de marzo de 2021). *Introducción a .NET con PostgreSQL - Crea una API Rest con PostgreSQL - ¿Mejor que SQL Server?* [Archivo de video]. Youtube. https://www.youtube.com/watch?v=cHGaDfzJyY4

Diagrama de clases

https://github.com/SlimeVRS/CineTEC/blob/main/Documentaci%C3%B3n/Im%C3%A1genes/Diagram%20Class.png

Explicación del diseño

Primero, se explicarán aquellas tablas que no dependen de ninguna otra. Les denominaremos tablas padre. Estas son:

- Branches.
- Clients.
- Roles.
- Protagonists
- Directors.
- Classification

¿Por qué se consideran así? Porque estas tablas no dependen de ninguna otra. Dentro de la estructura propia de la base de datos, se pueden agregar la cantidad que quiera de tuplas (siempre y cuando no supere el límite de un entero) en estas tablas sin necesidad de agregar llaves foráneas.

Seguidamente, están las tablas “hijo”, estas tablas dependen únicamente de las tablas padre utilizando llaves foráneas. Es decir, a la hora de crear una tupla para estas tablas, se necesita hacer referencia adecuadamente a una tabla padre. Las tablas hijas son:

- Movies.

Esta depende de las tablas Protagonists, Directors y classifications. Los motivos son obvios dado que son las características principales de una película y, por ejemplo, un actor puede estar en varias películas. Lo mismo ocurre para directores y clasificaciones.

- Employees.

Esta depende de las tablas Branches y Roles. Esto se debe a que un empleado debe trabajar en una sucursal y debe tener un rol. Asimismo, varios empleados pueden tener un rol y trabajar en la misma sucursal.

- Rooms.

Esta depende únicamente de la tabla Branches, dado que una sala necesita estar en una sucursal.

Finalmente, tenemos las tablas bebés. Estas tablas dependen tanto de las tablas hijos como de las tablas padres. Estas tablas son:

- Projections.

Las proyecciones dependen de las salas dado que se necesita saber en cual se proyectará la propia película.

- Seats.

Depende de las salas, principalmente porque un asiento necesita sí o sí un lugar en el cual estar.

- Bills.

Las facturas dependen de las tablas Branches, dado que se necesita el nombre de la sucursal donde el cliente verá la película; Movies, se requiere el nombre de la película que se quiere ver; Projections, se necesita la hora y día de la función; Clients, quién compró el o los boletos y Seats, cuáles asientos fueron reservados