CineTEC: Documentación técnica

Bases de Datos

Instituto Tecnológico de Costa Rica

Proyecto #1

Carlos Adrián Araya Ramírez ***2018319701***

José Alejandro Chavarría Madriz ***2019067306***

Sebastián Mora Godínez ***2019227554***

Michael Shakime Richards Sparks ***2018170667***

Prof. Marco Rivera Meneses

Índice

Diagramas

Modelo Conceptual

(Ver en ***figura 1***)

Modelo Relacional

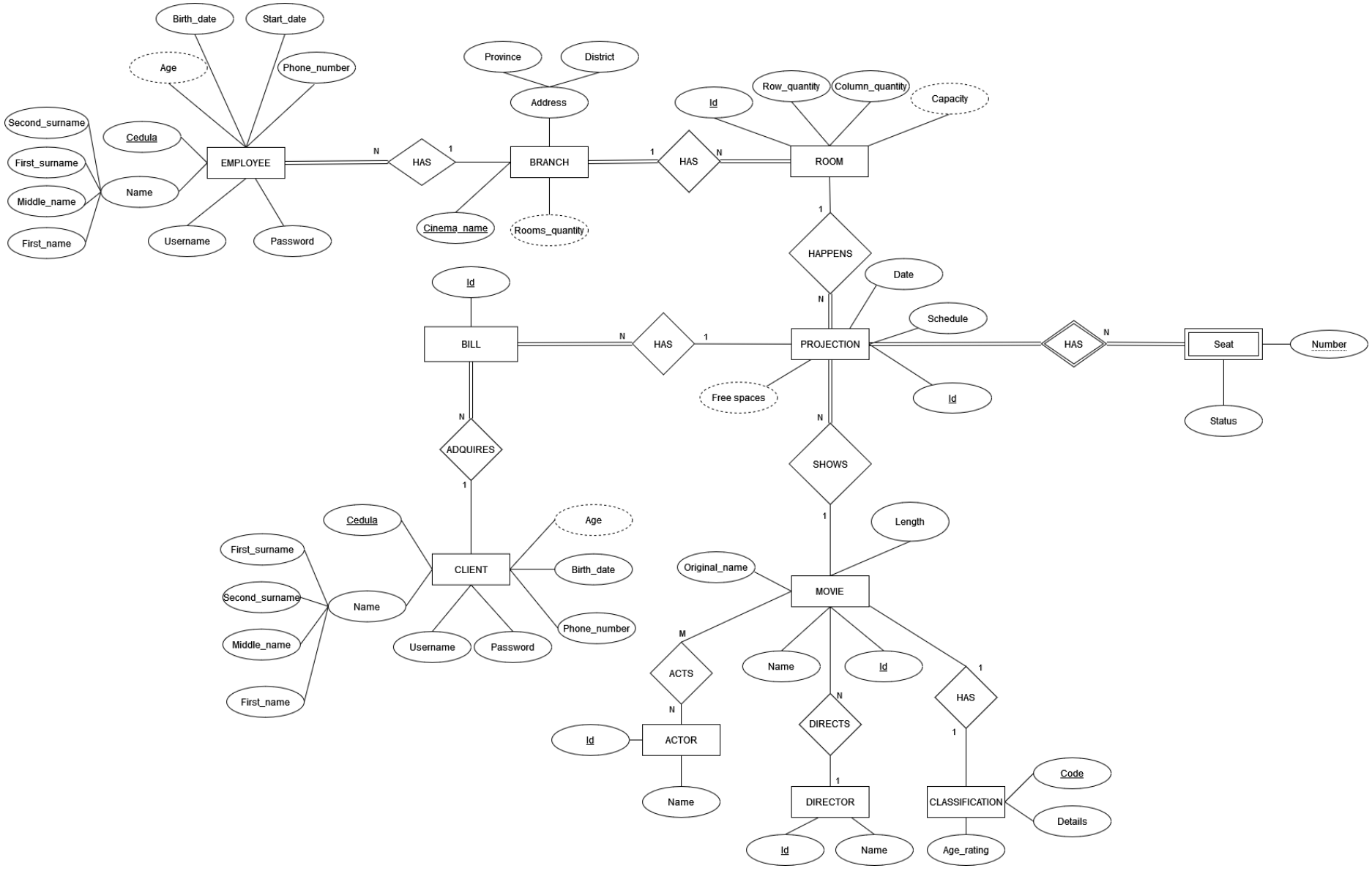
(Ver en ***figura 2***)

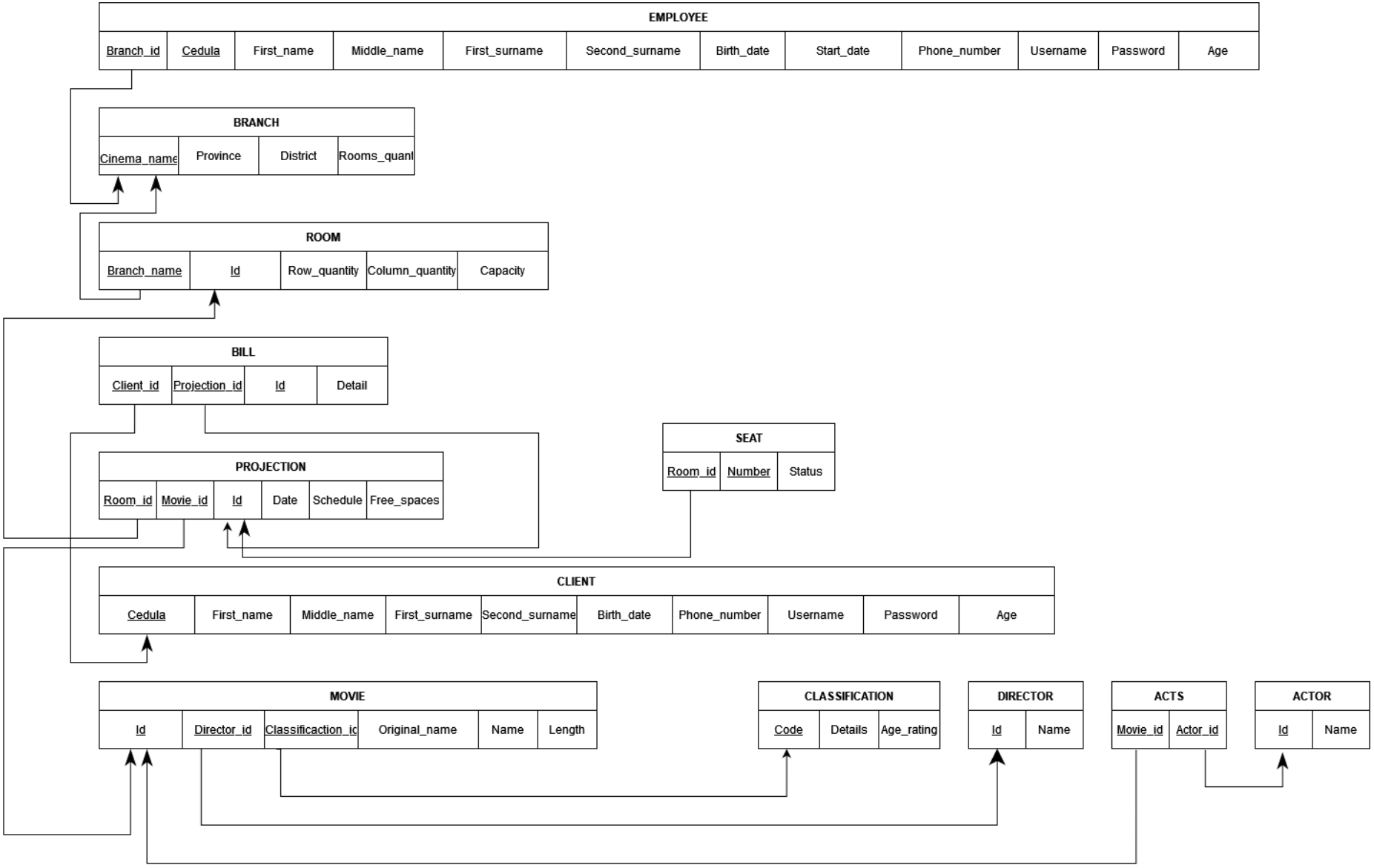
Diagrama de Clases

(Ver en ***figura 3***)

Diagrama de Arquitectura

(Ver en ***figura 4***)


Birth_date
Start_date
Age
Phone_number
Second_surname
Cedula
First_surname
Name
Middle_name
First_name
EMPLOYEE
Username
Password
N
HAS
1
Province
District
Address
BRANCH
Cinema_name
Rooms_quantity
1
HAS
N
Id
Row_quantity
Column_quantity
Capacity
ROOM
1
HAPPENS
N
Date
Schedule
Id
Free spaces
PROJECTION
HAS
N
Seat
Number
Status
Id
BILL
N
HAS
1
N
ADQUIRES
1
Cedula
First_surname
Second_surname
Name
Middle_name
First_name
CLIENT
Age
Birth_date
Phone_number
Username
Password
N
SHOWS
1
Length
Original_name
MOVIE
M
ACTS
N
Id
ACTOR
Name
Name
Id
N
DIRECTS
1
DIRECTOR
Id
Name
1
HAS
1
Code
Details
CLASSIFICATION
Age_rating


EMPLOYEE
Branch_id
Cedula
First_name
Middle_name
First_surname
Second_surname
Birth_date
Start_date
Phone_number
Username
Password
Age
BRANCH
Cinema_name
Province
District
Rooms_quant
ROOM
Branch_name
Id
Row_quantity
Column_quantity
Capacity
BILL
Client_id
Projection_id
Id
Detail
SEAT
Room_id
Number
Status
PROJECTION
Room_id
Movie_id
Id
Date
Schedule
Free_spaces
CLIENT
Cedula
First_name
Middle_name
First_surname
Second_surname
Birth_date
Phone_number
Username
Password
Age
MOVIE
Id
Director_id
Classificaction_ic
Original_name
Name
Length
CLASSIFICATION
Code
Details
Age_rating
DIRECTOR
Id
Name
ACTS
Movie_id
Actor_id
ACTOR
Id
Name

API

Controllers

ClientsController
+ get(): IEnumerable<Client>
+ Get(int cedula): Client
+ Login(string username, string password): Client
+ Post([FromBody] Client client): IActionResult
+ Put(int cedula, [FromBody] Client client): IActionResult
+ Delete(int cedula): ActionResult

EmployeesController
+ get(): IEnumerable<Employee>
+ Get(int cedula): Employee
+ Login(string username, string password): Employee
+ Post([FromBody] Employee employee): IActionResult
+ Put(int cedula, [FromBody] Employee employee): IActionResult
+ Delete(int cedula): IActionResult

ActorsController
+ Get(): IEnumerable<Actor>
+ Get_byId(int id): Actor
+ Get_byName(string name): Actor
+ Put(string name, [FromBody] Actor actor): ActionResult
+ Delete(string name): ActionResult

MoviesController
+ Get(): IEnumerable<Movie>
+ Get(int id): Movie
+ GetAll_Select(): IEnumerable<Movie>
+ Get_Select(string name): Movie
+ GetByName(string name): Movie
+ Post([FromBody] MovieCreation movie_stats)
+ PutByName(string name, [FromBody] MovieCreation movie_stats): IActionResult
+ Put(int id, [FromBody] MovieCreation movie_stats): IActionResult
+ DeleteByName(string name): IActionResult
+ DeleteById(int id): IActionResult

ClassificationsController
+ Get(): IEnumerable<Classification>
+ Get(string code): Classification
+ Post([FromBody] Classification classification): IActionResult
+ Put(string code, [FromBody] Classification classification): IActionResult
+ Delete(string code): ActionResult

BranchesController
+ Get(): IEnumerable<Branch>
+ Get(string cinema_name): Branch
+ Get_all_rooms(string cinema_name): IEnumerable<Movie>
+ Get(string cinema_name, string date): IEnumerable<Projection>
+ Get_projections_by_branch(string cinema_name): IEnumerable<Projection>
+ Get_all_projections_dates_byBranch(string cinema_name): List<string>
+ Post([FromBody] Branch branch): IActionResult
+ Put(string cinema_name, [FromBody] Branch branch): IActionResult
+ Delete(string cinema_name): ActionResult

RoomsController
+ Get(): IEnumerable<Room>
+ Get(int id): Room
+ Post([FromBody] Room room): IActionResult
+ Put(int room_id, [FromBody] Room room): ActionResult
+ Delete(int id): IActionResult

ActsController
+ Get(): IEnumerable<Acts>
+ GetActs_byMovieId(int movie_id): IEnumerable<Acts>
+ GetActs_byActorsId(int actor_id): IEnumerable<Acts>
+ GetActs_byActorsId(int actor_id): IEnumerable<Acts>

DirectorsController
+ Get(): IEnumerable<Director>
+ Get_byId(int id): Director
+ Get_byName(string name): Director
+ Put(string name, [FromBody] Director director): ActionResult
+ Delete(string name): ActionResult

ProjectionsControllers
+ Get(): IEnumerable<Projection>
+ Get(int id): Projection
+ Post(int covid, [FromBody] Projection projection): IActionResult
+ Put(int id, [FromBody] Projection projection): IActionResult
+ Delete(int id): IActionResult

SeatsController
+ Get(): IEnumerable<Seat>
+ Get(int projection_id, int number): Seat
+ Put(int projection_id, int number, [FromBody] Seat Seat): IActionResult
+ Delete(int projection_id, int number): IActionResult

Models

Client
+ cedula: int
+ first_name: string
+ middle_name: string
+ first_surname: string
+ second_surname: string
+ birth_date: Date
+ phone_number: string
+ username: string
+ password: string

Employee
+ cedula: int
+ first_name: string
+ middle_name: string
+ first_surname: string
+ second_surname: string
+ birth_date: Date
+ start_date: Date
+ phone_number: string
+ username: string
+ password: string
+ branch_id: string

Movie
+ id: int
+ classification_id: string
+ director_id: int
+ original_name: string
+ name: string
+ length: string

Classification
+ code: string
+ details: string
+ age_rating: int

Actor
+ id: int
+ name: string

Acts
+ movie_id: int
+ actor_id: int

Director
+ id: int
+ name: string

Branch
+ cinema_name: string
+ item: attribute
+ district: string

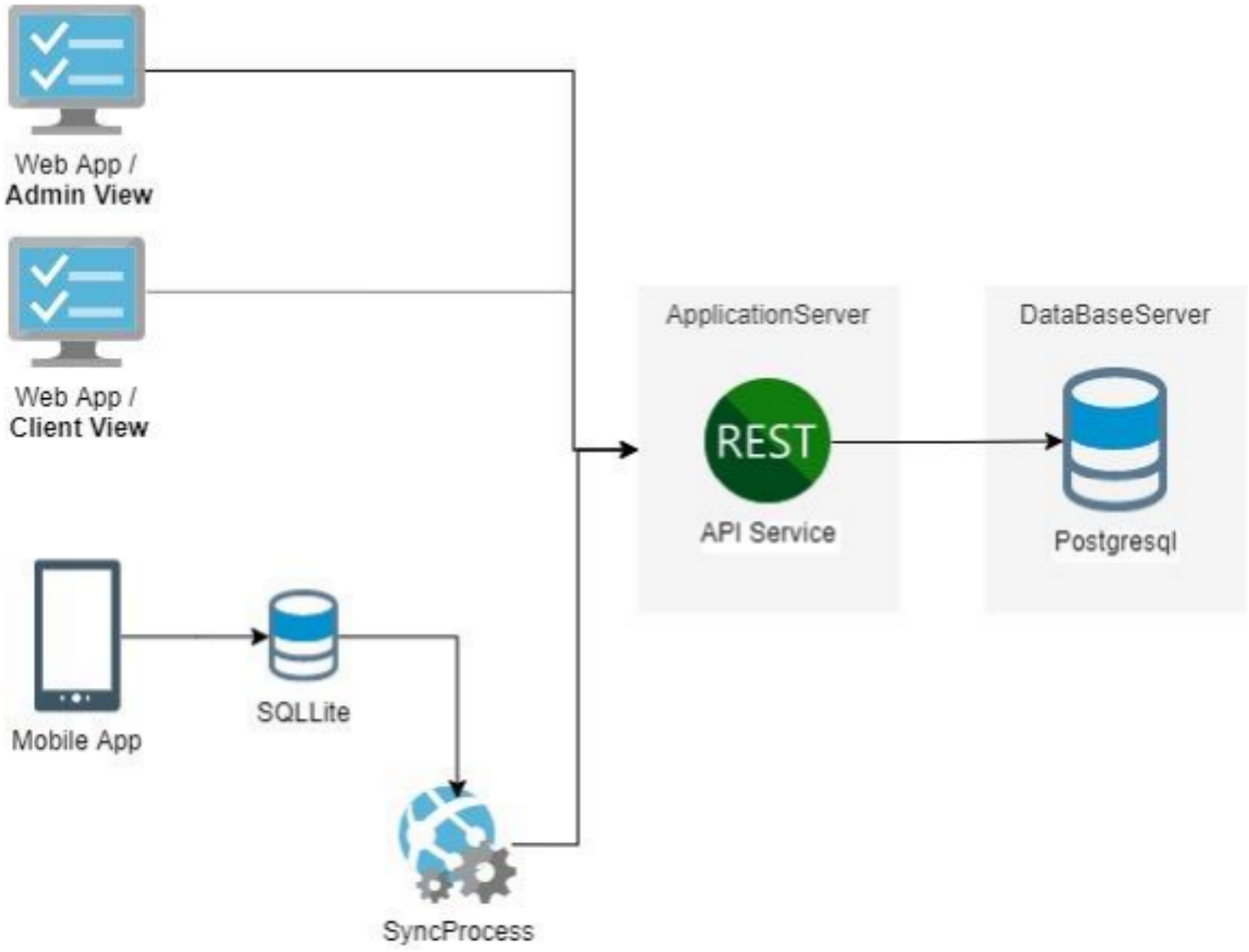
Room
+ branch_name: string
+ id: int
+ row_quantity: int
+ column_quantity: int

Projection
+ room_id: int
+ movie_id: int
+ id: int
+ date: Date
+ schedule: string

Seat
+ projection_id: int
+ number: int
+ status: string

Context

CRUDContext
+ DbSet<Employee> Employees
+ DbSet<Client> Clients
+ DbSet<Branch> Branches
+ DbSet<Seat> Seats
+ DbSet<Projection> Projections
+ DbSet<Movie> Movies
+ DbSet<Classification>
+ DbSet<Director> Directors
+ DbSet<Acts> Acts
+ DbSet<Actor> Actors
+ DbSet<ProjectionJSON> ProjectionJSONs


Web App /
Admin View
Web App /
Client View
ApplicationServer
REST
API Service
DataBaseServer
Postgresql
Mobile App
SQLLite
SyncProcess

Mapeo de Diagrama Relacional

a) *Mapeo de entidades fuertes:*

Las identidades fuertes identificadas fueron EMPLOYEE, BRANCH, ROOM, MOVIE, PROJECTION, ACTOR, DIRECTOR, CLASSIFICATION Y CLIENT, por lo que se creó una relación para cada una de estas entidades.

Para EMPLOYEE los atributos identificados fueron: branch_id, first_name, middle_name, first_surname, second_surname, birth_date, phone_number, username, password, como atributo derivado está age y como llave primaria **cedula**.

Para CLIENT se identificaron: first_name, middle_name, first_surname, second_surname, birth_date, phone_number, username, password, como atributo derivado está age y como llave primaria **cedula**.

Para BRANCH los atributos identificados fueron: province, district, como atributo derivado está room_quantity y como llave primaria **cinema_name.**

Para ROOM los atributos fueron: row_quantity, columna_quantity, como derivado está capacity y como llave primaria se utiliza **id** que es autoincremental.

Para MOVIE se encuentran: original_name, name, length y como llave primaria al igual que en la entidad anterior, se utiliza **id** que es autoincremental.

Para ACTOR es simplemente una llave primaria de **id** autoincremental con un atributo name.

Para DIRECTOR al igual que ACTOR se utiliza una llave primaria de **id** autoincremental y un atributo name.

Para CLASSIFICATION se identifica la llave primaria **code** y los atributos de details y age_rating.

Para PROJECTION están date, schedule, como atributo derivado free_spaces y como llave primaria **id** que es autoincremental.

b) *Mapeo de entidades débiles*

En el caso de las entidades débiles, se identificó SEAT y ACTS. Los atributos ACTS representan una llave compuesta, la cual está conformada por las llaves primarias de MOVIE y ACTOR, por otra parte para SEAT se identificó el atributo status y como llave compuesta se utilizan los atributos number y la llave primaria de PROJECTION.

c) *Mapeo de asociaciones binarias 1:1*

Existe una única relación 1:1, la cual es HAS entre MOVIE y CLASSIFICATION, como la participación total se encuentra del lado de MOVIE, se agrega a esta tabla una llave foránea (classification_id) la cual corresponde a la llave principal (id) de CLASSIFICATION. Es necesario que exista una clasificación para poder establecer una película.

d) *Mapeo de asociaciones binarias 1:N*

En este paso se incluye como llave foránea en la relación del lado N la llave primaria de la relación de lado 1. Por lo que se procede a realizar los siguientes pasos.

Para la relación de BRANCH a EMPLOYEE, se incluye branch_name como llave foránea en EMPLOYEE, que corresponde a la llave primaria de BRANCH.

Para la relación de BRANCH a ROOM, se incluye la llave primaria de BRANCH, cinema_name como llave foránea de ROOM (branch_name).

Para la relación de ROOM a PROJECTION, se incluye room_id como llave foránea en PROJECTION, que corresponde a la llave primaria de ROOM.

Para la relación de PROJECTION a SEAT, se debería incluir la llave primaria de PROJECTION en SEAT, no obstante ya está incluida por ser una entidad débil, por lo que se omite este paso.

Para la relación de MOVIE a PROJECTION, se incluye movie_id como llave foránea en PROJECTION, que corresponde a la llave primaria de MOVIE.

Para la relación de MOVIE a DIRECTOR, se incluye movie_id como llave foránea en DIRECTOR, que corresponde a la llave primaria de MOVIE.

Para la relación de PROJECTION a BILL, se incluye projection_id como llave foránea en BILL, que corresponde a la llave primaria de PROJECTION.

Para la relación de CLIENT a BILL, se incluye cedula como llave foránea en BILL, que corresponde a la llave primaria de CLIENT.

e) *Mapeo de asociaciones binarias N:M*

Existe una única relación N:M, la cual es ACTS, que corresponde a la relación entre MOVIE y ACTOR. Para esto se crea una nueva relación intermedia, para la cual su llave primaria es compuesta y está conformada por la llave primaria de MOVIE y la de ACTOR.

f) *Mapeo de atributos multivaluados*

Se omite pues no hay este tipo de atributos en la implementación propuesta.

Descripción del modelo

Se crearon todos los modelos con sus atributos a partir de cada relación del diagrama relacional con el objetivo de conectar la base de datos de postgres con el API usando Entity Framework en .NET.

Luego se creó una clase Context donde se definen las funciones necesarias del de acceso (CRUD) entre .NET y postgresql. Es en esta clase donde se hace uso de los querys para obtener la información necesaria de la base de datos.

Finalmente se crearon los controladores de cada entidad en los cuales se definen todos los HTTP request tanto para la gestión de la base de datos por parte del administrador como para el contenido de los clientes.

Estructuras Desarrolladas

Para el almacenamiento y organización de los datos se utilizaron tablas en una base de datos implementada haciendo uso de PostgreSQL, estas tablas tienen una estructura desarrollada para cada tipo de entidad del mini mundo tomando ciertos atributos que los caracterizan particularmente, todos tienen un comando *PRIMARY KEY*() que utiliza uno o varios atributos para definir la unicidad de cada instancia de la entidad, de esta forma se puede hacer una búsqueda sencilla y evita duplicados.

A continuación se muestra la forma de cada una de estas tablas:

★ **Actors**

id: int

name: varchar(30)

PRIMARY KEY(id)

★ **Directors**

id: int

name: string

PRIMARY KEY(id)

★ **Acts**

movie_id: int

actor_id: int

PRIMARY KEY(movie_id, actor_id)

★ **Classifications**

code: varchar(30)

details: varchar(30)

age_rate: int

PRIMARY KEY(code)

★ **Projections**

room_id: int

movie_id: int

id: int

date: date

schedule: string

PRIMARY KEY(id)

★ **Rooms**

branch_name: varchar(30)

id: int

row_quantity: int

column_ quantity: int

PRIMARY KEY(id)

★ **Clients**

cedula: int

first_name: varchar(30)

middle_name: varchar(30)

first_surname: varchar(30)

second_surname: varchar(30)

birth_date: date

phone_number: varchar(30)

username: varchar(30)

password: varchar(30)

PRIMARY KEY (cedula)

★ **Movies**

id: int

classification_id: string

director_id: int

original_name: string

name: string

length: string

PRIMARY KEY(id)

★ **Employees**

branch_id: varchar(30)

cedula: int

first_name: varchar(30)

middle_name: varchar(30)

first_surname: varchar(30)

second_surname: varchar(30)

birth_date: date

phone_number: varchar(30)

username: varchar(30)

password: varchar(30)

PRIMARY KEY (cedula)

★ **Seats**

id: int

projection_id: int

number: int

status: varchar(30)

PRIMARY KEY(projection_id, number)

Asimismo, las entidades cuentan con atributos que las relacionan con otras entidades, los cuales son denominados *foreign keys*. Estos atributos toman el valor de la *primary key* de la entidad a la que se está referenciando. Las relaciones presentes son las siguientes:

★ Referencia Employees → Branches

Interpretación: *Toma el valor de primary key de la sucursal en donde trabaja el empleado instanciado.*

```
ALTER TABLE "Employees"
ADD CONSTRAINT EMPLOYEE_BRANCH_FK FOREIGN KEY(branch_id)
REFERENCES "Branches"(cinema_name);
```

★ Referencia Rooms → Branches

Interpretación: *Toma el valor de primary key de la sucursal a donde pertenece la sala instanciada.*

```
ALTER TABLE "Rooms"
ADD CONSTRAINT ROOM_BRANCH_FK FOREIGN KEY(branch_name)
REFERENCES "Branches"(cinema_name);
```

★ Referencia Seats → Projections

Interpretación: *Toma el valor de primary key de la proyección a la cual pertenece la silla instanciada.*

```
ALTER TABLE "Seats"
ADD CONSTRAINT SEAT_PROJECTION_FK FOREIGN KEY(projection_id)
REFERENCES "Projections"(id);
```

★ Referencia Seats → Projections

Interpretación: *Toma el valor de primary key de la proyección a la cual pertenece la silla instanciada.*

```
ALTER TABLE "Seats"
ADD CONSTRAINT SEAT_PROJECTION_FK FOREIGN KEY(projection_id)
REFERENCES "Projections"(id);
```

★ Referencia Movies → Classifications

Interpretación: *Toma el valor de primary key de la clasificación asignada a la película instanciada.*

```
ALTER TABLE "Movies"
ADD CONSTRAINT MOVIE_DIRECTOR_FK FOREIGN KEY(classification_id)
REFERENCES "Classifications"(code);
```

★ Referencia Movies → Directors

Interpretación: *Toma el valor de primary key del director que realizó la película instanciada.*

```
ALTER TABLE "Movies"
ADD CONSTRAINT MOVIE_CLASSIFICATION_FK FOREIGN KEY(director_id)
REFERENCES "Directors"(id);
```

★ Referencia Acts → Movies

Interpretación: *Toma el valor de primary key de la película en la que protagoniza el actor que existe en la relación de su segundo atributo (actor_id)*

```
ALTER TABLE "Acts"
ADD CONSTRAINT ACTS_MOVIE_FK FOREIGN KEY(movie_id)
REFERENCES "Movies"(id);
```

★ **Referencia Acts → Actors**

Interpretación: *Toma el valor de primary key del actor que protagoniza la película que existe en la relación de su primer atributo (movie_id)*

```
ALTER TABLE "Acts"
ADD CONSTRAINT ACTS_ACTOR_FK FOREIGN KEY(actor_id)
REFERENCES "Actors"(id);
```

★ **Referencia Projections→ Rooms**

Interpretación: *Toma el valor de primary key de la sala en la cual se realizará la proyección instanciada.*

```
ALTER TABLE "Projections"
ADD CONSTRAINT PROJECTION_ROOM_FK FOREIGN KEY(room_id)
REFERENCES "Rooms"(id);
```

★ **Referencia Projections→ Movie**

Interpretación: *Toma el valor de primary key de la película que se va a dar en la proyección instanciada.*

```
ALTER TABLE "Projections"
ADD CONSTRAINT PROJECTION_MOVIE_FK FOREIGN KEY(movie_id)
REFERENCES "Movies"(id);
```

Algoritmos Desarrollados

Algoritmo de login

En el lado de la app web, se verifica el inicio de sesión ya sea de un empleado o cliente, de forma tal que se realiza un request de tipo GET que envía el *username* y *password* del usuario que quiere ingresar al sistema. Dentro del API se realiza el análisis, comparando estos valores con la tabla respectiva para cada entidad (empleado o cliente).

Si se logra tener una congruencia entre nombre y contraseña se le permite al usuario ingresar a las demás vistas de la aplicación, en caso contrario enviará un mensaje de “Nombre de usuario o contraseña incorrectos”.

Algoritmo de creación de tablas de ítems

Para llenar el componente item-holder con cada uno de los ítems para cierta categoría (empleados, sucursales, etc) se hace una llamada de GET según el URL. La respuesta después es introducida en un ciclo que genera un componente ítem dentro de item-holder con los datos de cada ítem.

El componente ítem posee una variable que identifica a cada posible item que puede contener, sin embargo sólo mostrará aquellas que estén condicionadas para el URL donde se encuentre el usuario.

Algoritmo de sincronización

Para el proceso de sincronización se crearon ciertas funciones que permiten actualizar cada una de las tablas que se necesitarán en la aplicación móvil, como lo

son la tabla de clientes, películas, salas y proyecciones. Cuando se desee acceder a algunas de las vistas que necesiten la información de algunas de estas tablas, se procederá a actualizar, realizando las consultas necesarias al API y agregandola a la base de datos, la información necesaria para mostrar la versión más actualizada. Además, se crea una opción para actualizar la base de datos en el momento en el momento que se desee.

Algoritmo de presentación de asientos

Para mostrar los asientos primero son llamados todos los asientos de la proyección seleccionada a través del API. Con el resultado se obtiene la información de todos los asientos de la proyección y de la sala, como cantidad de columnas y filas, cantidad de asientos y número y estado del asiento. Con esta información se procede a crear mediante un ngFor componentes seat-row-holder con el número de filas. Cada seat-row-holder entonces calcula según su número de fila la fracción de asientos que le corresponde, y los toma de la lista general de asientos. Ya con estos datos aislados procede a generar mediante un segundo ngFor los botones que abstraen las propiedades de los asientos para mostrar en la interfaz.

Algoritmo de parseo de proyecciones

Cuando se ha seleccionado una sucursal y un día específicos se manda al API un get request para todos los datos de todas las proyecciones para esas especificaciones. El API retorna una lista con todas las proyecciones, pero para ser mostradas en interfaz solamente es necesario lista con cada película y sus varios horarios. Por lo que cuando se recibe la lista es parseada e insertada en otra lista que se crea dinámicamente. Si la película no ha sido insertada toma toda la información y la añade a la lista, pero si la película ya está en la lista entonces solo

toma la información del horario, la sala y el id específico de la proyección. Repite este proceso hasta haber recorrido todas las proyecciones enviadas por el api. Una vez esta lista es completada es enviada al projection-holder que crea cada proyección acorde con los datos enviados. Adicionalmente cada proyección crea con otro ngFor botones especiales que mantienen los datos exactos de la hora, proyección y sala específica que representan.

Problemas Conocidos

Tamaño de interfaz

Cuando se está en la interfaz de administrador, si se minimiza el tamaño de la interfaz de vista, los cuadros de input para los distintos tipos de ítems modificables se sobreponen a los cuadros con la información de los ítems.

Problemas Encontrados

PUT sobre una tupla con valor ID igual a cero

Se encontró un error al intentar realizar un PUT a una entidad que contaba con un ID autogenerado y que tomaba como valor el 0. Al finalizar el protocolo request, se obtenía un valor de que el proceso se realizó de forma exitosa, no obstante la entidad no recibía ningún cambio.

Al realizar pruebas contundentes, curiosamente se observó que solo ocurría con las tuplas de id igual a cero, así que la solución fue omitir el cero para los ids y empezar todos desde el uno, luego de este cambio no hubo problema alguno.

Visualización de los ítems de películas, sucursales y proyecciones.

Se encontró un error al visualizar las películas, sucursales, proyecciones, cuando la cantidad de estas excede un cierto valor se comenzaban a repetir, por lo tanto no se mostraban todos los items.

Después de investigar se encontró que android reciclaba los ítems para ahorrar recursos. La solución fue que en caso de que ítem ya existiera se retornaba ese mismo, pero si era nulo, se debía crear un nuevo item y asignar los nuevos valores.

Comunicación de la aplicación móvil con el API .

Al momento de realizar la comunicación entre la aplicación móvil y el API para actualizar la base de datos en SQLite mostraba un error de conexión, por lo que el request no se efectuaba de manera exitosa.

Después de realizar una investigación del error, se encontró que se en el URL de la petición se debería cambiar localhost por 10.0.2.2, de esta forma android reconoce que la petición la debe realizar al localhost de la computadora.

Creación de matrices con componentes de angular

Para crear componentes de angular de manera iterativa se utiliza el comando *ngFor dentro del html . Sin embargo, esto restringe a un solo ciclo for, no puede haber ciclos anidados directamente. Por lo que como solución se crean componentes row. Estos son creados mediante el primer ciclo de for , y reciben los parámetros correspondientes para una fila en cada ciclo. Este componente row a su vez genera mediante un segundo ciclo for todos los componentes abstraídos como elemento único, para la fila que los genera. Esta implementación fue necesaria para generar las vistas de las películas en la página principal y la creación de asientos para la selección de asientos.

Generación de XML y pdf

Para la creación XML se usa la biblioteca “xmlbuilder” para crear el xml con los requerimientos de Hacienda. Para la generación del pdf se utiliza la biblioteca “jsPDF”, que permite generar pdfs a partir de objetos HTML.

CORS en el servidor de API

Al intentar hacer requests HTTP el servidor se negaba a responder por CORS.

Para solucionar este problema fue necesario implementar un proxy del lado de Angular que permitiera realizar estos requests a server. Esto se hace mediante el archivo de tsconfig.json

Conclusiones

- La mayoría de los sistemas de reporte necesitan de NuGet packages disponibles en las bibliotecas de .NET de Visual Studio.
- Es posible desarrollar aplicaciones Web bastante robustas con Angular ya que posee una gran cantidad de funcionalidades.
- Para agilizar el desarrollo sin necesidad de instalar ambientes u otras aplicaciones es posible trabajar con Azure y otras plataformas que permiten subir sitios web gratuitos.
- El entorno de desarrollo de Android para desarrollar una aplicación móvil permite crear aplicaciones móviles de manera sencilla y eficiente.
- SQLite facilita el desarrollo de aplicaciones móviles que deban integrar una base de datos para su funcionamiento.
- Existen gran cantidad de plantillas de código abierto en angular bootstrap. Permiten facilitar un diseño rápido y estético de una página web.
- Se pueden crear matrices de componentes en angular, pero para ello son necesarios tres componentes. Un padre, un componente como abstracción de fila y un componente de abstracción para cada elemento.

Recomendaciones

- Es recomendable que al desarrollar un API en .NET, utilizar Visual Studio para tener acceso a mayor funcionalidad como sistemas de reporte.
- Antes de comenzar un proyecto con Angular es importante hacer una investigación inicial puesto que es una plataforma con un grado de complejidad importante, producto de sus muchas funcionalidades.
- Utilizar sitios web gratuitos para hospedar APIs se puede permitir un desarrollo en paralelo de una aplicación web que lo requiera, sin la necesidad de instalar los ambientes necesarios para ejecutar el API.
- Se recomienda utilizar el entorno de desarrollo Android para desarrollar aplicaciones móviles por las diferentes facilidades y funcionalidades que ofrece.
- Se recomienda utilizar SQLite como sistema de gestión de bases de datos para desarrollar aplicaciones móviles.
- Utilizar plantillas preconstruidas de angular bootstrap para generar rápidamente aplicaciones con interfaces llamativas.
- Utilizar un sistema de tres componentes cuando se desea hacer un ngFor dentro de otro. Por ejemplo para la creación de matrices de componentes.

Bibliografía

[1] "Entity Framework Core Tutorials", *Entityframeworktutorial.net*, 2021. [Online]. Available: https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx. [Accessed: 29- Sep- 2021].

[2] C. Libre, “Creación de xml Factura Electrónica,” 2018. [Online]. Available: https://github.com/CRLibre/API_Hacienda/wiki/Creaci%C3%B3n-de-xml-Factura-Electr%C3%B3nica [Accessed: 9- Oct- 2021].

[3] "PostgreSQL 13.4 Documentation", PostgreSQL Documentation, 2021. [Online]. Available: https://www.postgresql.org/docs/13/. [Accessed: 17- Oct- 2021].

[4] "Documentation | Android Developers", Android Developers, 2021. [Online]. Available: https://developer.android.com/docs. [Accessed: 17- Oct- 2021].