Instituto Tecnológico de Costa Rica

Ingeniería en Computadores

Bases de Datos

Proyecto 2 GymTEC

Estudiantes:

Mario Gudiño Rovira - 2017106391

Ignacio Carazo Nieto -2017090425

Joseph Jiménez – 2016133677

Profesor:

Luis Diego Noguera

I Semestre 2021

ÍNDICE

Descripción de las estructuras de datos desarrolladas

Las estructuras de datos utilizadas en este proyecto fueron principalmente las tablas. A partir de MongoDB se logra crear una base de datos no relacional que permite guardar la información del cliente ver figura 1 haciendo uso del formato texto llamado JSON .

Figura 1. Datos de cliente guardados en MongoDb

SQL Server permite crear entidades con sus llaves primarias, limitaciones, llaves foráneas, a su vez permite el uso de varios tipos de datos de gran utilidad como lo es VARCHAR, INT, entre otros. Como son varias tablas se mencionan como ejemplo las tablas Gym, Employee y Machine.

```sql
CREATE TABLE Gym(
        name VARCHAR(10) NOT NULL,
        phoneNumber INT NOT NULL,
        spaActive INT,
        storeActive INT,
        openingDate VARCHAR(11),
        businessHours VARCHAR(10) NOT NULL,
        maxCapacity INT NOT NULL,
        adminID INT NOT NULL
);

ALTER TABLE Gym
ADD CONSTRAINT pk_gym_name PRIMARY KEY (name);

ALTER TABLE Gym
ADD CONSTRAINT fk_gym_admin FOREIGN KEY (adminID)
REFERENCES Employee (id)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

Figura 2.Tabla Gym

```sql
CREATE TABLE Machine(
        serialNumber INT NOT NULL,
        brand VARCHAR(20),
        cost INT,
        typeID INT,
        gymName VARCHAR(10)
);

ALTER TABLE Machine
ADD CONSTRAINT pk_machine_snumber PRIMARY KEY (serialNumber);

ALTER TABLE Machine
ADD CONSTRAINT fk_machineType_ID FOREIGN KEY (typeID)
REFERENCES MachineType (id)
ON DELETE CASCADE
ON UPDATE CASCADE;

ALTER TABLE Machine
ADD CONSTRAINT fk_gym_name FOREIGN KEY (gymName)
REFERENCES Gym (name)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

Figura 3. Tabla Employee

```sql
CREATE TABLE Employee(
        id INT NOT NULL,
        name VARCHAR(20) NOT NULL,
        primaryLastName VARCHAR(20) NOT NULL,
        secondaryLastName VARCHAR(20) NOT NULL,
        province VARCHAR(20) NOT NULL,
        canton VARCHAR(20) NOT NULL,
        district VARCHAR(20) NOT NULL,
        email VARCHAR(40) NOT NULL,
        password VARCHAR(60) NOT NULL,
        numberOfClasses INT,
        laboredHours INT,
        salary INT,
        role VARCHAR(20),
        spreadsheetTypeID INT,
        gymName VARCHAR(20)
);


ALTER TABLE Employee
ADD CONSTRAINT pk_emp_id PRIMARY KEY (id);

ALTER TABLE Employee
ADD CONSTRAINT unique_emp_email UNIQUE (email);
```

Figura 4. Tabla Machine

Por medio SQL server se permite definir el nombre de estas tablas, los atributos con su respectivo tipo, sus respectivos identificadores y las referencias necesarias hacia las otras tablas.

En estas tablas se pueden contener los datos, los datos se organizan con arreglo a un formato de filas y columnas, cada fila con su registro único; En el caso de mongo debe sujetarse a las reglas de sintaxis que usa la notación de objeto de JavaScript, con eso es valido para guardar los datos necesarios

Descripción de los SP, Triggers y Vistas

Se utilizaron varios stored procedures para manejar la lógica de la aplicación, el formato CRUD fue aplicado para la gestión de las tablas, con SCOPE_IDENTITY() se captura el valor de la identidad. Para crear, hace uso de la sentencia INSERT. Al leer se obtiene los registros de la tabla basado en la llave primaria especificada, se actualiza haciendo uso de la sentencia UPDATE en la tabla, basado en la llave primaria para un registro indicado, en el caso de borrar se elimina una fila especificada en la cláusula WHERE.

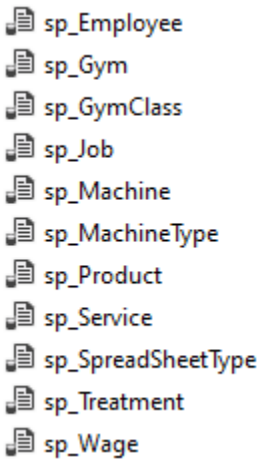
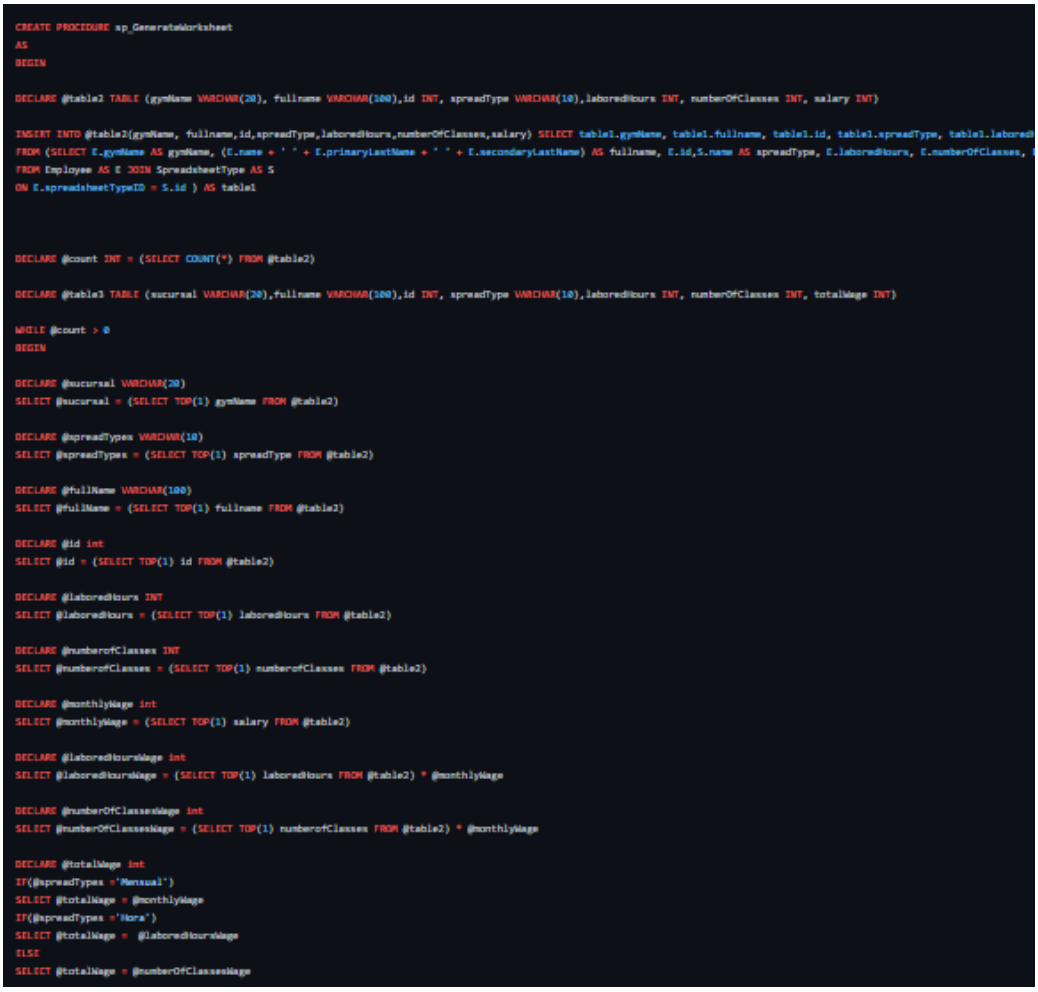


Figura 5. Stored Procedures con CRUD

Para la especificación se solicitó un sp que permite al administrador saber el pago que debe realizar a todos los empleados del gimnasio. Para calcular el pago de los tipos de planilla en los Mensuales: se consulta el monto indicado en el salario en la tabla empleado. Para Horas: se consulta las horas laboradas en la tabla empleado y se multiplica por el monto en el salario del empleado; Para el monto de la clase: se consulta las clases impartidas por el empleado y se multiplica por el monto en el salario del empleado. Por último se agrupa por sucursal, número de cédula, nombre completo del empleado, número de clases impartidas/horas laboradas y monto a pagar.



```
CREATE PROCEDURE sp_GenerateWorksheet
AS
BEGIN

DECLARE @table2 TABLE (gymName VARCHAR(20), fullname VARCHAR(100),id INT, spreadType VARCHAR(10),laboredHours INT, numberOfClasses INT, salary INT)

INSERT INTO @table2(gymName, fullname,id,spreadType,laboredHours,numberOfClasses,salary) SELECT table1.gymName, table1.fullname, table1.id, table1.spreadType, table1.laboredH
FROM (SELECT E.gymName AS gymName, (E.name + ' ' + E.primaryLastName + ' ' + E.secondaryLastName) AS fullname, E.id,S.name AS spreadType, E.laboredHours, E.numberOfClasses, E
FROM Employee AS E JOIN SpreadsheetType AS S
ON E.spreadsheetTypeID = S.id ) AS table1



DECLARE @count INT = (SELECT COUNT(*) FROM @table2)

DECLARE @table3 TABLE (sucursal VARCHAR(20),fullname VARCHAR(100),id INT, spreadType VARCHAR(10),laboredHours INT, numberOfClasses INT, totalWage INT)

WHILE @count > 0
BEGIN

DECLARE @sucursal VARCHAR(20)
SELECT @sucursal = (SELECT TOP(1) gymName FROM @table2)

DECLARE @spreadTypes VARCHAR(10)
SELECT @spreadTypes = (SELECT TOP(1) spreadType FROM @table2)

DECLARE @fullName VARCHAR(100)
SELECT @fullName = (SELECT TOP(1) fullname FROM @table2)

DECLARE @id int
SELECT @id = (SELECT TOP(1) id FROM @table2)

DECLARE @laboredHours INT
SELECT @laboredHours = (SELECT TOP(1) laboredHours FROM @table2)

DECLARE @numberofClasses INT
SELECT @numberofClasses = (SELECT TOP(1) numberofClasses FROM @table2)

DECLARE @monthlyWage int
SELECT @monthlyWage = (SELECT TOP(1) salary FROM @table2)

DECLARE @laboredHoursWage int
SELECT @laboredHoursWage = (SELECT TOP(1) laboredHours FROM @table2) * @monthlyWage

DECLARE @numberOfClassesWage int
SELECT @numberOfClassesWage = (SELECT TOP(1) numberofClasses FROM @table2) * @monthlyWage

DECLARE @totalWage int
If(@spreadTypes ='Mensual')
SELECT @totalWage = @monthlyWage
If(@spreadTypes ='Hora')
SELECT @totalWage =  @laboredHoursWage
ELSE
SELECT @totalWage = @numberOfClassesWage
```

Figura 6. Stored Procedures para Generacion de Planilla

En la especificación se ha solicitado el uso de un trigger para gestionar los tratamientos que brinda los spas, se hizo el caso en cuanto se inserta una entidad gym, se revisa si esta posee el spa activo si es así, se ejecuta un stored procedure que obtiene los tratamientos por default y los inserta en el gimnasio respectivo, de la misma manera se hace si la entidad del gimnasio es actualizada.

```
CREATE TRIGGER TriggerTreatmentCreate
    ON  Gym
    AFTER INSERT,UPDATE
AS
    BEGIN
        SET NOCOUNT ON;
        DECLARE @spaActive INT;
        DECLARE @name VARCHAR(40);
        SELECT TOP 1 @spaActive = i.spaActive, @name = i.name
        FROM INSERTED i;
                IF(@spaActive=1)
                BEGIN
                        EXEC sp_TreatmentCreate @name = 'Masaje Relajante', @gymName = @name;
                        EXEC sp_TreatmentCreate @name = 'Masaje Descarga', @gymName = @name;
                        EXEC sp_TreatmentCreate @name = 'Masaje Sauna', @gymName = @name;
                        EXEC sp_TreatmentCreate @name = 'Bano Vapor', @gymName = @name;
                END
    END;
```

Figura 7. Trigger para Generación de Tratamientos de Spa

Descripción detallada de la arquitectura desarrollada

Tal y como se aprecia en la figura 8 se puede ver el diagrama de arquitectura del proyecto. Primeramente se tiene una aplicación web la cual está conformada por herramientas como Angular, Bootstrap, Html5, entre otras. Para que la comunicación sea exitosa entre la aplicación web y el rest api hecho en C# se deben de enviar y recibir los datos en formato Json, donde una vez dentro de cada componente por separado estos pueden serializar la información a datos reconocidos por sus respectivos lenguajes. Una vez terminada esta comunicación el rest api procede a solicitar algún query alguna de las base de datos respecto a la petición generada por la aplicación web. La base de datos no relacional la cual utiliza mongoDB almacena los datos referentes al cliente por lo que el api tiene contacto con esta a la hora de registros y login. Utilizando Microsoft SQL Server para la base de datos relacional se tiene almacenado todo lo correspondiente a las sucursales, servicios, tratamientos, máquinas, entre otras entidades.

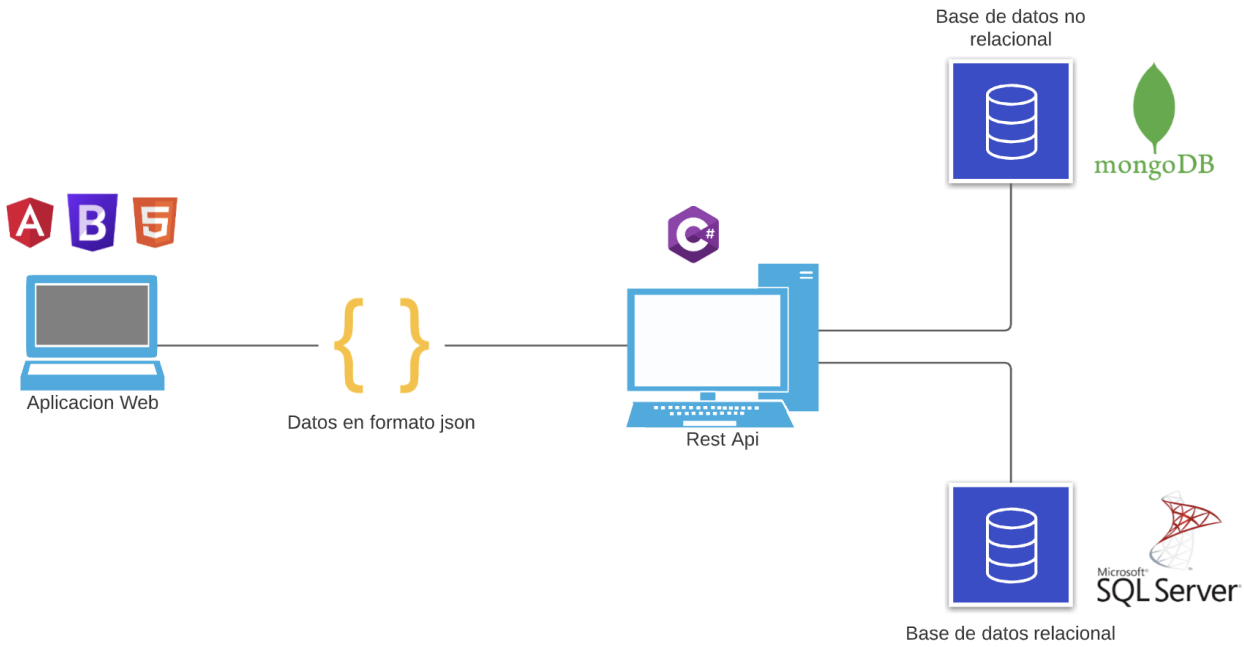


Figura 8. Diagrama de arquitectura

Problemas Conocidos

Para el acceso a la base de datos en Mongodb no se usó la terminación api/Client/user/password, sino, que se usa la cédula, api/Client/idCed para así lograr acceder a la data guardada en la tabla que almacena los clientes.

Problemas Encontrados

Hay algunos controles de restricciones respecto a borrar entidades que son foreign key de alguna otra entidad. Estas se pueden borrar y en la otra entidad aun aparece como si existiera. El api de la base de datos relacional no pudo desplegarse en Azure.

Conclusiones

- Al hacer uso de los stored procedures, triggers y views, mejora la capacidad de lectura y mantenibilidad de las sentencias SQL durante la gestión de los datos.

- Por medio de la creación de 2 API 's se logra acceder a bases de datos diferentes y que la aplicación haga uso de estos datos, por medio de una página web se logra exponer

las diferentes funcionalidades al usuario. Por medio del trabajo en equipo, buena comunicación, apoyo entre todas las partes, se logra plantear una solución precisa, objetiva y válida para dar solución al problema a medida que fue posible.

Recomendaciones

- Comunicarse con los miembros del equipo constantemente siempre va a ser importante, se logra estar acuerdo entre las partes con más frecuencia, los imprevistos que suceden dan lugar a reorganizaciones de tiempo, de carga de trabajo, para así lograr sacar con éxito la tarea.
- Es importante romper la costumbre de algunos programadores de solo empezar a codificar. En este caso cada integrante del grupo realizó una investigación sobre las tecnologías, se planificó y se reparte la carga dependiendo del conocimiento fuerte de cada uno .

Bibliografía

- Mark Otto, a., 2021. Bootstrap. [online] Getbootstrap.com. Available at: <https://getbootstrap.com> [Accessed 3 March 2021].
- GitBooks. 2021. Arquitectura de una API REST · Desarrollo de aplicaciones web. [online] Available at: <https://juanda.gitbooks.io/webapps/content/api/arquitectura-api-rest.html> [Accessed 12 March 2021].
- Au-Yeung, J., Brecht, D., Shafer, D., Goldstein, R., Donovan, R. and Aladdin, M., 2021. Best practices for REST API design - Stack Overflow Blog. [online] Stack Overflow Blog. Available at: <https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/> [Accessed 23 March 2021].

- Developers, A., 2021. Security with HTTPS and SSL. [online] Android.com. Available at: <https://developer.android.com/training/articles/security-ssl> [Accessed 20 March 2021].
- Rubiyath, A., 2021. Android Volley Tutorial – Making HTTP GET, POST, PUT | Alif's Blog. [online] Itsalif.info. Available at: <https://www.itsalif.info/content/android-volley-tutorial-http-get-post-put> [Accessed 25 March 2021].
- Software Engineer specializing in native Android Development with Java, N., 2021. Making a simple GET and POST request using Volley (Beginners Guide). [online] Medium. Available at: <https://nabeelj.medium.com/making-a-simple-get-and-post-request-using-volley-beginners-guide-ee608f10c0a9> [Accessed 2 April 2021].
- Medic, M., 2021. Creando usando procedimientos almacenados CRUD. [online] SQL Shack - articles about database auditing, server performance, data recovery, and more. Available at: <https://www.sqlshack.com/es/creando-usando-procedimientos-almacenados-crud/> [Accessed 17 June 2021].

Anexos

Link al repositorio en GitHub

https://github.com/IgnacioCarazo/GymTEC