Realizado por VICTOR ELIAS BARRERA FLOREZ – Desarrollador Backend

1. **Planteamiento del problema o historia de caso de uso.**

    - En le planteamiento del ejerció lo primero que identifique el objetivo principal que debe cumplir es una caja registradora.
    - Implementar un servicio api REST el cual debe responder 6 rutas o endpoints.
        1. Caja base a la caja
        2. Vaciar caja
        3. Estado de caja
        4. Realizar un pago
        5. Ver registro de logs de eventos
        6. Saber estado de la caja según fecha y hora determinado
    - Identifique que el plus del servicio REST debe ser el cambio mas optimo para dar el cambio de vuelto.
    - Identifique que servicio REST si no tiene para realizar cambios debe devolver una descripción donde informe el caso.
    - Identifique que servicio REST tiene que tener una ruta donde devuelva el total de dinero que tienen la caja especificando por denominación de cuantas monedas/billetes.
    - Identifique que servicio REST tiene que tener una ruta log donde muestre todo el movimiento que se realizan en la caja con los siguientes datos: fecha y hora, monto salió o entro.
    - Identifique que el servicio REST debe mostrar el histórico de la caja recreando el estado actual mostrando la suma de los ventos entrada y salida de dinero indicando la fecha y hora.
    - Identifique que el servicio REST debe tener la opción de retirar el dinero de la caja quedando el saldo 0.

2. **Identificación de lenguaje o framework.**

    Para este proyecto tome la decisión desarrollar el ejercicio con un framework de PHP llamado laravel en la versión 5.5.* y utilizar como motor de almacenamiento MySQL 5.7 e implemente Docker versión 20.10.2, docker-compose versión 1.27.4

3. **Información de diseño.**

    - **Caja base a la caja**: se plantea almacenar estado actual de la caja en la tabla denominaciones para evitar duplicidad de información en la base de datos ya que en el ejercicio existe una sola caja.

- **Vaciar caja:** se plante primero guardar la información de denominación y existencia que hay actualmente en caja y todos los movimientos para luego se coloca la existencia 0 y se almacena en log el tipo de movimiento de “vaciarCaja”.
- **Estado de caja:** se plante mostrar los datos actuales de la tabla denominaciones.
- **Realizar un pago:** se plantea primero valida si la entrada es correcta luego se verifica que el monto ingresado se mayor o igual al cobro si es así se procede a calcular el cambio. Si el calculo de cambio es exitoso entonces se procede a actualizar los valores en la caja.
- **Ver registro de logs de eventos:** se plantea mostrar todos los movimiento y eventos realizados.
- **Saber estado de la caja según fecha y hora determinado:** se plantea mostrar según la fecha y hora ingresa todos los movimientos realizados en el sistema.

4. La estructura de nuestro código es (MVC)

Aborde el problema de la aplicación según la naturaleza del Framework de laravel, aunque este proyecto no se ha requerido la realización de una vista (Plantilla HTML) para poder observar los datos, sino que me he valido de la herramienta postman para el testeo de nuestra aplicación si hemos implementado el uso de Modelo (Base de Datos) y Controladores.

Use migraciones para la creación de las tablas en las que se registraran los datos

Use un modelo para acceder a nuestros datos

Use controladores para comunicarnos con nuestro modelo

De hecho, toda la lógica de la aplicación se encuentra dentro de los controladores que además de acceder a los datos, realiza algunos cálculos y operaciones específicas. Tales como son, calcular el vuelto, saber cuantas monedas o billetes de la denominación correspondiente.

Patrones de Diseño Utilizados

En este proyecto se ha aplicado el patrón de diseño **Factory Method**. Como se puede observar se definen varios Factory para que nuestra aplicación no tenga que comunicarse.

```php
<?php

use Faker\Generator as Faker;

$factory->define(App\Movimiento::class, function (Faker $faker) {
    return [
        //
    ];
});
```

Para crear fabricas de objetos que nos permita obtener datos falsos para poder hacer pruebas con nuestra aplicación.

```php
<?php

use App\Denominacione;
use Illuminate\Database\Seeder;

class DenominacionesSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $denominaciones = [
            "cienmil" => 100000,
            "cincuentamil" => 50000,
            "veintemil" => 20000,
            "diezmil" => 10000,
            "cincomil" => 5000,
            "dosmil" => 2000,
            "mil" => 1000,
            "quinientos" => 500,
```

```
            "doscientos" => 200,
            "cien" => 100,
            "cuncuenta" => 50,
        ];

        foreach ($denominaciones as $key => $value) {
            $denominacion = new Denominacione(["nombre" => $key, "valor" => $valu
e, "existencia" => 0]);
            $denominacion->save();
        }
    }
}
```

Otro patrón que utilizamos es el patrón **Observer Patter,** el cual nos permite saber cuando se ejecuta una acción en una clase todo va a un log de eventos donde se registran los diferentes movimientos de la caja.

5. **Implementar para montar el proyecto.**

   Ubicar los 2 archivos (Dockerfile, docker-compose.yaml) en la raíz de la carpeta donde va estar almacenada el proyecto.

   Realizar raíz del de la carpeta git clone https://github.com/victorcel/prueba-tecnica-merkeo-backend

   Creamos un archivo cat docker-compose/nginx/mercadeo.conf con la siguiente información:

```
server {
    listen 80;
    index index.php index.html;
    error_log  /var/log/nginx/error.log;
    access_log /var/log/nginx/access.log;
    root /var/www/public;
    location ~ \.php$ {
        try_files $uri =404;
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        fastcgi_pass app:9000;
        fastcgi_index index.php;
        include fastcgi_params;
```

```
        fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }
    location / {
        try_files $uri $uri/ /index.php?$query_string;
        gzip_static on;
    }
}
```

En le archivo se configura nginx para escuchar en el puerto 80 y usar index.php como la página predeterminada y se establece la raíz del documento /var/www/public, y luego configurará Nginx para usar el servicio de la aplicación en el puerto 9000 para procesar archivos * .php.

Configuración del .env archivo de la aplicación
cd prueba-tecnica-merkeo-backend
cd .env.example .env

Modificar.env para corregir la información

```
DB_CONNECTION=mysql
DB_HOST=db
DB_PORT=3306
DB_DATABASE=dc_pruebaMerqueo
DB_USERNAME=merqueo
DB_PASSWORD=password
```

Ejecute la instalación del composer para instalar las dependencias de la aplicación

```
chmod -R 777 storage .env
docker-compose up -d
docker-compose exec app ls -l
docker-compose exec app composer install
docker-compose exec app php artisan key:generate
docker-compose exec app php artisan migrate --seed
```