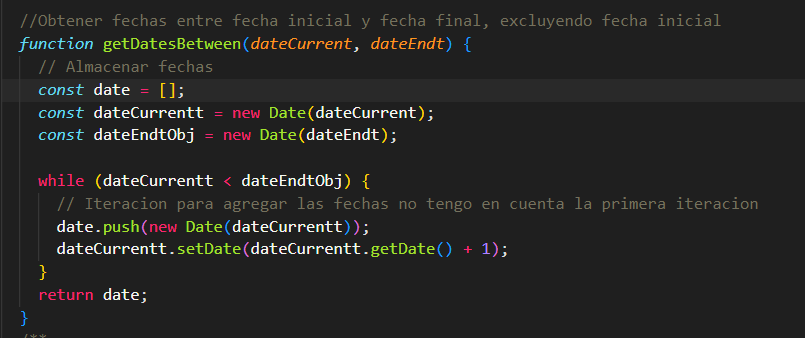
Se requiere entonces para la solución un algoritmo que permita:

* Recibir una fecha inicial y una fecha final.
* Validar si el valor que se están recibiendo en fecha inicial es un Object date y si fecha final es un String de fechas.
* Transformar el valor de fecha final en un formato de fecha y en un Object date.
* Calcular el rango de fechas entre fecha inicial y fecha final.
* Identificar los días sábado, domingo y festivo si están dentro del rango de fecha suministrado.
* Imprimir los días hábiles disponibles.

En esta solución hacemos uso de la programación estructurada, esto consiste en dividir en una serie de funciones más pequeñas y manejables el problema, y cada función realiza una tarea específica. Las funciones pueden ser llamadas (invocadas) desde otras funciones para realizar una secuencia de operaciones que trabajan juntas para resolver un problema.

1.La función “**getDatesBetween()”** me permite calcular el rango de fechas entre dos parámetros, el primero corresponde a la fecha inicial **“dateCurrent”**, y el segundo a la fecha final **“dateEndt”**.

Paso a paso de la funcionalidad:



* Se crea un array vacío llamado “**date”** para almacenar las fechas calculadas.
* Se crea un objeto “**Date”** llamado “**dateCurrentt”** a partir de la fecha “**dateCurrent”**. Esto se hace para poder manipular fácilmente la fecha actual durante el proceso de iteración.
* Se crea otro objeto “**Date”** llamado “**dateEndtObj”** a partir de la fecha “**dateEndt”**. Esto también se hace para poder manipular la fecha final durante el proceso de iteración.
* Se inicia un bucle “**while”** que se ejecutará mientras “**dateCurrentt”** sea menor que “**dateEndtObj”**. Esto garantiza que se agregarán fechas hasta llegar a la fecha final, pero sin incluir la fecha inicial.
* Dentro del bucle, se agrega una copia de “**dateCurrentt”** al array “**date”** usando “**date.push(new Date(dateCurrentt))”**. Esto se hace para evitar que todas las fechas en el array **date** se refieran a la misma instancia de objeto “**Date”**.
* Luego, se aumenta la fecha actual en un día utilizando “**dateCurrentt.setDate(dateCurrentt.getDate() + 1)”.** Esto permite avanzar al siguiente día en cada iteración del bucle.
* El bucle se repite hasta que “**dateCurrentt”** sea igual o mayor que “**dateEndtObj”**. En ese momento, el bucle se detiene y la función devuelve el array “**date”** con todas las fechas calculadas.

2. La función “**isWeekendOrHolday ()”** me permite verificar si dentro del rango de fechas suministrado hay algún día que sea un sábado, un domingo o un día festivo, esta función recibe un objeto “**Date”** llamado “**date”**.

Paso a paso de la funcionalidad:

Texto

Descripción generada automáticamente

* Se utiliza “**date.getDay()”** para obtener el día de la semana de la fecha “**date”**. El método **getDay()** devuelve un valor numérico del 0 al 6, donde 0 representa el domingo, 1 representa el lunes y así sucesivamente hasta 6 que representa el sábado. Se almacena este valor en la variable **“day”**.
* Luego, se realiza una comprobación con una declaración **“if”** para verificar si el valor de **“day”** es igual a 0 (domingo) o igual a 6 (sábado). Si se cumple alguna de estas condiciones, significa que la fecha es un fin de semana, por lo que la función devuelve “**true**”, indicando que es un día no laboral.
* Si el valor de **“day”** no es 0 ni 6, entonces la función continuará ejecutando el siguiente bloque de código.
* La función tiene un array llamado “**holidays”** que contiene fechas específicas de días festivos. Estas fechas están definidas para el año 2023. Cada elemento del array es un objeto **“Date”** que representa una fecha festiva particular, como por ejemplo el 7 de agosto, el 21 de agosto etc.
* A continuación, se realiza un bucle **“for...of”** para recorrer el array **“holidays”**. En cada iteración, se compara si la fecha **“date”** es igual a alguna de las fechas festivas almacenadas en el array **“holidays”**. Esto se hace utilizando **“date.toDateString() === holiday.toDateString()”**. El método “**toDateString()”** devuelve una cadena que representa la fecha en un formato específico (por ejemplo**, "Tue Aug 07 2023"**) que permite comparar fácilmente las fechas.
* Si la fecha **“date”** coincide con alguna de las fechas festivas en el array **“holidays”**, la función devuelve **“true”**, indicando que es un día festivo.
* Si no se cumple ninguna de las condiciones anteriores (es decir, no es fin de semana ni día festivo), la función devuelve false, indicando que es un día laboral normal.

3. La función “**getWorkingDays()”** me permite recibir dos parámetros, el primero corresponde a la fecha inicial **“dateCurrent”**, y el segundo a la fecha final **“dateEndt”**. Con estos dos valores calculo los días hábiles disponibles entre las dos fechas.

Paso a paso de la funcionalidad:

Texto

Descripción generada automáticamente

* Se llama a la función **“getDatesBetween(dateCurrent, dateEndt)”** para obtener un array de todas las fechas entre **“dateCurrent”** y **“dateEndt”**, excluyendo la fecha inicial pero incluyendo la fecha final. La función **“getDatesBetween”** es aquella que previamente se ha explicado y devuelve todas las fechas dentro de ese rango.
* El array de fechas obtenido en el paso anterior se almacena en la variable **“dates”**.
* Se utiliza el método **“filter()”** en el array **“dates”** para crear un nuevo array llamado **“workingDays”** que solo contiene las fechas que no son fines de semana ni días festivos. Esto se logra utilizando la función **“isWeekendOrHoliday”** como criterio para filtrar las fechas. La función **“filter()”** recorre cada fecha en **“dates”** y mantiene solo aquellas fechas para las cuales la función **“isWeekendOrHoliday”** devuelve false, lo que significa que no son fines de semana ni días festivos.
* Finalmente, la función **“getWorkingDays”** devuelve el número de días laborales (días hábiles) encontrados en el array **“workingDays”**, que es la cantidad de fechas que pasaron el filtro y no son fines de semana ni días festivos.

4. La función “**isDateFormat()”** me permite recibir un parámetro y verificar si dicho parámetro corresponde un formato de fechas.

Paso a paso de la funcionalidad:

Texto

Descripción generada automáticamente

* La función tiene una variable **“dateRegex”** que contiene una expresión regular. La expresión regular es **/^\d{1,2}\/\d{1,2}\/\d{4}$/.**
* La expresión regular **^\d{1,2}\/\d{1,2}\/\d{4}$** está compuesta por los siguientes elementos:

**^**: Representa el inicio de la cadena.

**\d{1,2}**: Representa uno o dos dígitos numéricos.

**\/**: Representa el carácter de barra inclinada (slash) "/", que es utilizado como separador en el formato de fecha.

**\d{1,2}**: Representa uno o dos dígitos numéricos.

**\/**: Representa otro carácter de barra inclinada (slash) "/".

**\d{4}**: Representa cuatro dígitos numéricos.

**$**: Representa el final de la cadena.

En resumen, la expresión regular verifica que la cadena tenga el siguiente formato: "dd/mm/aaaa", donde "dd" es el día del mes (uno o dos dígitos), "mm" es el mes (uno o dos dígitos) y "aaaa" es el año (cuatro dígitos).

* Luego, se utiliza el método **“test ()”** de la expresión regular para verificar si la cadena value coincide con el patrón definido por la expresión regular. El método **“test()”** devuelve true si la cadena coincide con la expresión regular y false si no coincide.
* La función **“isDateFormat”** devuelve el resultado de la verificación, es decir, true si la cadena value tiene un formato de fecha válido y false si no tiene el formato correcto.

5. La función “**changeDateFormat ()”** me permite recibir un parámetro tipo String con un formato de fecha **“10/8/2023”** por lo cual lo que haremos será transformar este formato al siguiente ejemplo: **“2023-8-21”**.

Paso a paso de la funcionalidad:

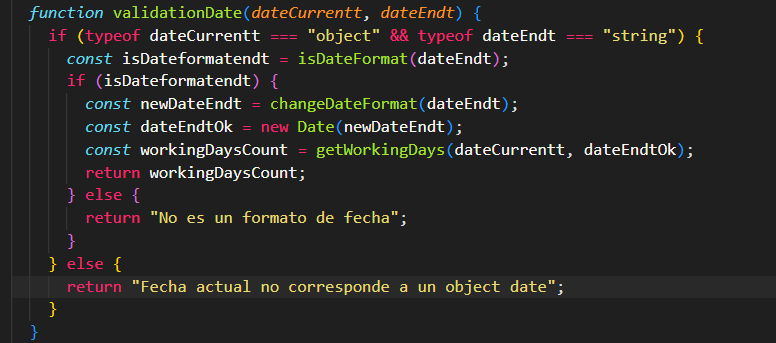
Texto

Descripción generada automáticamente

* La función comienza dividiendo la cadena **“formatDate”** en partes utilizando el método **split ("/")**. Esto se hace para separar el día, el mes y el año, que están separados por barras inclinadas "/" en el formato original.
* Después de dividir la cadena, se obtienen tres valores: **“day”**, que representa el día del mes, **“month”**, que representa el mes, y **“year”**, que representa el año.
* Luego, la función utiliza la sintaxis de plantillas de cadena (template literals) de JavaScript para formar una nueva cadena con el formato deseado. En este caso, se coloca ${**year**}-${**month**}-${**day**} dentro de las comillas invertidas **(` `**). Esto indica que se reemplazarán las variables ${**year**}, ${**month**} y ${**day**} con sus valores correspondientes.
* Finalmente, la función devuelve la nueva cadena resultante con el formato **"aaaa-mm-dd"**. Los valores de **year**, **month** y **day** se concatenarán en el orden correcto, utilizando guiones "-" como separadores.

6. La función **“validationDate”** recibe dos parámetros, en este caso el primero corresponde a la fecha inicial **“dateCurrent”**, y el segundo a la fecha final **“dateEndt”**, dentro de esta función se realizan las validaciones correspondientes y se calculan los días hábiles entre las fechas. Esta función será la encargada de llamar a la función de **“isDateFormat ()”**, la función de **“changeDateFormat”** y **“getWorkingDays”** definidas anteriormente.

Paso a paso de la funcionalidad:



* La función comienza verificando si el tipo de **“dateCurrentt”** es un **"object"** y el tipo de **“dateEndt”** es un **"string"**. Esto se hace con la condición **typeof dateCurrentt === "object" && typeof dateEndt === "string"**. Esto es importante porque la función espera que **“dateCurrentt”** sea una fecha válida **(objeto Date)** y **“dateEndt”** sea una cadena de texto que representa una fecha en formato **"dd/mm/aaaa"**.
* Si la condición anterior se cumple, significa que **“dateCurrentt”** es un objeto Date y **“dateEndt”** es una cadena de texto que representa una fecha.
* A continuación, se llama a la función **“isDateFormat(dateEndt)”** para verificar si **“dateEndt”** tiene un formato de fecha válido (en formato "dd/mm/aaaa"). La función **“isDateFormat”** fue explicada previamente y devuelve true si **“dateEndt”** tiene un formato de fecha válido y false si no lo tiene.
* Si “**isDateFormat(dateEndt)**” devuelve true, significa que **“dateEndt”** tiene un formato de fecha válido. Entonces, la función procede a realizar las siguientes operaciones:

**a**. Se llama a la función **“changeDateFormat(dateEndt)”** para cambiar el formato de **“dateEndt”** de "dd/mm/aaaa" a "aaaa-mm-dd". La función **changeDateFormat** también fue explicada previamente y devuelve la fecha en el nuevo formato.

**b**. Se crea un nuevo objeto Date **“dateEndtOk”** utilizando el valor devuelto por **“changeDateFormat(dateEndt)”**. Esto se hace para obtener un objeto Date válido con la fecha en el formato correcto.

**c**. Finalmente, se llama a la función **“getWorkingDays (dateCurrentt, dateEndtOk)”** para calcular los días hábiles entre las fechas **“dateCurrentt”** y **“dateEndtOk”**. La función **getWorkingDays** fue explicada previamente y devuelve el número de días hábiles entre las fechas.

* Si **“isDateFormat(dateEndt)”** devuelve false, significa que **“dateEndt”** no tiene un formato de fecha válido. En este caso, la función devuelve la cadena "No es un formato de fecha".
* Si la condición inicial **typeof dateCurrentt === "object" && typeof dateEndt === "string"** no se cumple, significa que al menos uno de los argumentos no es válido. En este caso, la función devuelve la cadena "Fecha actual no corresponde a un object date".