MANUAL DEL PROGRAMADOR PARA EL SISTEMA DE INFORMACIÓN WEB PARA LA GESTIÓN DOCUMENTAL EN LA EMPRESA PRONTOALUM HNS

OSCAR STIVEN OVALLE FAJARDO

JHON JAIRO BERNAL

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS

FACULTAD TECNOLÓGICA

SISTEMATIZACIÓN DE DATOS

BOGOTA D.C.

2018

Contenido

INTRODUCCIÓN 6
1. PROGRAMACIÓN 7
2. CONTRATOS-CONTROLLER.JS 8
2.1 Module, controller 8
2.2 ContratosControllerFunc 8
2.3 Inicializar combos: 9
2.4 Listar 9
2.5 Validar plazos 10
2.6 Acción 11
2.7 Filtrar: 12
2.8 Limpiar filtro de búsqueda: 12
2.9 Cambiar Pagina: 13
2.10 irConsultar: 13
2.11 Guardar 14
2.12 actualizar: 16
2.13 cambiarContructora 18
2.14 mostrarModalObra 18
2.15 validarCamposRequeridos 19
2.16 guardarObra 19
2.17 cambioFechaRadicacion. 20
2.18 enviarNotificacion 21
2.19 irOtrosi: 22
2.20 irPoliza 23
2.21 contar 23
2.22 enviarMail 24
2.23 actualizarNotificacionContrato 25
2.24 validarPlazo 26

2.25 listarObra 28
2.26 listarConstructora 29
2.27 guardarConstructora 30
2.28 guardarObra 31
2.29 Documentos Adjuntos 32
2.30 eliminarDocumento 33
2.31 dataURItoBlob 35
2.32 enviarDocumentos 35
2.33 verDocumento 37
3. Contratos- Services.js 39
3.1 Inicio Services comercial 39
3.2 contratosData 41
3.3 requestFailed 41
3.4 guardar 42
3.5 actualizar 43
3.6 listarObra 44
3.7 listarConstructora 46
3.7 guardarObra 47

TABLA DE ILUSTRACIONES

FIGURA 1 Paquetes 7
FIGURA 2 Module, controller 8
FIGURA 3 ContratosControllerFunc 9
FIGURA 4 Inicializar combos 9
FIGURA 5 Listar 10
FIGURA 6 Validar plazos 11
FIGURA 7 Acción 12
FIGURA 8 Filtrar: 12
FIGURA 9 Limpiar filtro 13
FIGURA 10 Cambiar Pagina 13
FIGURA 11 irConsultar: 14
FIGURA 12 Guardar 16
FIGURA 13 actualizar 18
FIGURA 14 cambiarContructora 18
FIGURA 15 mostrarModalObra 19
FIGURA 16 validarCamposRequeridos 19
FIGURA 17 guardarObra 20
FIGURA 18 cambioFechaRadicacion 20
FIGURA 19 enviarNotificacion 22
FIGURA 20 irOtrosi 22
FIGURA 21 irPoliza 23
FIGURA 22 contar 24
FIGURA 23 enviarMail 25
FIGURA 24 actualizarNotificacionContrato 26
FIGURA 25 validarPlazo 27
FIGURA 26 listarObra 28
FIGURA 27 listarConstructora 30
FIGURA 28 guardarConstructora 31
FIGURA 29 guardarObra 32
FIGURA 30 Documentos Adjunto 33
FIGURA 31 eliminarDocumento 34
FIGURA 32 dataURItoBlob 35
FIGURA 33 enviarDocumentos 36
FIGURA 34 verDocumento 39
FIGURA 35 Inicio Services comercial 40
FIGURA 36 contratosData 41
FIGURA 37 requestFailed 42
FIGURA 38 guardar 43

FIGURA 39 actualizar....44
FIGURA 40 listarObra....45
FIGURA 41 listarConstructora....47
FIGURA 42 guardarObra....48

INTRODUCCIÓN

El propósito de este manual del programador, es dar a conocer al lector los códigos fuentes del programa realizado. Para ello tratamos de la forma más concisa de explicar cada uno de los códigos, junto con la programación utilizada en el desarrollo del software, esto con el fin de que el usuario pueda modificar a gusto alguno de los valores y parámetros de las funciones que se encuentran expuestas en la programación de sistema

1. PROGRAMACIÓN

Este proyecto está dividido en Dos partes importantes para el funcionamiento del sistema de información como se ve en la figura 1, tiene Dos paquetes que funcionan de la siguiente manera:

- **Controller.js:** Los controladores en AngularJS son objetos que permiten desarrollar la lógica de la aplicación, enlaza el ámbito, con la vista y permite tener un control total de los datos.

- **Services.js:** son más que pequeñas fábricas de funciones y objetos para más tarde, como veremos, poder inyectar a nuestros controladores, muy útil y necesario. Aquí, como hemos dicho, podremos guardar funciones, arrays, objetos y todo lo que deseemos tener disponible para inyectar donde nos haga falta, en nuestro caso crearemos un array para la factoría y otro para el servicio, de esta forma veremos cómo poder utilizarlos después en una vista.

Estos paquetes tienen consigo una serie de funciones que le dan el funcionamiento al sistema se explicara lo más detallado posible las funciones más importantes.

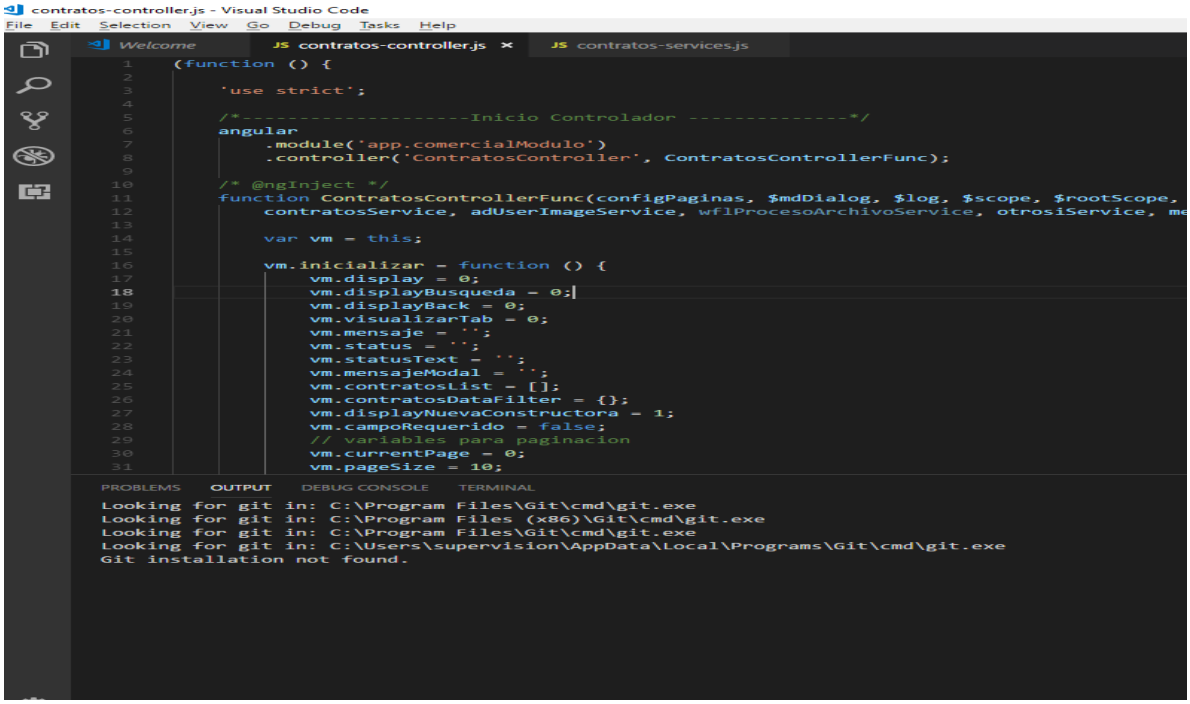


FIGURA 1 Paquetes

2. CONTRATOS-CONTROLLER.JS

Este paquete contiene las vistas o interfaces del sistema de información, este paquete maneja funciones .js que son:

2.1 Module, controller

En esta función se inicia el modulo comercial y el controlador como se ve en la figura 2

```
angular
        .module('app.comercialModulo')
        .controller('ContratosController', ContratosControllerFunc);
```

FIGURA 2 Module, controller

2.2 ContratosControllerFunc

Inyecta todas nuestras variable globales y llama a nuestro objeto inicializar como se ve en la figura 3.

```
function ContratosControllerFunc(configPaginas, $mdDialog, $log, $scope,
$rootScope, $sce, $window, $location, $controller, Loadfuncion,
alertfuncion, mostrarAlertasFormulario,
        contratosService, adUserImageService, wflProcesoArchivoService,
otrosiService, mensajesMailsService) {

        var vm = this;

        vm.inicializar = function () {
            vm.display = 0;
            vm.displayBusqueda = 0;
            vm.displayBack = 0;
            vm.visualizarTab = 0;
            vm.mensaje = '';
            vm.status = '';
```

```
            vm.statusText = '';
            vm.mensajeModal = '';
            vm.contratosList = [];
            vm.contratosDataFilter = {};
            vm.displayNuevaConstructora = 1;
            vm.campoRequerido = false;
            // variables para paginacion
            vm.currentPage = 0;
            vm.pageSize = 10;
            vm.pages = [];
            vm.skip = 0;
            // variables para Tabs
            vm.selectedTab = 0;
        };
vm.inicializar();
```

FIGURA 3 ContratosControllerFunc

2.3 Inicializar combos:

En esta función está inicializando las listas de las obras y constructoras para poder filtrarlas como se ve en la figura 4.

```
vm.inicializarCombos = function () {
            if (!vm.obraList) { listarObra(); }
            if (!vm.contructoraList) { listarConstructora(); }
            if (!vm.estadoList) { listarEstado(); }
        };
        vm.inicializarCombos();
```

FIGURA 4 *Inicializar combos*

2.4 Listar

En esta función tabula y especifica que contratos están vencidos y los llama para mostrarlo en la página deseada como se ve en la figura 5.

```
vm.listar = function () {
        vm.contratosList = [];
        vm.contratosDataFilter.clientId = $rootScope.objetoUser.clientId;
```

```
        vm.contratosDataFilter.orgId = $rootScope.objetoUser.orgId;
        vm.contratosDataFilter.skip = vm.skip;
        vm.contratosDataFilter.limit = vm.pageSize;
        if (!vm.contratosDataFilter.estado) {
            vm.contratosDataFilter.estado = 'A';
        }

        Loadfuncion.mostrar();


contratosService.listar(vm.contratosDataFilter).then(requestCompleted);
```

FIGURA 5 Listar

2.5 Validar plazos

Esta es la funcion donde se valida los plazos y se listan en cada buscador como se ve en la figura 6.

```
contratosService.listar(vm.contratosDataFilter).then(requestCompleted);

            function requestCompleted(dataResponse) {

                if (dataResponse && dataResponse.resultado &&
dataResponse.resultado === 'OK') {

                    $log.debug('contratos-controller listar
requestCompleted: dataResponse{OK} ', dataResponse);
                    vm.contratosList =
dataResponse.resultadosList.resultado;
                    vm.mensajeLista = '';
                    Promise.all(vm.contratosList.map(function (element) {
                        if (element.fechaVencimiento) {
                            element.fechaVencimiento = new
Date(element.fechaVencimiento);

element.fechaVencimiento.setDate(element.fechaVencimiento.getDate() + 1);
                            element.fechaRadicacion = new
Date(element.fechaRadicacion);
```

```
element.fechaRadicacion.setDate(element.fechaRadicacion.getDate() + 1);
                            element.fechaVencimientoFormat =
moment(element.fechaVencimiento).format('YYYY-MM-DD')
                            return element
                        }
                    }))
                        .then(function (resp) {
                            validarPlazo();
                        })


                } else if (dataResponse) {

                    $log.debug('contratos-controller listar
requestCompleted: dataResponse{ERROR} ', dataResponse);
                    if (dataResponse.status) { vm.status =
dataResponse.status; }
                    if (dataResponse.statusText) { vm.statusText =
dataResponse.statusText; }
                    if (dataResponse.data &&
Object.prototype.toString.call(dataResponse.data) === '[object Array]') {
vm.validacionesList = dataResponse.data; vm.displayBack = 1; }
                    else if (dataResponse.mensajeError && typeof
dataResponse.mensajeError === 'string') { }
                    vm.mensajeLista = dataResponse.mensajeError;
                }
                contar();
                Loadfuncion.desbanecer();
            }
        };
        vm.listar();
```

FIGURA 6 Validar plazos

2.6 Acción

Son las acciones de listar, adicionar, consultar editar como se ve en la figura 7.

```
vm.getDescAccion = function () {
        if (vm.display === 0) { return 'Lista de Contratos'; }
        if (vm.display === 1) { return 'Adicionar nuevo Contrato'; }
        if (vm.display === 2) { return 'Consultar Contratos'; }
        if (vm.display === 3) { return 'Editar Contratos'; }
        else if (vm.display === -1) { return 'Mensajes '; }
    };
```

FIGURA 7 ***Acción***

2.7 Filtrar:

Filtra y elimina todo lo de los textos como se ve en la figura 8.

```
vm.filtrar = function () {
        if (vm.contratosDataFilter.descripcion == "") {
            console.log("entor a borrar descripcion")
            delete vm.contratosDataFilter.descripcion;
        }
        vm.skip = 0;
        vm.currentPage = 0;
        vm.listar();
    };
```

FIGURA 8 Filtrar:

2.8 Limpiar filtro de búsqueda:

Borra lo que ya se busco en el texto de consultar como se ve en la figura 9.

```
vm.limpiarFiltroBusqueda = function () {
        vm.skip = 0;
        vm.currentPage = 0;
        vm.contratosDataFilter = {};
        vm.listar();
    };
```

FIGURA 9 Limpiar filtro

2.9 **Cambiar Pagina**:

En esta funcion esta cambia de modulo al inicio (index) como se ve en la figura 10.

```
vm.cambiarPagina = function (index) {
            vm.currentPage = index;
            vm.skip = (index) * vm.pageSize
            vm.listar();
        };
```

FIGURA 10 Cambiar Pagina

2.10 irConsultar:

En esta funcion colsutaremos dependiendo el parámetro ingresado en las cajas de texto como se ve en la figura 11

```
vm.irConsultar = function (item) {
            vm.inicializarCombos();
            vm.guardarNuevoDocumento = false;
            vm.contratosData = {};
            mostrarAlertasFormulario.inicializarFormulario($scope.form2);
            mostrarAlertasFormulario.inicializarFormulario($scope.form3);

            if (item.fechaRadicacion) {
                item.fechaRadicacionFormat = new
Date(item.fechaRadicacion)
                vm.fechaMinimaVencimiento = item.fechaRadicacionFormat;
            }
            if (item.fechaVencimiento) {
                item.fechaVencimientoFormat = new
Date(item.fechaVencimiento)
```

```
            }
            vm.contratosData = item;
            if (vm.contratosData.anexo) {
                vm.contratosData.adjuntado = true;
                vm.contratosData.verAdjuntados = true;
            }
        };
```

FIGURA 11 irConsultar:

2.11 Guardar

En esta funcion se manejara los registros guardados de cualquier procedencia como se ve en la figura 12.

```
vm.guardar = function () {
            let flag = true;
            // Validar Documentos requeridos
            if (vm.contratosData.adjuntado != true) {
                flag = false;
            }

            if (flag) {
                if (vm.contratosData.fechaRadicacionFormat) {
                    vm.contratosData.fechaRadicacion =
moment(vm.contratosData.fechaRadicacionFormat).format('YYYY-MM-DD');
                }
                if (vm.contratosData.fechaVencimientoFormat) {
                    vm.contratosData.fechaVencimiento =
moment(vm.contratosData.fechaVencimientoFormat).format('YYYY-MM-DD');
                }

                vm.contratosData.clientId =
$rootScope.objetoUser.clientId;
                vm.contratosData.orgId = $rootScope.objetoUser.orgId;
                vm.contratosData.estado = 'A';
                vm.contratosData.notificacion = '0';
```

```
                $log.debug('contratos-controller guardar DATA{} ',
vm.contratosData);

                contratosService.guardar(vm.contratosData,
$rootScope.objetoUser).then(requestCompleted);

                function requestCompleted(dataResponse) {

                    if (dataResponse && dataResponse.resultado &&
dataResponse.resultado === 'OK') {

                        $log.debug('contratos-controller guardar
requestCompleted: dataResponse{OK} ', dataResponse);


vm.enviarDocumentos(dataResponse.objetoResponse.contratoId)
                        vm.display = 0;
                        mostrarDialog(dataResponse.mensaje)

                    } else if (dataResponse) {

                        $log.debug('contratos-controller guardar
requestCompleted: dataResponse{ERROR} ', dataResponse);
                        if (dataResponse.status) { vm.status =
dataResponse.status; }
                        if (dataResponse.statusText) { vm.statusText =
dataResponse.statusText; }
                        if (dataResponse.data &&
Object.prototype.toString.call(dataResponse.data) === '[object Array]') {
vm.validacionesList = dataResponse.data; vm.displayBack = 1; }
                        else if (dataResponse.data && typeof
dataResponse.data === 'string') { vm.mensaje = dataResponse.data; }
                        vm.display = -1;
                        vm.mensaje = dataResponse.mensajeError;
                        vm.displayBack = 0;
                    }
                }
            }
            else {
                $mdDialog.show(
                    $mdDialog.alert()
```

```
.parent(angular.element(document.querySelector('#popupContainer')))
                        .clickOutsideToClose(true)
                        .title('Error')
                        .textContent('Algunos documentos adjuntos son
requeridos')
                        .ariaLabel('Alert Dialog')
                        .ok('ACEPTAR')
                        .targetEvent()
                );
            }
        };
```

FIGURA 12 Guardar

2.12 **actualizar**:

En esta funcion se manejara los registros modificación de cualquier procedencia como se ve en la figura 13.

```
    vm.actualizar = function () {

            if (vm.contratosData.fechaRadicacionFormat) {
                vm.contratosData.fechaRadicacion =
moment(vm.contratosData.fechaRadicacionFormat).format('YYYY-MM-DD');
            }
            if (vm.contratosData.fechaVencimientoFormat) {
                vm.contratosData.fechaVencimiento =
moment(vm.contratosData.fechaVencimientoFormat).format('YYYY-MM-DD');
            }

            vm.contratosData.clientId = $rootScope.objetoUser.clientId;
            vm.contratosData.orgId = $rootScope.objetoUser.orgId;
```

```
            $log.debug('contratos-controller actualizar DATA{} ',
vm.contratosData);

            contratosService.actualizar(vm.contratosData,
$rootScope.objetoUser).then(requestCompleted);

            function requestCompleted(dataResponse) {

                if (dataResponse && dataResponse.resultado &&
dataResponse.resultado === 'OK') {

                    $log.debug('contratos-controller actualizar
requestCompleted: dataResponse{OK} ', dataResponse);

                    vm.display = 0;
                    mostrarDialog(dataResponse.mensaje)
                    if (vm.guardarNuevoDocumento == true) {
                        vm.enviarDocumentos(vm.contratosData.contratoId)

                    }
                    else {
                        vm.limpiarFiltroBusqueda()
                    }

                } else if (dataResponse) {

                    $log.debug('contratos-controller actualizar
requestCompleted: dataResponse{ERROR} ', dataResponse);
                    if (dataResponse.status) { vm.status =
dataResponse.status; }
                    if (dataResponse.statusText) { vm.statusText =
dataResponse.statusText; }
                    if (dataResponse.data &&
Object.prototype.toString.call(dataResponse.data) === '[object Array]') {
vm.validacionesList = dataResponse.data; vm.displayBack = 1; }
                    else if (dataResponse.data && typeof dataResponse.data
=== 'string') { vm.mensaje = dataResponse.data; }
                    vm.display = -1;
                    vm.mensaje = dataResponse.mensajeError;
                    vm.displayBack = 0;
                }
```

```
            }
        };
```

FIGURA 13 actualizar

2.13 cambiarContructora

En esta funcion cambiara la constructora cuando se filtre en el campo como se ve en la figura 14.

```
vm.cambiarContructora = function (obraId) {

            let ind = vm.obraList.findIndex(function (elemt) { return
elemt.obraId == obraId })
            if (ind != -1) {
                vm.contratosData.constructoraId =
vm.obraList[ind].constructoraId;
            }

        };
```

FIGURA 14 cambiarContructora

2.14 mostrarModalObra

En esta funcion mostraremos el método mostrar obra al ser filtrada la constructora como se ve en la figura 15.

```
    vm.mostrarModalObra = function () {
            vm.contratosDataObra = {};
            vm.contratosDataConstructora = {};
            vm.displayNuevaConstructora = 1;
            vm.campoRequerido = false;
            $('#modalObra').modal('show');
        };
```

FIGURA 15 mostrarModalObra

2.15 validarCamposRequeridos

En esta funcion se manejan las alertas de espacios no completos en los formularios como se ve en la figura 16.

```
vm.validarCamposRequeridos = function () {
            vm.contratosDataConstructora = {};
            if (vm.contratosDataObra.constructoraId) { delete
vm.contratosDataObra.constructoraId; }

            if (vm.displayNuevaConstructora == 1) {
                vm.campoRequerido = false;
            }
            else {
                vm.campoRequerido = true;
            }
        };
```

FIGURA 16 validarCamposRequeridos

2.16 guardarObra

En esta funcion se manejara los registros de guardar la información de la obra como se ve en la figura 17.

```
vm.guardarObra = function () {

            Loadfuncion.mostrar();

            if (vm.displayNuevaConstructora == 0) {
                guardarConstructora()
            }
            else {
```

```
                guardarObra()
            }
        };
```

FIGURA 17 guardarObra

2.17 **cambioFechaRadicacion.**

Se modifica las fechas cuando se crean nuevos cambios en los contratos, cursos, exámenes como se ve en la figura 18.

```
vm.cambioFechaRadicacion = function () {
            Loadfuncion.mostrar();
            setTimeout(function () {
                vm.fechaMinimaVencimiento = new
Date(vm.contratosData.fechaRadicacionFormat)
                if (vm.contratosData.fechaVencimientoFormat &&
vm.contratosData.fechaVencimientoFormat) {

                    let fechaRad = new
Date(vm.contratosData.fechaRadicacionFormat);
                    let fechaVen = new
Date(vm.contratosData.fechaVencimientoFormat);
                    if (fechaRad > fechaVen) {
                        delete vm.contratosData.fechaVencimientoFormat;
                        let mensajeFecha = 'La fecha de vencimiento no
puede ser anterior a la fecha de radicacion'
                        mostrarDialog(mensajeFecha)
                    }

                }
                Loadfuncion.desbanecer();
            }, 100)
        };
```

FIGURA 18 cambioFechaRadicacion

2.18 enviarNotificacion

En esta funcion se manejara las notificaciones de envio por correo electrónico como se ve en la figura 19.

```
  vm.enviarNotificacion = function (data) {
            let notificacionData = {};
            notificacionData.clientId = $rootScope.objetoUser.clientId;
            notificacionData.orgId = $rootScope.objetoUser.orgId;
            notificacionData.nombre = data.descripcion;
            notificacionData.fechaVencimiento =
data.fechaVencimientoFormat;
            notificacionData.estado = 'A';
            if (data.not == 2) {
                notificacionData.descripcion = "El contrato " +
data.descripcion + " esta proximo a vencerce el dia: " +
data.fechaVencimientoFormat;
            }
            if (data.not == 3) {
                notificacionData.descripcion = "El contrato " +
data.descripcion + " entro en vigencia el dia: " +
data.fechaVencimientoFormat;
            }

            $log.debug('contratos-controller guardarNotificacion DATA{} ',
notificacionData);

            contratosService.guardarNotificacion(notificacionData,
$rootScope.objetoUser).then(requestCompleted);

            function requestCompleted(dataResponse) {
```

```
                if (dataResponse && dataResponse.resultado &&
dataResponse.resultado === 'OK') {

                    $log.debug('contratos-controller guardarNotificacion
requestCompleted: dataResponse{OK} ', dataResponse);
                    enviarMail(notificacionData)
                    actualizarNotificacionContrato(data)
                    mostrarDialog("Notificacion ha sido creada
exitosamente");

                } else if (dataResponse) {

                    $log.debug('contratos-controller guardarNotificacion
requestCompleted: dataResponse{ERROR} ', dataResponse);

                }
            }


        };
```

FIGURA 19 enviarNotificacion

2.19 irOtrosi

En esta funcion se manejara los registros de los otrosi de cada contrato como se ve en la figura 20.

```
vm.irOtrosi = function () {
            $log.debug('contratos-controller irOtrosi  contratoId Data{} -
> ', vm.contratosData.contratoId);
            $location.path('/otrosi').search({ contratoIdDefault:
vm.contratosData.contratoId });
            Loadfuncion.mostrar();
        }
```

FIGURA 20 irOtrosi

2.20 irPoliza

En esta funcion mostraremos la clasificación y categorías específicas que se manejaran en el sistema hay que tener en cuenta que hay más métodos pero en esta ocasión describiremos los más importantes de este como se ve en la figura 21.

```
      vm.irPoliza = function () {
           $log.debug('contratos-controller irPoliza  contratoId Data{} ->
', vm.contratosData.contratoId);
           $location.path('/poliza').search({ contratoIdDefault:
vm.contratosData.contratoId });
           Loadfuncion.mostrar();
       }
```

FIGURA 21 irPoliza

2.21 **contar**

En esta funcion mostraremos el método contador de todas las páginas como se ve en la figura 22.

```
   function contar() {
           $log.debug("contratos-controller contar DATA{}",
vm.contratosDataFilter)

contratosService.contar(vm.contratosDataFilter).then(requestCompleted);

           function requestCompleted(dataResponse) {
               if (dataResponse != null) {
                   $log.debug('contratos-controller contar
requestCompleted: dataResponse{OK} ', dataResponse);
                   let cant = dataResponse.objetoResponse;
                   let respuesta = configPaginas.paginacion(cant,
vm.currentPage, vm.pageSize);
                   vm.pages = respuesta.pages;
                   let pagina = angular.copy(vm.pages)
                   vm.finalPage = pagina.pop();

               } else if (dataResponse) {
```

```
                    $log.debug('contratos-controller contar
requestCompleted: dataResponse{ERROR} ', dataResponse);
                }

            }
        };
```

FIGURA 22 contar

2.22 **enviarMail**

En esta funcion se manejara los registros envearan los correos como se ve en la figura 23.

```
    function enviarMail(data) {
            let correos = [];
            let email = "osovallef@gmail.com";
            correos.push(email);
            let enviar = {};
            enviar.email = correos;
            enviar.mensaje = data.descripcion;
            enviar.asunto = "Alerta sistema de gesion documetal";
            enviar.adClientId = $rootScope.objetoUser.clientId;
            enviar.adOrgId = $rootScope.objetoUser.orgId;
            enviar.userId = "10002";
            enviar.msgEnvioMasivoId = "133";

            $log.debug('contratos-controller envioMails enviar{} ',
enviar);
            mensajesMailsService.envioMails(enviar,
enviar).then(requestCompleted);
            function requestCompleted(dataResponse) {
                if (dataResponse && dataResponse.resultado &&
dataResponse.resultado === 'OK') {
                    $log.debug('contratos-controller envioMails
requestCompleted: dataResponse{OK} ', dataResponse);

                } else if (dataResponse) {
                    $log.debug('contratos-controller envioMails
requestCompleted: dataResponse{ERROR} ', dataResponse);
                }
```

FIGURA 23 enviarMail

2.23 actualizarNotificacionContrato

En esta funcion se manejara actualizaciones del semáforo como se ve en la figura 24.

```
        function actualizarNotificacionContrato (data) {
            let contratoData = {};
            contratoData.clientId = $rootScope.objetoUser.clientId;
            contratoData.orgId = $rootScope.objetoUser.orgId;
            contratoData.contratoId = data.contratoId;

            if (data.not == 2) {
                contratoData.notificacion = 1;
            }
            if (data.not == 3) {
                contratoData.notificacion = 2;
            }

            $log.debug('contratos-controller actualizar DATA{} ',
contratoData);

            contratosService.actualizar(contratoData,
$rootScope.objetoUser).then(requestCompleted);

            function requestCompleted(dataResponse) {

                if (dataResponse && dataResponse.resultado &&
dataResponse.resultado === 'OK') {

                    $log.debug('contratos-controller actualizar
requestCompleted: dataResponse{OK} ', dataResponse);

                    vm.limpiarFiltroBusqueda()

                } else if (dataResponse) {

                    $log.debug('contratos-controller actualizar
requestCompleted: dataResponse{ERROR} ', dataResponse);
```

```
                    if (dataResponse.status) { vm.status =
dataResponse.status; }
                    if (dataResponse.statusText) { vm.statusText =
dataResponse.statusText; }
                    if (dataResponse.data &&
Object.prototype.toString.call(dataResponse.data) === '[object Array]') {
vm.validacionesList = dataResponse.data; vm.displayBack = 1; }
                    else if (dataResponse.data && typeof dataResponse.data
=== 'string') { vm.mensaje = dataResponse.data; }
                    vm.display = -1;
                    vm.mensaje = dataResponse.mensajeError;
                    vm.displayBack = 0;
                }
            }

        };
```

FIGURA 24 actualizarNotificacionContrato

2.24 validarPlazo

Valida que la fecha de vencimiento no sea mayor que la de la fecha inicial de cualquier parámetro como se ve en la figura 25.

```
function validarPlazo() {

            vm.contratosList.map(function (element) {
                // Traer fechas de la lista y cambiarles el formato
                var fechaVencimiento =
moment(element.fechaVencimiento).format('YYYY-MM-DD');
                var fechaActual = moment().format('YYYY-MM-DD');
                // ejecutar la funcion moment en cada fecha
                var fVen = moment(fechaVencimiento);
                var fAct = moment(fechaActual);
                // Calcular la diferencia de dias
                var diasDiferenciaFechaFin = fVen.diff(fAct, 'days')
                console.log("diasDiferenciaFechaFin-> ",
diasDiferenciaFechaFin)
```

```
                element.semaforo = '';
                if (element.estado == 'A') {
                    // Valida si ya se vencio el contrato
"rojo"
                    if (diasDiferenciaFechaFin < 0) {
                        element.semaforo = 'Caduco fecha de contrato'
                        element.rojo = true;
                        element.amarillo = true;
                        element.verde = true;

                        if (element.notificacion == "0" ||
element.notificacion == "1") {
                            element.not = 3;
                        }
                    }
                    // Valida si tiene uno o mas de 20 dias por vencerse
"amarillo"
                    else if (diasDiferenciaFechaFin < 20 &&
diasDiferenciaFechaFin >= 0) {
                        element.semaforo = "Contrato proximo a vencerse"
                        element.rojo = false;
                        element.amarillo = true;
                        element.verde = true;
                        if (element.notificacion == "0") {
                            element.not = 2;
                        }
                    }
                    // Valida si tiene tiene menos de 20 dias por vencerse
"verde"
                    else if (diasDiferenciaFechaFin >= 20) {
                        element.semaforo = "Contrato vigente"
                        element.rojo = false;
                        element.amarillo = false;
                        element.verde = true;
                        element.not = 1;
                    }
                }
            });
        }
```

FIGURA 25 validarPlazo

2.25 listarObra

Se listan en la tabla todas las obras existentes como se ve en la figura 26.

```
function listarObra() {
            return new Promise(function (resolve, reject) {
                let objetoData = {};
                objetoData.clientId = $rootScope.objetoUser.clientId;
                objetoData.orgId = $rootScope.objetoUser.orgId;
                objetoData.estado = 'A';

                $log.debug('contratos-controller listarObra  Data{} ',
objetoData);

                contratosService.listarObra(objetoData,
$rootScope.objetoUser).then(requestCompleted);

                function requestCompleted(dataResponse) {

                    if (dataResponse && dataResponse.resultado &&
dataResponse.resultado === 'OK') {

                        $log.debug('contratos-controller listarObra
requestCompleted: dataResponse{OK} ', dataResponse);
                        vm.obraList =
dataResponse.resultadosList.resultado;

                    } else if (dataResponse) {

                        $log.debug('contratos-controller listarObra
requestCompleted: dataResponse{ERROR} ', dataResponse);

                    }
                    resolve(true)
                }
            })
        };
```

FIGURA 26 listarObra

2.26 listarConstructora

En esta funcion lista todas las contructoras existentes en las tablas como se ve en la figura 27.

```
function listarConstructora() {
            return new Promise(function (resolve, reject) {
                let objetoData = {};
                objetoData.clientId = $rootScope.objetoUser.clientId;
                objetoData.orgId = $rootScope.objetoUser.orgId;
                objetoData.estado = 'A';

                $log.debug('contratos-controller listarConstructora
Data{} ', objetoData);

                contratosService.listarConstructora(objetoData,
$rootScope.objetoUser).then(requestCompleted);

                function requestCompleted(dataResponse) {

                    if (dataResponse && dataResponse.resultado &&
dataResponse.resultado === 'OK') {

                        $log.debug('contratos-controller
listarConstructora requestCompleted: dataResponse{OK} ', dataResponse);
                        vm.contructoraList =
dataResponse.resultadosList.resultado;

                    } else if (dataResponse) {

                        $log.debug('contratos-controller
listarConstructora requestCompleted: dataResponse{ERROR} ', dataResponse);

                    }
                    resolve(true)
                }
            })
        };
```

FIGURA 27 listarConstructora

2.27 guardarConstructora

Esta funcion guarda la información de la de la constructora como se ve en la figura 28.

```
function guardarConstructora() {
            return new Promise(function (resolve, reject) {
                vm.contratosDataConstructora.clientId =
$rootScope.objetoUser.clientId;
                vm.contratosDataConstructora.orgId =
$rootScope.objetoUser.orgId;
                vm.contratosDataConstructora.estado = 'A';

                $log.debug('contratos-controller guardarConstructora
DATA{} ', vm.contratosDataConstructora);


contratosService.guardarConstructora(vm.contratosDataConstructora,
$rootScope.objetoUser).then(requestCompleted);

                function requestCompleted(dataResponse) {

                    if (dataResponse && dataResponse.resultado &&
dataResponse.resultado === 'OK') {

                        $log.debug('contratos-controller
guardarConstructora requestCompleted: dataResponse{OK} ', dataResponse);

guardarObra(dataResponse.objetoResponse.constructoraId)

resolve(dataResponse.objetoResponse.constructoraId)

                    } else if (dataResponse) {

                        $log.debug('contratos-controller
guardarConstructora requestCompleted: dataResponse{ERROR} ',
dataResponse);
                        resolve()
                    }
```



FIGURA 28 guardarConstructora

2.28 guardarObra

Esta funcion guarda la información de la de la obra como se ve en la figura 29.

```
function guardarObra(constructoraId) {
            return new Promise(function (resolve, reject) {
                vm.contratosDataObra.clientId =
$rootScope.objetoUser.clientId;
                vm.contratosDataObra.orgId = $rootScope.objetoUser.orgId;
                vm.contratosDataObra.estado = 'A';

                if (constructoraId && constructoraId != null) {
                    vm.contratosDataObra.constructoraId = constructoraId;
                }

                $log.debug('contratos-controller guardarObra DATA{} ',
vm.contratosDataObra);

                contratosService.guardarObra(vm.contratosDataObra,
$rootScope.objetoUser).then(requestCompleted);

                function requestCompleted(dataResponse) {

                    if (dataResponse && dataResponse.resultado &&
dataResponse.resultado === 'OK') {

                        $log.debug('contratos-controller guardarObra
requestCompleted: dataResponse{OK} ', dataResponse);
                        vm.obraResponse = dataResponse.objetoResponse;

                        listarObra()
                            .then(function (resp) {
                                vm.contratosData.obraId =
vm.obraResponse.obraId
                                listarConstructora()
```

```
                                    .then(function (resp) {

vm.cambiarContructora(vm.obraResponse.obraId)
                                        Loadfuncion.desbanecer();
                                        $('#modalObra').modal('hide');

mostrarDialog(dataResponse.mensaje);
                                    })
                            })
                        resolve(dataResponse.mensaje)

                    } else if (dataResponse) {

                        $log.debug('contratos-controller guardarObra
requestCompleted: dataResponse{ERROR} ', dataResponse);
                        resolve(dataResponse.mensajeError)
                    }
                }
            })
        };
```

FIGURA 29 guardarObra

2.29 **Documentos Adjuntos**

Esta funcion deja subir los archivos adjunntos de cualquier formulario como se ve en la figura 30.

```
// <-- Documentos Adjuntos -->
        $scope.$on("fileSelected", function (event, args) {
            $scope.$apply(function () {
                vm.guardarNuevoDocumento = true;
                console.log("args->", args)
                let objeto = args.file
                vm.contratosData.document = objeto;
                vm.contratosData.adjuntado = true;
                vm.contratosData.vistaPrevia = true;
```

```
            });
        });
```

FIGURA 30 Documentos Adjunto

2.30 eliminarDocumento

Por ultimo esta es una de las funciones para eliminar los documentos adjuntos en los formularios si son subidos mal como se ve en la figura 31.

```
vm.eliminarDocumento = function () {
            if (vm.contratosData.document != undefined) {
                delete vm.contratosData.document
            }
            vm.contratosData.adjuntado = false;
            vm.contratosData.vistaPrevia = false;
            vm.contratosData.verAdjuntados = false;
        };
vm.showImage = function () {

            return new Promise(function (resolve, reject) {

                var reader = new FileReader();

                reader.onload = function (e) {
                    $scope.imagem = e.target.result;
                    resolve($scope.imagem);
                };

                vm.nombreArchivo = vm.contratosData.document.name
                reader.readAsDataURL(vm.contratosData.document);

            })
                .then(function (res) {

                    vm.extension = '';
                    var imagem = res;
```

```
                    var a = document.createElement("a");

                    if (imagem.toLowerCase().indexOf('image/jpeg') > 0) {
                        vm.type = 'image/jpeg';
                        vm.extension = '.jpg';
                    } else
                        if (imagem.toLowerCase().indexOf('image/png') > 0)
{
                            vm.type = 'image/png';
                            vm.extension = '.png';
                        } else
                            if
(imagem.toLowerCase().indexOf('application/pdf') > 0) {
                                vm.type = 'application/pdf';
                                vm.extension = '.pdf';
                            } else
                                if
(imagem.toLowerCase().indexOf('application/msword') > 0) {
                                    vm.type = 'application/msword';
                                    vm.extension = '.doc';
                                } else {
                                    vm.type = 'application/octet-stream';
                                    vm.extension = '.docx';
                                }

                    var decodedImage = dataURItoBlob(imagem);
                    var blob = new Blob([decodedImage], { type: vm.type
});
                    var fileURL = URL.createObjectURL(blob);
                    $log.debug('showImage fileURL{} ', fileURL);
                    a.href = fileURL;
                    a.download = vm.nombreArchivo;
                    a.click();
                    window.URL.revokeObjectURL(fileURL);
                })
        }
```

FIGURA 31 eliminarDocumento

2.31 **dataURItoBlob**

Por ultimo esta es una de las funcions más importantes que nos ayuda a hacer la clasificación de combates a través de árboles con este método se puede hacer una pirámide que nos permita saber cómo son las formaciones para los combates como se ve en la figura 32.

```
function dataURItoBlob(dataURI) {

            // convert base64/URLEncoded data component to raw binary data
held in a string
            var byteString;
            if (dataURI.split(',')[0].indexOf('base64') >= 0)
                byteString = atob(dataURI.split(',')[1]);
            else
                byteString = unescape(dataURI.split(',')[1]);

            // separate out the mime component
            var mimeString =
dataURI.split(',')[0].split(':')[1].split(';')[0];

            // write the bytes of the string to a typed array
            var ia = new Uint8Array(byteString.length);
            for (var i = 0; i < byteString.length; i++) {
                ia[i] = byteString.charCodeAt(i);
            }

            return new Blob([ia], { type: mimeString });
        }
```

FIGURA 32 dataURItoBlob

2.32 enviarDocumentos

Esta funcion envía el pdf por via correo electrónico como se ve en la figura 33.

```
vm.enviarDocumentos = function (id) {
```

```
            let file = vm.contratosData.document;
            let formData = {};
            formData.adClientId = $rootScope.objetoUser.clientId;
            formData.adOrgId = $rootScope.objetoUser.orgId;
            formData.id = id;


            $log.debug('Contrato-controller enviarDocumentos formData{} ',
formData);

            adUserImageService.cargarDocumento(file, formData,
$rootScope.objetoUser).then(requestCompleted);

            function requestCompleted(dataResponse) {

                if (dataResponse && dataResponse.resultado &&
dataResponse.resultado === 'OK') {

                    $log.debug('Contrato-controller enviarDocumentos
requestCompleted: dataResponse{OK} ', dataResponse);
                    if (dataResponse.objetoResponse != null) {

                        vm.guardarDocumentos(dataResponse.objetoResponse);
                    }

                } else if (dataResponse) {

                    $log.debug('Contrato-controller enviarDocumentos
requestCompleted: dataResponse{ERROR} ', dataResponse);

                }
            }

        };
```

FIGURA 33 enviarDocumentos

2.33 verDocumento

Esta funcion deja ver los documentos pdf en el descargar como se ve en la figura 34.

```
vm.verDocumento = function () {

            Loadfuncion.mostrar();
            let wflProcesoArchivoData = {};
            wflProcesoArchivoData.adClientId =
$rootScope.objetoUser.clientId;
            wflProcesoArchivoData.adOrgId = $rootScope.objetoUser.orgId;

            wflProcesoArchivoData.nombreArchivo = vm.contratosData.nombre;

            var extensionDoc =
wflProcesoArchivoData.nombreArchivo.slice((wflProcesoArchivoData.nombreArchi
vo.lastIndexOf(".") - 1 >>> 0) + 2);

            $log.debug('dpePersona-controller verDocumento
wflProcesoArchivoData{} ', wflProcesoArchivoData, ' - extensionDoc ',
extensionDoc);

            otrosiService.verDocumento(wflProcesoArchivoData,
$rootScope.objetoUser).then(requestCompleted);


            function requestCompleted(dataResponse) {

                $log.debug('dpePersona-controller verDocumento
requestCompleted: dataResponse{} ', dataResponse);
                if (dataResponse) {
                    var a = document.createElement("a");

                    if (extensionDoc == 'pdf') {
                        var file = new Blob([dataResponse], { type:
'application/pdf' });
                    }
                    else if (extensionDoc == 'xlsx' || extensionDoc ==
'xls') {
                        var file = new Blob([dataResponse], { type:
'application/vnd.ms-excel' });
```

```
                    }
                    else if (extensionDoc == 'jpg' || extensionDoc ==
'jpge') {
                        var file = new Blob([dataResponse], { type:
'image/jpeg' });
                    }
                    else if (extensionDoc == 'docx' || extensionDoc ==
'doc') {
                        var file = new Blob([dataResponse], { type:
'application/msword' });
                    }
                    else if (extensionDoc == 'txt') {
                        var file = new Blob([dataResponse], { type:
'text/plain' });
                    }
                    else if (extensionDoc == 'ppt' || extensionDoc ==
'pptx') {
                        var file = new Blob([dataResponse], { type:
'application/vnd.ms-powerpoint' });
                    }
                    else {
                        var file = new Blob([dataResponse], { type:
'application/octet-stream' });
                    }

                    $log.debug('dpePersona-controller verDocumento file{} ',
file);
                    var fileURL = URL.createObjectURL(file);
                    $log.debug('dpePersona-controller verDocumento fileURL{}
', fileURL);
                    a.href = fileURL;
                    a.download = wflProcesoArchivoData.nombreArchivo;
                    a.click();
                    window.URL.revokeObjectURL(fileURL);

                } else if (dataResponse && dataResponse.mensajeError) {

                    alertfuncion.getError();
                    $log.debug('dpePersona-controller verDocumento
requestCompleted dataResponse.mensajeError{} ', dataResponse.mensajeError);
```

```
                }
            }
            Loadfuncion.desbanecer();
        };
    };
})();
```

FIGURA 34 verDocumento

3. CONTRATOS- SERVICES.JS

3.1 Inicio Services comercial

Inicializa sesión en servicios comercial dependiendo el tipo de rol como se ve en la figura 35.

```
(function () {

    'use strict';

    /*--------------------Inicio Services comercial --------------*/
    angular
        .module('app.comercialModulo')
        .service('contratosService', contratosServiceFunc);

    /* @ngInject */
    function contratosServiceFunc($log, $rootScope, $location, $http,
urlService) {

        // llamado para obtener la url Loopback
        var urlLB2 = urlService.urlLB2();
```

```
        return {

            listar: function (contratosData) {
                var url = urlLB2 + 'ContratosVs/listarContratosV';

                $log.debug('Contratos-services listar DATA_ENVIADA: url{}
', url, ' - contratosData{Object} ', contratosData);
                return $http.post(url,
contratosData).then(requestCompleted, requestFailed);

                function requestCompleted(response) {

                    $log.debug('Contratos-services listar then:
response{} ', response);
                    urlService.updateTokenAuthorization(response);
                    return response.data;
                }

                function requestFailed(error) {

                    $log.error('Contratos-services listar requestFailed:
error{} ', error);
                    var newMessage = '';
                    if (error.data) { newMessage = newMessage +
error.statusText + ' - ' + error.data; }
                    if (error.status === 401) { $rootScope.msgError =
newMessage; $location.path('/login'); }
                    else if (error.status === 400 || error.status === 404)
{ /* $rootScope.msgError = error.data; $location.path('/400');*/ }
                    else if (error.status === 500) { $rootScope.msgError =
newMessage; $location.path('/500'); }
                    else { $rootScope.sttError = error.status;
$rootScope.msgError = newMessage; $location.path('/infoPage'); }
                    urlService.updateTokenAuthorization(error);
                    return error;

                }
            },
```

FIGURA 35 Inicio Services comercial

3.2 contratosData

Inicializa todas las funciones de los contratos como se ve en la figura 36.

```
  contar: function (contratosData) {
                var url = urlLB2 + 'ContratosVs/contarContratosV';

                $log.debug('Contratos-services contar DATA_ENVIADA: url{}
', url, ' - contratosData{Object} ', contratosData);
                return $http.post(url,
contratosData).then(requestCompleted, requestFailed);

                function requestCompleted(response) {

                    $log.debug('Contratos-services contar then:
response{} ', response);
                    urlService.updateTokenAuthorization(response);
                    return response.data;
                }
```

FIGURA 36 contratosData

3.3 requestFailed

Muestra los errores existentes en el programa como se ve en la figura 37.

```
function requestFailed(error) {

                    $log.error('Contratos-services contar requestFailed:
error{} ', error);
                    var newMessage = '';
```

```
                    if (error.data) { newMessage = newMessage +
error.statusText + ' - ' + error.data; }
                    if (error.status === 401) { $rootScope.msgError =
newMessage; $location.path('/login'); }
                    else if (error.status === 400 || error.status === 404)
{ /* $rootScope.msgError = error.data; $location.path('/400');*/ }
                    else if (error.status === 500) { $rootScope.msgError =
newMessage; $location.path('/500'); }
                    else { $rootScope.sttError = error.status;
$rootScope.msgError = newMessage; $location.path('/infoPage'); }
                    urlService.updateTokenAuthorization(error);
                    return error;

                }
            },
```

FIGURA 37 requestFailed

3.4 guardar

Se llama este servicio para guardar lo que se necesite como se ve en la figura 38.

```
guardar: function (contratosData) {
                var url = urlLB2 + 'Contratos/guardarContrato';

                $log.debug('Contratos-services guardar DATA_ENVIADA: url{}
', url, ' - contratosData{Object} ', contratosData);
                return $http.post(url,
contratosData).then(requestCompleted, requestFailed);

                function requestCompleted(response) {

                    $log.debug('Contratos-services guardar then:
response{} ', response);
                    urlService.updateTokenAuthorization(response);
                    return response.data;
                }
```

```
                function requestFailed(error) {

                    $log.error('Contratos-services guardar requestFailed:
error{} ', error);
                    var newMessage = '';
                    if (error.data) { newMessage = newMessage +
error.statusText + ' - ' + error.data; }
                    if (error.status === 401) { $rootScope.msgError =
newMessage; $location.path('/login'); }
                    else if (error.status === 400 || error.status === 404)
{ /* $rootScope.msgError = error.data; $location.path('/400');*/ }
                    else if (error.status === 500) { $rootScope.msgError =
newMessage; $location.path('/500'); }
                    else { $rootScope.sttError = error.status;
$rootScope.msgError = newMessage; $location.path('/infoPage'); }
                    urlService.updateTokenAuthorization(error);
                    return error;

                }
            },
```

FIGURA 38 guardar

3.5 actualizar

Se llama este servicio para actualizar cualquier formulario como se ve en la figura 39.

```
actualizar: function (contratosData) {
                var url = urlLB2 + 'Contratos/actualizarContrato';

                $log.debug('Contratos-services actualizar DATA_ENVIADA:
url{} ', url, ' - contratosData{Object} ', contratosData);
                return $http.post(url,
contratosData).then(requestCompleted, requestFailed);

                function requestCompleted(response) {
```

```
                    $log.debug('Contratos-services actualizar then:
response{} ', response);
                    urlService.updateTokenAuthorization(response);
                    return response.data;
                }

                function requestFailed(error) {

                    $log.error('Contratos-services actualizar
requestFailed: error{} ', error);
                    var newMessage = '';
                    if (error.data) { newMessage = newMessage +
error.statusText + ' - ' + error.data; }
                    if (error.status === 401) { $rootScope.msgError =
newMessage; $location.path('/login'); }
                    else if (error.status === 400 || error.status === 404)
{ /* $rootScope.msgError = error.data; $location.path('/400');*/ }
                    else if (error.status === 500) { $rootScope.msgError =
newMessage; $location.path('/500'); }
                    else { $rootScope.sttError = error.status;
$rootScope.msgError = newMessage; $location.path('/infoPage'); }
                    urlService.updateTokenAuthorization(error);
                    return error;

                }
            },
```

FIGURA 39 actualizar

3.6 listarObra

Se llama la funcion listar obra para las consultas como se ve en la figura 40.

```
            listarObra: function (contratosData) {
                var url = urlLB2 + 'Obras/listarObra';
```

```
                $log.debug('Contratos-services listarObra DATA_ENVIADA:
url{} ', url, ' - contratosData{Object} ', contratosData);
                return $http.post(url,
contratosData).then(requestCompleted, requestFailed);

                function requestCompleted(response) {

                    $log.debug('Contratos-services listarObra then:
response{} ', response);
                    urlService.updateTokenAuthorization(response);
                    return response.data;
                }

                function requestFailed(error) {

                    $log.error('Contratos-services listarObra
requestFailed: error{} ', error);
                    var newMessage = '';
                    if (error.data) { newMessage = newMessage +
error.statusText + ' - ' + error.data; }
                    if (error.status === 401) { $rootScope.msgError =
newMessage; $location.path('/login'); }
                    else if (error.status === 400 || error.status === 404)
{ /* $rootScope.msgError = error.data; $location.path('/400');*/ }
                    else if (error.status === 500) { $rootScope.msgError =
newMessage; $location.path('/500'); }
                    else { $rootScope.sttError = error.status;
$rootScope.msgError = newMessage; $location.path('/infoPage'); }
                    urlService.updateTokenAuthorization(error);
                    return error;

                }
            },
```

FIGURA 40 listarObra

3.7 listarConstructora

Se llama el servicio listar constructora para consultar toda la informacion como se ve en la figura 41.

```
listarConstructora: function (contratosData) {
                var url = urlLB2 + 'Constructoras/listarConstructora';

                $log.debug('Contratos-services listarConstructora
DATA_ENVIADA: url{} ', url, ' - contratosData{Object} ', contratosData);
                return $http.post(url,
contratosData).then(requestCompleted, requestFailed);

                function requestCompleted(response) {

                    $log.debug('Contratos-services listarConstructora
then:  response{} ', response);
                    urlService.updateTokenAuthorization(response);
                    return response.data;
                }

                function requestFailed(error) {

                    $log.error('Contratos-services listarConstructora
requestFailed: error{} ', error);
                    var newMessage = '';
                    if (error.data) { newMessage = newMessage +
error.statusText + ' - ' + error.data; }
                    if (error.status === 401) { $rootScope.msgError =
newMessage; $location.path('/login'); }
                    else if (error.status === 400 || error.status === 404)
{ /* $rootScope.msgError = error.data; $location.path('/400');*/ }
                    else if (error.status === 500) { $rootScope.msgError =
newMessage; $location.path('/500'); }
                    else { $rootScope.sttError = error.status;
$rootScope.msgError = newMessage; $location.path('/infoPage'); }
                    urlService.updateTokenAuthorization(error);
                    return error;

                }
```

```
            },
```

FIGURA 41 listarConstructora

3.7 guardarObra

Se llama el servicio guarda obra para guardar toda la informcaion como se ve en la figura 42.

```
guardarObra: function (contratosData) {
                var url = urlLB2 + 'Obras/guardarObra';

                $log.debug('Contratos-services guardarObra DATA_ENVIADA:
url{} ', url, ' - contratosData{Object} ', contratosData);
                return $http.post(url,
contratosData).then(requestCompleted, requestFailed);

                function requestCompleted(response) {

                    $log.debug('Contratos-services guardarObra then:
response{} ', response);
                    urlService.updateTokenAuthorization(response);
                    return response.data;
                }

                function requestFailed(error) {

                    $log.error('Contratos-services guardarObra
requestFailed: error{} ', error);
                    var newMessage = '';
                    if (error.data) { newMessage = newMessage +
error.statusText + ' - ' + error.data; }
```

```
                    if (error.status === 401) { $rootScope.msgError =
newMessage; $location.path('/login'); }
                    else if (error.status === 400 || error.status === 404)
{ /* $rootScope.msgError = error.data; $location.path('/400');*/ }
                    else if (error.status === 500) { $rootScope.msgError =
newMessage; $location.path('/500'); }
                    else { $rootScope.sttError = error.status;
$rootScope.msgError = newMessage; $location.path('/infoPage'); }
                    urlService.updateTokenAuthorization(error);
                    return error;

                }
            },
```

FIGURA 42 guardarObra