ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Grado en Ingeniería del Software

“Sistema de Apoyo a la decisión para salud estructural”

“Decision Support System for structural health”

Realizado por
Juan Ramírez Fernández
Tutorizado por
Luis Manuel Llopis Torres
Departamento
Lenguajes y ciencias de la programación.

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE DE 2019

Fecha defensa:
El Secretario del Tribunal

Resumen: En este trabajo de Fin de Grado se ha realizado el desarrollo de una aplicación web de monitorización de infraestructura, basado en el uso del framework de JavaScript Angular JS. Con el objetivo de implementar un “Sistema de apoyo a la decisión”.

Hoy en día en las empresas es habitual la monitorización de las infraestructuras, para analizar su estado a tiempo real, con el fin de emitir diagnósticos y prever comportamientos indeseados.
Estos análisis se suelen realizar con algoritmos inteligentes o comparándolos con un histórico. En nuestro caso la técnica utilizada ha sido la “comparación de patrones” aplicada a datos de vibraciones en pavimentos.

Se plantea una alternativa a las convencionales aplicaciones de escritorio, desarrollando una web en la que el usuario puede comprobar y analizar la salud estructural de un equipamiento desde cualquier lugar y dispositivo.

El usuario definirá los patrones de análisis deseados a partir de los cuales podrá generar un informe con los datos recogidos.

Abstract: In this Final Degree Project an infraestructure monitorization web based on JavaScript Angular JS has been developed. The objetive of this Project is the generation of a “Decision Support System”.

Nowadays is usual among companies all over the world real time monitorization of their infraestructures, in order to anticipate unwanted behaviours. This kind of analysis are typically generated with Smart-algorithms or by historic comparison. In this particular case “pattern’s correlation” has been the selected technique (aplied to pavement vibrations).

The web represents an alternative to conventional desktop applications: the user can control and analyse the structural health of an equipment from any place and device.

A report can be generated according to the user’s requirements.

TABLA DE CONTENIDOS

PÁG

TABLA DE ILUSTRACIONES

1. Introducción y objetivos

Existe una gran cantidad de sistemas que monitorizan infraestructuras para obtener información que analice la salud estructural de éstas. Este análisis de los datos puede hacerse con algoritmos inteligentes o comparándolos con un histórico.

El objetivo de este análisis puede ser no sólo decidir sobre el estado de la infraestructura sino también poder emitir un diagnóstico de su comportamiento futuro. Los sistemas que integran todas las fases de recogida de datos, modelados, análisis y predicción son los denominados “Sistemas de apoyo a la decisión”.

Estas aplicaciones tienen una estructura definida respetando las etapas que cubren estos sistemas, recogida de datos, modelado, análisis, diagnóstico y predicción, todo ello integrado en una base de datos. Dado que la gran mayoría de estos sistemas son aplicaciones de escritorio el objetivo es desarrollar un sistema de apoyo a la decisión web que facilite el acceso desde cualquier lugar y desde cualquier dispositivo.

Puesto que los SADs son dependientes de la tecnología de monitorización en esta propuesta lo aplicaremos a datos de vibraciones aplicados a pavimentos y analizaremos, diagnosticaremos y predeciremos el estado de dicha infraestructura.

Aunque se estudiaran diferentes alternativas para el análisis y predicción, la técnica que se utilizará será la comparación de patrones. La aplicación será parametrizable para que los usuarios puedan incluir las características especiales de cada infraestructura.

Para este proyecto fin de grado, tenemos como objetivos:

- Realizar el análisis, diseño y desarrollo para la creación de una web en Angular JS.
- Describir a grandes rasgos que es Angular JS y como se ha utilizado.
- La adquisición de conocimientos en otras tecnologías como HTML, PHP y MySQL.
- Plantear una solución para que los clientes de dicha aplicación puedan realizar el análisis de los datos en cualquier lugar y con la mayor exactitud posible, pudiendo modificar los parámetros de análisis cuando se quiera.

2. Tecnologías

En cuanto al desarrollo de la aplicación, se ha utilizado Angular JS, junto a otras tecnologías. Angular JS es un popular framework de JavaScript para el desarrollo en el lado del cliente.

Hace uso del patrón Modelo-Vista-Controlador, que permite separar la capa de presentación, la capa lógica y los componentes de la aplicación.

Su principal uso es la creación de aplicaciones web de una sola página, dotándola de gran fluidez y rapidez.

Junto a Angular JS, para poder realizar la aplicación web, ha sido necesario el uso de las siguientes tecnologías:

- Html: Sus siglas significan *HyperText Markup Language* (lenguaje de marcas de hipertexto), y hace referencia al lenguaje de marcado para la elaboración de páginas web. El HTML se escribe en forma de "etiquetas", rodeada por corchetes angulares (<,>, /). Los elementos son la estructura básica de HTML, y tienen dos propiedades básicas: atributos y contenido. Cada atributo y contenido tienen ciertas restricciones para que se considere válido al documento HTML.
  Se ha utilizado html5 para el desarrollo de las vistas de la aplicación.

- CSS: Son las siglas de *Cascading Style Sheets* (Hoja de estilo en cascada), es un leguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML o XML. El World Wide Web Consortium (W3C) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores. La idea detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

  La información de estilo puede ser definida en un documento separado o en el mismo documento HTML. En nuestro caso hemos utilizado un documento separado.

- PHP: Es un lenguaje de programación de uso general de código del lado del servidor, originalmente diseñado para el desarrollo web de contenido dinámico[1]. Fue uno de los primeros lenguajes de programación del lado del servidor, que se podían incorporar directamente en el documento HTML, en lugar de llamar a un archivo externo para que procesara los datos.

El código es interpretado por un servidor web con un módulo de procesador de PHP que genera la página web resultante. PHP ha evolucionado por lo que ahora se incluye también una interfaz por línea de comando que puede ser usada en aplicaciones graficas independientes. Puede ser usado en la mayoría de los servidores web al igual que en casi todos los sistemas operativos y plataformas sin ningún coste.
Son los archivos con extensión .php del código.

Ilustración 1 Logo PHP

- MySQL: Es un sistema de gestión de bases de datos relacional, desarrollado bajo licencia dual GPL/Licencia comercial por Oracle Corporation, y está considerada como la base de datos open source más popular del mundo, y una de las más populares en general junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo web. MySQL fue inicialmente desarrollado por MYSQL AB. Esta fue adquirida por Sun Microsystems en 2008, y ésta a su vez fue comprada por Oracle Corporation en 2010, la cual ya era dueña desde 2005 de Innobase Oy, empresa finlandesa desarrolladora del motor InnoDB para MySQL.

Ilustración 2 Logo MySQL

Dentro de los archivos php, se han realizado en MYSQL una parte de la queries necesaria para obtener información de la base de datos[2].

- Entorno de desarrollo: PHPStorm es un IDE comercial multiplataforma que proporciona un editor para PHP, HTML y JavaScript[3], está construido sobre la plataforma IntelliJ IDEA de JetBrain. Se puede descargar y evaluar de forma gratuita durante 30 días, pasado ese tiempo hay que adquirir una licencia. Sin embargo, se puede obtener una licencia individual de estudiante con duración de 1 año.

Ilustración 3 Logo PhpStorm

2.1 ¿Qué es Angular JS?

Angular JS, o simplemente Angular, es un framework de JavaScript de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de modelo-vista-controlador (MVC), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles.

Ilustración 4 Logo

Se centra en el desarrollo Front-End, es decir, en la interfaz de la aplicación web. Este framework adapta y amplía el HTML tradicional, para servir mejor contenido dinámico a través de un data binding bidireccional, que permite la sincronización automática de modelos y vistas [4].

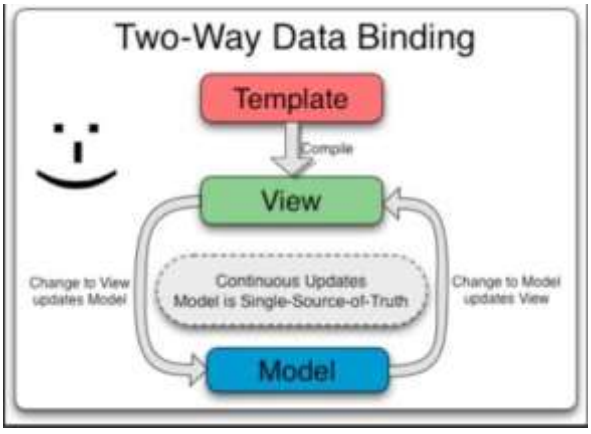

Ilustración 5 Sincronización modelo-vistas

La biblioteca lee el HTML que contiene atributos de las etiquetas personalizadas adicionales, entonces obedece a las directivas de los atributos personalizados, y une las piezas de entrada o salida de la página a un modelo representado por las variables estándar de JavaScript. Los valores de las variables de JavaScript se pueden configurar manualmente, o recuperados de los recursos JSON estáticos o dinámicos.

Angular JS se puede combinar con el entorno en tiempo de ejecución Node.js, el framework para servidor Express.js y la base de datos MongoDB para formar el conjunto MEAN.

Angular JS está construido en torno a la creencia, de que la programación declarativa es la que debe utilizarse para generar interfaces de usuario y enlazar componentes de software, mientras que la programación imperativa es excelente para expresar la lógica de negocio.

Angular JS pone menos énfasis en la manipulación del DOM y mejora la testeabilidad y el rendimiento.

Los objetivos de diseño:

- Separar la manipulación del DOM de la lógica de la aplicación. Esto mejora la capacidad de prueba del código.
- Considerar a las pruebas de la aplicación como iguales en importancia a la escritura de la aplicación. La dificultad de las pruebas se ve reducida drásticamente por la forma en que el código está estructurado.
- Disociar el lado del cliente de una aplicación del lado del servidor. Esto permite que el trabajo de desarrollo avance en paralelo, y permite la reutilización de ambos lados.
- Guiar a los desarrolladores a través de todo el proceso de desarrollo de una aplicación: desde el diseño de la interfaz de usuario, a través de la escritura de la lógica del negocio, hasta las pruebas.

Con el uso de la inyección de dependencias, Angular lleva servicios tradicionales del lado del servidor, tales como controladores dependientes de la vista, a las aplicaciones web del lado del cliente. En consecuencia, gran parte de la carga en el backend se reduce, lo que conlleva a aplicaciones web mucho más ligeras.

2.2 MVC

El modelo–vista–controlador (MVC) es un patrón de arquitectura de software, que separa la parte visual, de la funcionalidad y las estructuras de datos. Para ello el MVC propone la construcción de tres componentes distintos: el **modelo**,

la **vista** y el **controlador**, es decir, por un lado, define componentes para la representación de la información, y por otro lado para la interacción del usuario.

Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

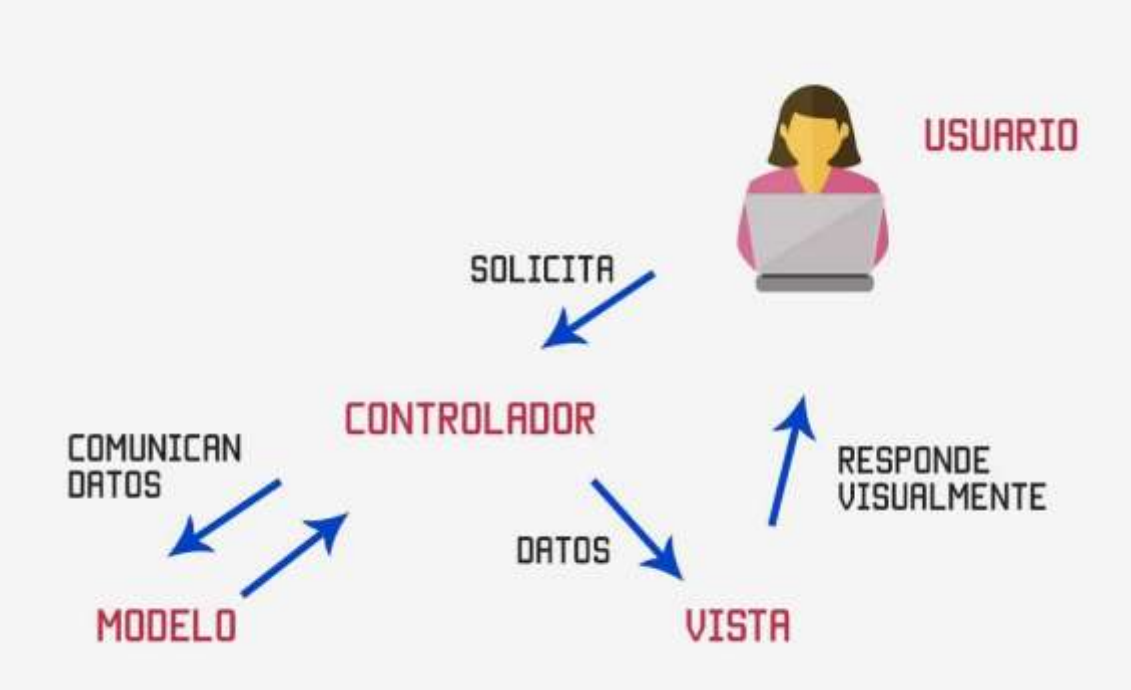

Ilustración 6 Modelo-vista-controlador

Los componentes del MVC se pueden definir como:

- **Modelo:** Es la representación de la información con la cual el sistema opera, por lo tanto, gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la vista aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al modelo a través del controlador.

- **Vista:** Presenta el modelo (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario), por tanto, requiere de dicho modelo la información que debe representar como salida.

- **Controlador:** Responde a eventos (usualmente acciones del usuario), e invoca peticiones al modelo cuando se hace alguna solicitud sobre la información (por ejemplo, editar un registro en una base de datos). También puede enviar comandos a su vista asociada si se solicita un cambio en la forma en que se presenta el modelo, por tanto, se podría decir que el controlador hace de intermediario entre la vista y el modelo.

2.3 Ámbito

La traducción de scope es ámbito, sin embargo, el término es tan habitual que se suele usar directamente la palabra en inglés.

En una aplicación en Angular JS, no existe un único ámbito, hay diferentes scopes en diferentes partes del código, anidados o en paralelo. Los scope se crean automáticamente para cada uno de los controladores de una aplicación, y además existe un scope raíz para toda la aplicación ($rootScope)[5].

2.3.1 Scope hace de enlace entre las vistas y los controladores:

El scope es la magia que permite que los datos que se manejan en los controladores pasen a las vistas, y viceversa. Es el enlace que traslada esos datos de un lugar a otro, sin que el programador tenga que hacer nada.

En los controladores se inyecta el scope mediante el parámetro $scope. Posteriormente se podrán añadir datos como propiedades a ese objeto inyectado. Ejemplo:

```
.controller('otroCtrl', function($scope){
  $scope.algo = "probando scope...";
  $scope.miDato = "otra cosa..."
});
```

Ilustración 7 Ejemplo controlador

Lo anterior generará un dato en el scope llamado "algo" y otro llamado "miDato". Desde las vistas se puede acceder a datos del scope usando la sintaxis de dobles llaves:

```
{{ algo }}
```

Ilustración 8 Acceso a datos

Por lo tanto, "algo" es un dato que está almacenado en el scope desde el controlador. Con esta expresión se traslada a la página. También se puede acceder al scope al usar la directiva ng-model y definiendo un dato en el modelo.

```
<input type="text" ng-model="miDato">
```

Ilustración 9 Alternativa de acceso al scope

El scope sirve para trasladar datos entre el controlador y la vista. Para poder acceder a "algo" o a "miDato" no es necesario enviarlo desde el controlador a la vista ni de la vista al controlador, sólo hay que crear esa propiedad en el $scope. Una de las ventajas de Angular JS, es el Binding, cuando el dato se modifica, ya sea porque se le asigna un nuevo valor desde el controlador o porque se cambia lo que hay escrito en el INPUT de la vista, la otra parte es consciente de ese cambio sin tener que estar suscritos a eventos.

2.3.2 Ámbitos del scope dentro de la vista:

Cada scope tiene su ámbito restringido a una parte del código. Si revisamos el siguiente HTML:

```
<div ng-controller="miAppController">
    <p>Contenido de la vista acotado por un controlador</p>
    <p>{{ cualquierCosa }}</p>
</div>
<section>
    Desde fuera de esa división no hay acceso al scope del controlador
</section>
```

Ilustración 10 Ejemplo de restricción del scope

En este ejemplo, hay un scope creado por el controlador "miAppController", y tiene validez dentro de ese código HTML en el lugar donde se ha definido la directiva ngController. Es decir, los elementos que se asignen a $scope dentro de este controlador se podrán ver desde la etiqueta DIV y todas sus hijas, pero no desde etiquetas que se encuentren fuera, como es el caso del SECTION que hay después.

2.3.3 Alias del scope:

Existe otra modalidad de enviar datos al scope y es por medio de un alias. Para conseguir esto, al declarar los controladores en el HTML (directiva ng-controller) hay que utilizar la sintaxis "controller..as".

```
<div ng-controller="pruebaAppCtrl as vm">
```

Ilustración 11 Ejemplo de alias

En este caso se podrán generar datos en el scope de dos maneras, o bien a través del mismo objeto $scope que se puede inyectar, como está explicado en el apartado anterior, o bien a través de la variable this dentro de la función.

Ilustración 12 Ejemplo de controlador

```
.controller('pruebaAppCtrl', function($rootScope){
    var modeloDeLaVista = this;
    modeloDeLaVista.otroDato = "Esto está ahora en el scope, para acceder a través de un alias";
});
```

Gracias al alias en la vista se puede acceder a ese dato con una expresión como ésta:

```
{{ vm.otroDato }}
```

Ilustración 13 Ejemplo de acceso al

Este alias del scope facilita el trabajo cuando se está trabajando con varios ámbitos.

2.3.4 Ámbito raíz con $rootScope:

En Angular JS existe un ámbito o scope raíz, sobre el que se pueden insertar datos. Ese ámbito es accesible desde cualquier lugar de la aplicación, y es útil para definir valores a los que se puede acceder desde cualquier punto del HTML, independientemente del controlador de la aplicación en el que se esté trabajando. Se puede acceder al scope raíz de Angular desde el controlador, por medio de

$rootScope, que necesita ser inyectado en la función del controlador si se desea usar:

Ilustración 14 Ejemplo de $rootScope

```
.controller('pruebaAppCtrl', function($scope, $rootScope){
    $scope.scopeNormal = "Esto lo coloco en el scope normal de este controlador...";
    $rootScope.scopeRaiz = "Esto está en el scope raíz";
  });
```

Al guardar algo en el ámbito raíz, se puede acceder a ese valor, desde la vista, a través del nombre de la propiedad que se ha creado en $rootScope:

```
<div ng-controller="pruebaAppCtrl">
  {{ scopeNormal }}
  <br />
  {{ scopeRaiz }}
</div>
```

Ilustración 15 Ejemplo de acceso a rootScope

En este ejemplo se accede de la misma manera a un dato que está en el scope de este controlador y a un dato que está en el scope raíz. Esto sucede porque en Angular JS, si una variable del modelo no se encuentra en el scope actual, el propio sistema busca el dato en el scope padre. Si no, en el padre y así sucesivamente.

2.3.5 Conociendo $parent:

Un problema aparece cuando se tiene un dato con el mismo nombre en dos scopes distintos.

```
.controller('pruebaAppCtrl', function($scope, $rootScope){
    //sobreescribo una variable del scope
    //en realidad son dos datos distintos con dos valores distintos
    //uno está en el scope del controlador y el otro en el scope raíz "root"
    $scope.repetido = "Algo en el scope normal";
    $rootScope.repetido = "Algo en el scope raíz";
});
```

Ilustración 16 Ejemplo de ambigüedad Scope

En este caso, si en la vista se escribe {{ repetido }} existirá una ambigüedad, pues ese valor puede significar dos cosas, dependiendo de si se revisa un scope u otro. Angular resuelve esa situación devolviendo el dato del scope más

específico. De esta manera, retorna el valor que se encuentra en $scope.repetido y no se puede acceder a $rootScope.repetido, salvo que se use $parent.

2.3.6 Ejemplo con ámbitos corrientes y ámbito root:

Lo visto en este apartado, se puede resumir con el siguiente ejemplo:

```
div ng-app="pruebaApp" ng-controller="pruebaAppCtrl">
  {{ scopeNormal }}
  <br />
  {{ scopeRaiz }}
  <br />
  {{ algo }} --- {{ $parent.algo }}
</div>
```

Ilustración 17 Ejemplo HTML

```
.controller('pruebaAppCtrl', function($scope, $rootScope){
    $scope.scopeNormal = "Esto se coloca en el scope normal de este controlador...";
    $rootScope.scopeRaiz = "Esto está en el scope raíz";

    //si se sobreescribe una variable del scope

    $scope.algo = "Algo en el scope normal";
    $rootScope.algo = "Algo en el scope raíz";
  });
```

Ilustración 18 Ejemplo Javascript

2.4 Directivas

Las directivas son marcas en los elementos del árbol DOM, en los nodos del HTML, que indican al compilador de Angular que debe asignar cierto comportamiento a dichos elementos o transformarlos según corresponda.

Podríamos decir que las directivas nos permiten añadir comportamiento dinámico al árbol DOM haciendo uso de las nativas del propio Angular JS o extender la funcionalidad hasta donde necesitemos creando las nuestras propias. Se recomienda que sea en las directivas en el único sitio donde se manipule el árbol DOM, para que entre dentro del ciclo de vida de compilación, binding y renderización del HTML.

Las directivas son la técnica que nos va a permitir crear nuestros propios componentes visuales, encapsulando las posibles complejidades en la implementación, normalizando y parametrizándolos según nuestras necesidades.

2.4.1 Tipos de Directivas:

Hay varios tipos de directivas, las directivas nativas del propio Angular, y las creadas por nosotros mismos.

Las directivas nativas de Angular comienzan por el prefijo “ng-“. Ejemplos:

1. **ng-app**: Inicializa una aplicación de Angular JS.
2. **ng-controller**: Sirve para enlazar el controlador con la vista.
3. **ng-include**: Se usa para cargar trozos de HTML en la página.
4. **ng-view**: Se utiliza para indicar en qué parte del HTML vamos a inyectar las vistas.
5. **ng-model**: Enlaza datos de la aplicación con el HTML.
6. **ng-submit**: Especifica la función que se debe ejecutar cuando el formulario es enviado.
7. **ng-repeat**: Se encarga de repetir un elemento HTML.
8. **ng-click**: Se utiliza para especificar un evento click, es decir, el código que se debe ejecutar tras hacer click sobre un elemento del HTML en el que se ha incluido esta directiva.
9. **ng-show**: Permite que un elemento de la página esté visible (si está a true) o invisible (si está a false) en función de un valor de nuestro modelo.
10. **ng-cloak**: Esta directiva pausa el navegador, para que Angular realice las acciones necesarias, y a continuación se muestran los resultados esperados.
11. **ng-class**: Nos permite definir clases que posteriormente se aplican a los elementos.

Las **directivas propias** de Angular JS se crean de la forma siguiente:

```
.directive('nombreDirectiva', function(){

})
```

Ilustración 19 Ejemplo de directiva propia

Se establece una propiedad llamada Restrict, que define cómo será utilizada la directiva. Puede tomar los siguientes valores:

A: Restringe a la directiva ser adjunta utilizando un atributo, por ejemplo,

```
<div nombreDirectiva></div>
```

Ilustración 20 Propiedad A

E: Permite a la directiva ser utilizada cómo un elemento personalizado,

```
<nombreDirectiva></nombreDirectiva>
```

Ilustración 21 Propiedad E

C: Permite a la directiva ser usada añadiéndole una clase al elemento,

```
<div class="nombreDirectiva"></div>
```

Ilustración 22 Propiedad C

M: Permite a la directiva ser ejecutada desde un comentario HTML,

Por defecto Angular JS establece la directiva cómo atributo solamente, pero se pueden usar combinaciones de valores. Por ejemplo:

```
.directive('nombreDirectiva', function(){
    return {
        restrict: 'AE'
    }
})
```

Ilustración 23 Ejemplo de combinaciones de propiedades

2.5 Inyección de Dependencias

La Inyección de Dependencias (DI - Dependency Injection) es un pilar fundamental de Angular JS y es que la inyección de dependencias se relaciona con la forma en la que se hacen referencias desde el código.

2.5.1 ¿Qué es la inyección de dependencias?

Es un patrón de diseño orientado a objetos. Este patrón nos dice que los objetos necesarios en una clase serán suministrados y que por tanto no necesitamos que la propia clase cree estos objetos.

El framework de Angular JS gestiona la inyección de dependencias, por lo tanto, las dependencias, como por ejemplo de servicios, serán suministradas por Angular JS. Por ello, al crear un componente de Angular JS se deben especificar las dependencias que se esperan y será el propio framework el que proporcione los objetos que se solicitan. Por ejemplo, si se necesita utilizar un servicio en un controlador, al crear el controlador se debe especificar la dependencia del servicio y no intentar crear un objeto del servicio.

2.6 Servicios

2.6.1 ¿Qué son los servicios?

Los servicios son objetos singleton, inyectables por Dependency Injection (inyección de dependencias), donde se define la lógica de negocio de la aplicación, con el objetivo de que sea reutilizable e independiente de las vistas.

Los módulos de Angular exponen 5 métodos para definir servicios:

- constant
- value
- service
- factory
- provider

Constant es un servicio al que se le pasa directamente el valor de dicho servicio. Su principal característica es que se puede inyectar en cualquier sitio. Se define llamando al método constant de un módulo. A dicho método se le pasa el nombre de la constante y su valor.

Value es un objeto JavaScript que se le pasa al controlador y se instancia en la fase de configuración.

Por otro lado, están, provider, factory y service, que permiten crear objetos mucho más complejos que dependen de otros objetos. Cada uno es un caso más concreto del anterior, y como gran elemento diferencial, provider permite generar una API para configurar el servicio resultante.

2.6.2 Ciclo de vida de Angular JS:

Para entender las diferencias entre cada uno de estos elementos, es importante explicar el ciclo de vida Angular JS. El ciclo de vida de una aplicación en Angular JS se divide en dos fases: la de configuración y la de ejecución.

Fase de configuración:

La fase de configuración se ejecuta en primer lugar. Durante esta fase los servicios aún no pueden instanciarse (no pueden pasarse servicios por inyección de dependencias). El objetivo de esta fase es definir cuál va a ser la configuración de la aplicación a la hora de ejecutarse. Hay que tener precaución si se quiere configurar un servicio en la resolución de un método asíncrono en esta fase, porque el ciclo de configuración podría haber finalizado antes. En esta fase, solo se pueden inyectar constants y providers.

Fase de ejecución:

La fase de ejecución es donde se ejecuta toda la lógica de la aplicación y empieza una vez ha concluido la fase de configuración. La interacción entre las vistas, controladores y servicios de la aplicación sucede en esta fase. En esta fase se pueden inyectar constants, values, services y factories.

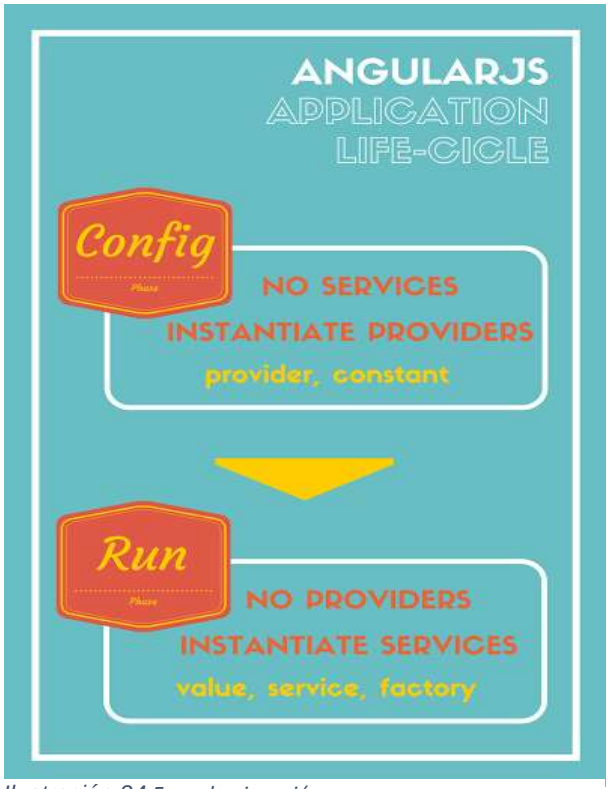

Ilustración 24 Fase de ejecución

2.6.3 Tipos de Servicios y Ejemplos:

- Constant:

Sirve para almacenar valores simples de cualquier tipo que no deben cambiar, no se pueden inyectar dependencias en su definición, y tampoco es configurable, pero si se puede inyectar en funciones de configuración.

```
myApp.constant('SERVERS',{  DEVELOPMENT: "http://localhost:8080/app", PRODUCTION:"http://myDomain.com/app"});
```

Ilustración 25 Ejemplo de definición de constante

- Value:

Permite definir objetos simples y primitivas que se pueden inyectar únicamente durante la fase de ejecución. No se pueden inyectar dependencias en su definición ni es configurable.

```
myApp.value('randomize',function(){
    return Math.floor(Math.random()*10000);
})
myApp.value('token','a1234567890');
myApp.value('User',{'id': 'someId'})
```

Ilustración 26 Ejemplo de definición de value

- Service:

Un servicio es una función constructor que define el servicio. Este servicio se puede inyectar únicamente durante la fase de ejecución. No obstante, si se pueden inyectar dependencias en su definición, aunque no es configurable.

Internamente, Angular utiliza el método new sobre este constructor a la hora de instanciar el servicio, por lo que se le pueden añadir propiedades con this. Ese objeto this es exactamente lo que devuelve el servicio.

A continuación, se puede ver un ejemplo de definición de servicio, donde se inyecta una dependencia (el value token del punto anterior):

```
myApp.service('AuthBearer', ['token', function(token) {
    this.authValue = "bearer " + token;
}]);
```

Ilustración 27 Ejemplo de definición de servicio

- Factory:

Una factoría es un caso más genérico de service, más enfocado a la inicialización del servicio, dado que no devuelve el constructor sino el objeto en sí mismo. Como en el servicio, se puede inyectar únicamente durante la fase de ejecución, y si se pueden inyectar dependencias en su definición, aunque no es configurable.

```
myApp.factory('apiToken', ['$window', 'clientId', function apiTokenFactory($window, clientId) {
  var encrypt = function(data1, data2) {
    // NSA-proof encryption algorithm:
    return (data1 + ':' + data2).toUpperCase();
  };
  var secret = $window.localStorage.getItem('myApp.secret');
  var apiToken = encrypt(clientId, secret);

  return apiToken;
}]);
```

Ilustración 28 Ejemplo de Factory

```
myApp.run(['apiToken', function(apiToken){
    console.log(apiToken);
}])
```

Ilustración 29 Inyección como servicio

- Provider

El provider es el caso más genérico de servicio, que además de generar un servicio inyectable durante la fase de ejecución e inyectar dependencias en su definición, proporciona una API para la configuración del servicio antes de que se inicie la aplicación.

```
myApp.provider('logger', function(){
  var logToConsole = false;

  this.enableConsole = function(flag){
    logToConsole = flag;
  };

  this.$get = function(){
    return {
debug: function(msg){  if(logToConsole){ console.log(msg);} }
    };
  };
})
```

Ilustración 30 Definicion de Provider

Donde los métodos de this conforman la API de configuración, y el método this.$get equivale a una factoría.

Para configurar el servicio logger, se tiene que usar su API en la fase de configuración, inyectando el loggerProvider:

```
myApp.config(['loggerProvider', function(loggerProvider){
  loggerProvider.enableConsole(true);
}])
```

Ilustración 31 Inyección de Provider

En la fase de ejecución, se utiliza el servicio logger:

```
myApp.run(['logger', function(logger){
    logger.debug('Hello world');
}])
```

Ilustración 32 Ejemplo de servicio

2.7 Promesas en Angular JS

2.7.1 ¿Qué son las promesas?

Una promesa es un objeto, que actúa como proxy en los casos en los que no se puede retornar el verdadero valor porque aún no se conoce, pero no se puede bloquear la función esperando a que llegue.

Por ejemplo, al hacer una llamada Ajax con $http, la llamada a $http no retorna ningún valor ya que aún no tiene dicho valor, pero tampoco se puede bloquear esperando a que llegue. En realidad, el servicio $http sí que retorna un valor. Lo que retorna es una promesa, y la promesa es un proxy que en un futuro contendrá el valor.

La promesa tendrá en un futuro el valor, pero para saber en qué momento la promesa tendrá el valor, se usa una función callback para trabajar asíncronamente. Las funciones callback son piezas de código ejecutable que se pasan como argumentos a otro código. Este último es el encargado de ejecutar este argumento cuando sea posible.

2.7.2 Servicio $q:

El servicio de $q es un servicio de Angular JS que contiene toda la funcionalidad de las promesas. Está basado en la implementación de Kris Kowal. Angular JS ha hecho su propia versión para que esté todo integrado en el propio framework.

Se puede comparar el sistema de promesas al problema del productor-consumidor. La similitud es que hay una parte que generará la información, por ejemplo, el método

$http y otra parte que consumirá la información, por ejemplo, el código de la aplicación que se está realizando. Esta separación es importante ya que hay 2 objetos con los que se tiene que tratar.

En la nomenclatura de Angular JS al productor se le llama deferred y al consumidor se le llama promise. Mediante el servicio de $q, se obtiene el objeto deferred llamando al método defer(), y a partir de él, se obtiene el objeto promise llamando a la propiedad promise.

Una vez creada mediante el método defer(), una promesa se puede encontrar en alguno de los siguiente 3 estados:

- Pendiente: Aún no se sabe si se podrá o no obtener el resultado.
- Resuelta: Se ha podido obtener el resultado. Se llega a este estado llamando al método deferred.resolve().
- Rechazada: Ha habido algún tipo de error y no se ha podido obtener el resultado. Se llega a este estado llamando al método deferred.reject().

2.8 Módulo ngRoute

2.8.1 ¿Qué es y para qué sirve?

Es el módulo de Angular JS que permite crear rutas profundas en la aplicación e intercambiar vistas dependiendo de la ruta.

El módulo ("module" en la terminología anglosajona de Angular) ngRoute es un potente paquete de utilidades para configurar el enrutado y asociar cada ruta a una vista y un controlador. Sin embargo, este módulo no está incluido en la distribución de base de Angular, sino que para usarlo hay que instalarlo y luego inyectarlo como dependencia en el módulo principal de la aplicación.

Instalación de ngRoute:

Para instalarlo hay que incluir el script del código JavaScript del módulo ngRoute:

```
<script src="angular-route.js"></script>
```

Ilustración 33 Script de ngRoute

Este script se tiene que incluir después de haber incluido el script principal de Angular JS.

Inyección de dependencias:

El segundo paso es inyectar la dependencia con ngRoute en el módulo general de la aplicación. Esto se hace en la llamada al método module() con el que se inicia cualquier programa Angular JS, indicando el nombre de las dependencias a inyectar en un array:

```
angular.module("app", ["ngRoute"])
```

Ilustración 34 Inyección de ngRoute

Configurar el sistema de enrutado con $routeProvider:

El sistema de enrutado de Angular JS permite configurar las rutas que se quieren crear en la aplicación de una manera declarativa.

Tiene un par de métodos: el primero es when(), que sirve para indicar qué se debe hacer en cada ruta que se desee configurar, y el método otherwise(), que sirve para marcar un comportamiento cuando se intente acceder a cualquier otra ruta no declarada.

Esta configuración se debe realizar dentro del método config(), que pertenece al modelo. De hecho, solo se puede inyectar $routeProvider en el método config() de configuración:

```
angular.module("app", ["ngRoute"])
    .config(function($routeProvider){
    //configuración y definición de las rutas
    });
```

Ilustración 35 Configuración NgRoute

Las rutas se configuran por medio del método when() que recibe dos parámetros. Por un lado, la ruta que se está configurando y por otro lado un objeto que tendrá los valores asociados a esa ruta. Los fundamentales son:

- "controller", para indicar el controlador.
- "templateUrl", indica el nombre del archivo, o ruta, donde se encuentra el HTML de la vista que se debe cargar cuando se acceda a la ruta.

2.8.2 Ejemplo de configuración de rutas:

```
angular.module("app", ["ngRoute"])
    .config(function($routeProvider){
        $routeProvider
            .when("/", {
                controller: "appCtrl",
                templateUrl: "home.html"
            })
            .when("/descargas", {
                controller: "appCtrl",
                templateUrl: "descargas.html"
            })
            .when("/opciones", {
                controller: "appCtrl",
                templateUrl: "opciones.html"
            });
    })
    .controller("appCtrl", function(){
        //código del controlador
    });
    <script src="app.js"></script>
</body>
```

Ilustración 36 Ejemplo de rutas

En este ejemplo se ha utilizado un único controlador, pero cada vista puede tener su propio controlador.

2.9 Cómo descargar Angular JS

Acceder a la página oficial de Angular JS: https://angularjs.org/

Ilustración 37 Descarga Angular JS

Incluir el enlace que aparece en CDN en el proyecto a realizar (será necesario estar conectado a internet para poder utilizarlo), o directamente descargar el archivo “angular.min.js” desde la opción Download (no será necesaria una conexión a internet para su utilización).

3. Requisitos

La toma de requisitos es la primera fase a la que nos enfrentamos en un desarrollo software. El cliente especifica lo que el software debe hacer. A continuación, se muestran sendos requisitos, categorizados en “Funcionales” y “No Funcionales”.

3.1 Requisitos funcionales

Los requisitos funcionales son los servicios que provee una aplicación. En nuestro caso son:

1. **Login:**

   El administrador o trabajador deberán entrar en la aplicación poniendo su usuario y contraseña. Si los datos son correctos, se accederá a la aplicación.

2. **Ver Perfil:**

   El administrador o el usuario podrá ver sus datos en la aplicación.

3. **Ver Estructuras:**

   El administrador puede ver todas las estructuras que hay en la aplicación.

4. **Crear Estructura:**

   El administrador podrá crear una “estructura” que quieran revisar. Tendrá que rellenar los datos que se solicitan y pulsar el botón de crear.

5. **Modificar Estructura:**

   El administrador podrá modificar las estructuras.

6. **Eliminar Estructura:**

   El administrador podrá eliminar una estructura de la aplicación. También se eliminarán los datos de vibraciones que tuviera y el usuario asociado.

7. **Ver Trabajadores:**

   El administrador podrá ver todos los trabajadores y administradores creados.

8. **Crear Usuario:**

   El administrador podrá crear un administrador o trabajador nuevo rellenando los datos que se solicitan.

9. **Modificar Usuario:**

   El administrador podrá crear un administrador o trabajador.

10. **Borrar Usuario:**

    El administrador podrá eliminar un trabajador o administrador.

11. **Asignar Estructura:**

    El administrador podrá añadir estructuras sin asignar a un trabajador.

12. **Quitar Estructura:**

    El administrador podrá quitar una estructura asignada a un trabajador.

13. **Crear defecto:**

    El trabajador podrá crear un defecto en una estructura. También podrá modificar y eliminar el defecto.

14. **Análisis:**

    El trabajador podrá analizar los datos de una estructura asignada a él con el defecto que desee. Podrá ver en una gráfica el resultado y sacar un informe en pdf.

3.2 Requisitos no funcionales

Especifica los criterios que pueden usarse para juzgar la operación de un sistema. Son requisitos que no describen información a guardar ni funciones a realizar, solamente características de funcionamiento. Son los siguiente:

1. **Disponibilidad**: Estará disponible para el trabajador desde cualquier parte que pueda conectarse a la red.
2. **Estabilidad**: Pretendemos que el sistema tenga los menos fallos posibles.
3. **Mantenibilidad**: Minimizaremos el esfuerzo para conservar el funcionamiento de la web.
4. **Escalabilidad**: Permite incorporar nuevas funciones sin afectar a las existentes.
5. **Usabilidad**: La web será de fácil manejo e intuitiva. Contará con manual de usuario.

4. Análisis

En este capítulo usaremos los requisitos captados anteriormente para identificar las entidades del software y sus casos de uso.

4.1 Diagrama Entidad-Relación

Detectadas las funciones que realice nuestro sistema, identificaremos las entidades y relaciones existentes entre ellas para poder modelar nuestra base de datos:

Entidades:

1. Usuario: Representado por las personas que utilizaran la aplicación, tanto administradores como trabajadores.
2. Estructura: Cada una de las estructuras que se van a analizar.
3. Defectos: Los límites que se buscan en una estructura.
4. Vibraciones: Datos que se extraen de las estructuras.

Relaciones:

1. **Crea**: Un usuario Administrador puede crear N estructuras como máximo, y una estructura solo puede ser creada por 1 usuario.
2. **Registra**: Un usuario Administrador puede registrar N usuarios.
3. **Añade**: Un usuario Trabajador puede añadir N defectos y un defecto puede ser creado por 1 usuario.
4. **Une**: Un usuario puede unirse a N estructuras, pero una estructura solo puede estar unida a 1 usuario.
5. **Tiene**: Una estructura puede tener N vibraciones asociadas, pero una vibración solo puede pertenecer a 1 estructura.

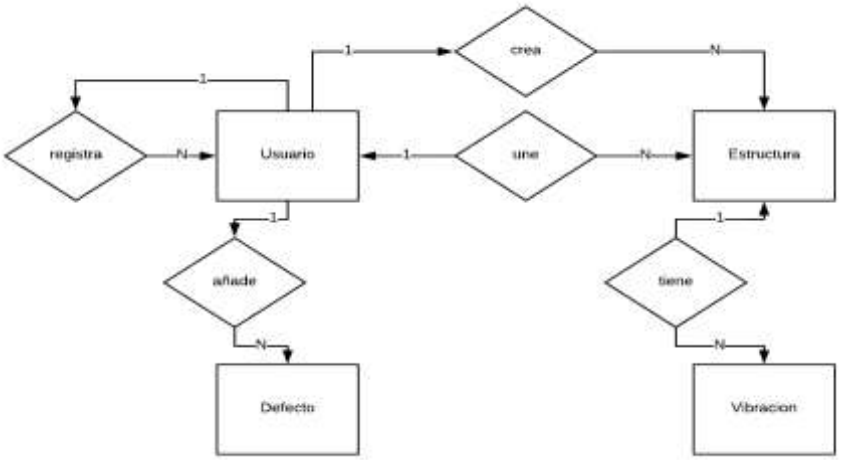

Ilustración 38 Diagrama Entidad-Relación

4.2 Casos de uso

En este apartado detallaremos y documentaremos todas las acciones posibles que pueden llevarse a cabo en la aplicación. Para realizar dicha tarea, usaremos el **Modelo de los Casos de Uso** especificado por el **Lenguaje Unificado de Modelado (UML)** como herramienta.

El Lenguaje Unificado de Modelado (UML), es una metodología para definir y documentar cada una de las acciones y/o actividades que pueden realizarse dentro de un sistema de información. Se emplearán los Casos de Uso para representar que acciones se pueden realizar en el sistema[6].

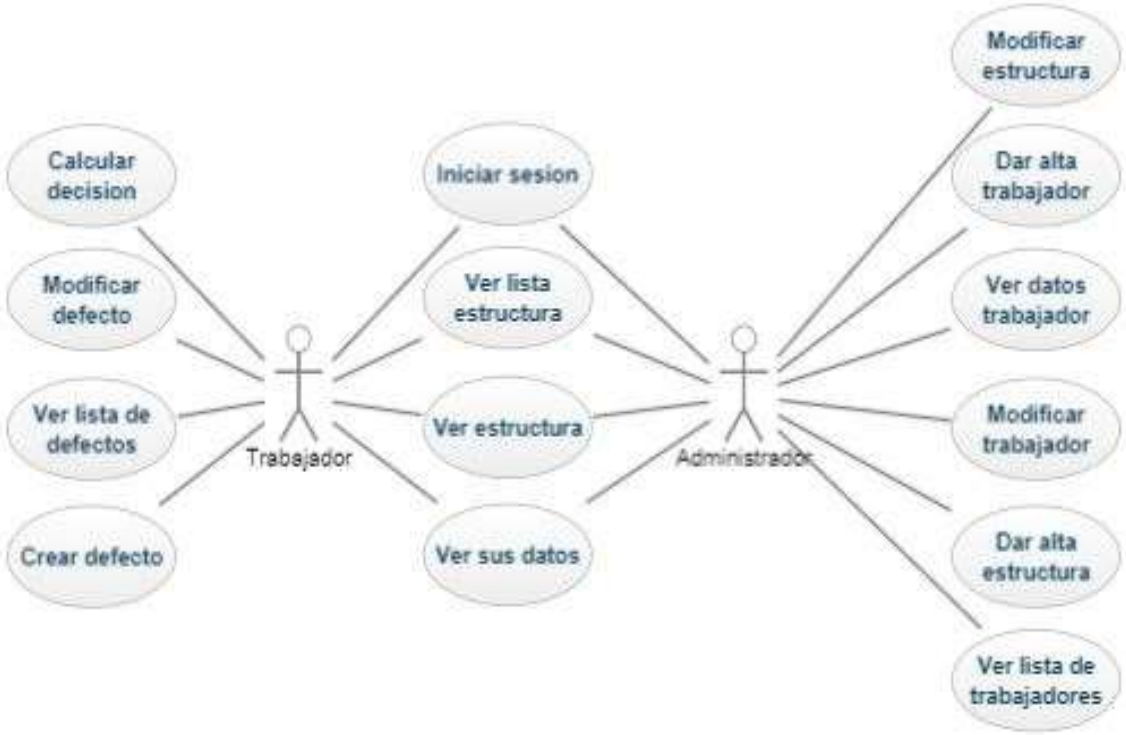

Ilustración 39 Diagrama Casos de uso

Los actores principales de la aplicación son los Administradores y los trabajadores.

A continuación, se detalla el listado con todos los casos de uso:

Código	Nombre
CU01	Iniciar sesión
CU02	Ver sus datos
CU03	Ver estructura
CU04	Calcular decisión
CU05	Ver lista de trabajadores
CU06	Ver datos trabajador
CU07	Modificar trabajador
CU08	Dar alta trabajador
CU09	Dar alta estructura
CU10	Ver lista estructuras
CU11	Modificar estructura
CU12	Ver lista defectos
CU13	Crear defecto
CU14	Modificar defecto

CU01 - Iniciar sesión

Título	**Iniciar sesión.**
Actores	Administrador y trabajador.
Descripción	El usuario inicia sesión en la web.
Pre-condición	
Post-condición	El usuario es logueado y entra en la pantalla principal.
Escenario principal	1. El usuario rellena los campos “usuario” y “contraseña”. 2. El usuario pulsa el botón en el botón login. 3. El sistema comprueba que los datos son correctos y coinciden. 4. El usuario entra en la página principal.
Escenario alternativo	3.b El sistema comprueba que los datos no son correctos. 3.c El sistema muestra un mensaje de error.

CU02 - Ver sus datos

Título	**Ver sus datos.**
Actores	Administrador y trabajador.
Descripción	El usuario consulta sus datos personales.
Pre-condición	El usuario debe estar logueado.
Post-condición	Cuando termina de ver sus datos volverá a la pantalla principal.
Escenario principal	1. El usuario pulsa el botón en la opción datos del menú. 2. El sistema carga la vista “ver datos”. 3. El usuario pulsa el botón volver.
Escenario alternativo	

CU03 - Ver estructura

Título	**Ver estructura.**
Actores	Administrador y trabajador.
Descripción	El usuario puede ver los datos la estructura seleccionada.
Pre-condición	El usuario debe estar logueado.
Post-condición	Aparece en pantalla todos los datos y se activa el botón simular.
Escenario principal	1. Usuario pulsa el botón en la estructura que desea ver. 2. El sistema carga una nueva ventana con los datos de la estructura. 3. El usuario pulsa el botón en el botón volver.
Escenario alternativo	

CU04 - Calcular decisión

Título	**Calcular decisión.**
Actores	Trabajador.
Descripción	El usuario pide el apoyo a la decisión de qué hacer con la estructura.
Pre-condición	El usuario debe estar logueado y en la página de ver estructura.
Post-condición	
Escenario principal	1. El usuario pulsa el botón “Análisis” 2. El sistema carga una ventana con los defectos y dos cajas para colocar el intervalo de tiempo. 3. El usuario indica el intervalo de tiempo que quiere ver y el defecto a analizar 4. El sistema muestra los resultados del análisis. 5. El usuario pulsa el botón “Informe”. 6. El sistema genera y descarga el informe. 7. El usuario pulsa el botón volver.
Escenario alternativo	

CU05 - Ver lista de trabajadores

Título	**Ver lista trabajadores.**
Actores	Administrador.
Descripción	El usuario consulta la lista de trabajadores.
Pre-condición	El usuario está logueado como administrador.
Post-condición	Cuando termina de ver los trabajadores vuelve a la pantalla principal.
Escenario principal	1. El usuario pulsa el botón “trabajadores” del menú. 2. El sistema carga una lista con los trabajadores que tiene la empresa. 3. El usuario pulsa el botón volver.
Escenario alternativo	

CU06 - Ver datos trabajador

Título	**Ver datos trabajador.**
Actores	Administrador.
Descripción	El usuario ve los datos de un trabajador.
Pre-condición	El usuario está logueado como administrador y se encuentra en la ventana de ver lista trabajadores.
Post-condición	El usuario vuelve a la pantalla lista de trabajadores.
Escenario principal	1. El usuario pulsa el botón “ver datos” de un trabajador. 2. El sistema carga una pantalla con los datos del trabajador. 3. El usuario pulsa el botón volver.
Escenario alternativo	3a. El usuario pulsa el botón “Modificar”. 4a. El sistema carga el caso de uso “Modificar usuario”. 3b. El usuario pulsa el botón “Eliminar”. 4b. El sistema carga el caso de uso “Eliminar usuario”.

CU07 - Modificar trabajador

Título	**Modificar trabajador.**
Actores	Administrador.
Descripción	El usuario modifica los datos de un trabajador.
Pre-condición	El usuario esta logueado como administrador y se encuentra en la pantalla de “datos trabajador”
Post-condición	El usuario vuelve a la pantalla “datos trabajador” con los datos del usuario modificados.
Escenario principal	1. El usuario pulsa el botón “Modificar” 2. El sistema habilita los campos de datos para su modificación. 3. El usuario modifica los datos necesarios. 4. El usuario pulsa el botón “Aceptar”. 5. El sistema comprueba que no hay datos erróneos. 6. El sistema muestra una pantalla de confirmación de cambios. 7. El usuario pulsa el botón “Aceptar” 8. El sistema cambia los datos del usuario y carga la pantalla “datos trabajador”.
Escenario alternativo	1-4a. El usuario pulsa el botón cancelar. 5a. El sistema carga la pantalla “Datos trabajador”. 6b. El sistema muestra en rojo los datos erróneos. 7b. El usuario modifica los datos erróneos. 8b. El usuario pulsa el botón “Aceptar”. 9b. El sistema comprueba que no hay datos erróneos. 7c. El usuario pulsa el botón “Cancelar”. 8c. El sistema vuelve a la pantalla “Modificar trabajador”.

CU08 - Dar alta trabajador

Título	**Dar alta trabajador.**
Actores	Administrador.
Descripción	El usuario da de alta a un trabajador.
Pre-condición	El usuario está logueado como administrador y se encuentra en la ventana “Ver lista trabajadores”
Post-condición	El usuario vuelve a la ventana “Ver lista trabajadores”
Escenario principal	1. El usuario pulsa el botón “Alta trabajador” 2. El sistema carga una ventana con un formulario. 3. El usuario rellena los campos necesarios. 4. El pulsa el botón “Aceptar” 5. El sistema comprueba que los datos no son erróneos. 6. El sistema muestra una ventana de confirmación 7. El usuario pulsa el botón “Aceptar”. 8. El sistema añade los datos y vuelve a la ventana “Ver lista trabajadores”.
Escenario alternativo	1-4a. El usuario pulsa el botón “Cancelar”. 5a. El sistema vuelve a la ventana “Ver lista trabajadores”. 6b. El sistema muestra en rojo los datos erróneos. 7b. El usuario modifica los datos erróneos. 8b. El usuario pulsa el botón “Aceptar”. 7c. El usuario pulsa el botón “Cancelar”. 8c. El sistema vuelve a la pantalla “Ver lista trabajadores”.

CU09 - Dar alta estructura

Título	**Dar alta estructura.**
Actores	Administrador.
Descripción	El usuario da de alta una estructura.
Pre-condición	El usuario está logueado como administrador y se encuentra en la ventana “Ver lista estructuras”
Post-condición	El usuario vuelve a la pantalla principal.
Escenario principal	1. El usuario pulsa el botón “Alta estructura”. 2. El sistema carga una ventana con el formulario. 3. El usuario rellena los campos necesarios. 4. El pulsa el botón “Aceptar” 5. El sistema comprueba que los datos no son erróneos. 6. El sistema muestra una ventana de confirmación 7. El usuario pulsa el botón “Aceptar”. 8. El sistema añade los datos y vuelve a la ventana principal.
Escenario alternativo	1-4a. El usuario pulsa el botón “Cancelar”. 5a. El sistema vuelve a la ventana “Ver lista trabajadores”. 6b. El sistema muestra en rojo los datos erróneos. 7b. El usuario modifica los datos erróneos. 8b. El usuario pulsa el botón “Aceptar”. 7c. El usuario pulsa el botón “Cancelar”. 8c. El sistema vuelve a la pantalla principal.

CU10 - Ver lista estructuras

Título	**Ver lista Estructuras.**
Actores	Administrador y trabajador.
Descripción	El usuario consulta la lista de estructuras.
Pre-condición	El usuario está logueado como administrador.
Post-condición	Cuando termina de ver las estructuras vuelve a la pantalla principal.
Escenario principal	1. El usuario pulsa el botón estructuras del menú. 2. El sistema carga una lista con las estructuras que tiene la empresa. 3. El usuario pulsa el botón volver.
Escenario alternativo	

CU11 - Modificar estructura

Título	**Modificar estructura.**
Actores	Administrador.
Descripción	El usuario modifica los datos de una estructura.
Pre-condición	El usuario esta logueado como administrador y se encuentra en la pantalla de “datos estructuras”
Post-condición	El usuario vuelve a la pantalla “datos estructura” con los datos de la estructura modificadas.
Escenario principal	1. El usuario pulsa el botón “Modificar” 2. El sistema habilita los campos de datos para su modificación. 3. El usuario modifica los datos necesarios. 4. El usuario pulsa el botón “Aceptar”. 5. El sistema comprueba que no hay datos erróneos. 6. El sistema muestra una pantalla de confirmación de cambios. 7. El usuario pulsa el botón “Aceptar” 8. El sistema cambia los datos del usuario y carga la pantalla “datos estructura”.
Escenario alternativo	1-4a. El usuario pulsa el botón cancelar. 5a. El sistema carga la pantalla “Datos Estructura”. 6b. El sistema muestra en rojo los datos erróneos. 7b. El usuario modifica los datos erróneos. 8b. El usuario pulsa el botón “Aceptar”. 9b. El sistema comprueba que no hay datos erróneos. 7c. El usuario pulsa el botón “Cancelar”. 8c. El sistema vuelve a la pantalla “Modificar Estructura”.

CU12 - Ver lista defectos

Título	**Ver lista defectos.**
Actores	Trabajador.
Descripción	El usuario consulta la lista de defectos
Pre-condición	El usuario está logueado como trabajador
Post-condición	Cuando termina de ver los trabajadores vuelve a la pantalla principal
Escenario principal	1. El usuario pulsa el botón “Defectos” del menú 2. El sistema carga una lista con los defectos que tiene cargados. 3. El usuario pulsa el botón
Escenario alternativo	

CU13 - Crear defecto

Título	**Crear defecto.**
Actores	Trabajador.
Descripción	El usuario crea un defecto.
Pre-condición	El usuario está logueado como trabajador y se encuentra en la ventana “Ver defectos”
Post-condición	El usuario vuelve a la pantalla principal
Escenario principal	1. El usuario pulsa el botón “Nuevo Defecto”. 2. El sistema carga una ventana con el formulario. 3. El usuario rellena los campos necesarios. 4. El usuario pulsa el botón “Aceptar”. 5. El sistema comprueba que los datos no son erróneos. 6. El sistema muestra una ventana de confirmación. 7. El usuario pulsa el botón “Aceptar”. 8. El sistema añade los datos y vuelve a la ventana principal.
Escenario alternativo	1-4a. El usuario pulsa el botón “Cancelar”. 5a. El sistema vuelve a la ventana “Ver defectos”. 6b. El sistema muestra en rojo los datos erróneos. 7b. El usuario modifica los datos erróneos. 8b. El usuario pulsa el botón “Aceptar”. 7c. El usuario pulsa el botón “Cancelar”. 8c. El sistema vuelve a la pantalla principal.

CU14 - Modificar defecto

Título	**Modificar defecto.**
Actores	Trabajador.
Descripción	El usuario modifica los datos de un defecto.
Pre-condición	El usuario esta logueado como trabajador y se encuentra en la pantalla de “Ver lista defectos”
Post-condición	El usuario vuelve a la pantalla “Ver lista defectos” con los datos del defecto modificado.
Escenario principal	1. El usuario pulsa el botón “Modificar”. 2. EL sistema habilita los campos de datos para su modificación. 3. El usuario modifica los datos necesarios. 4. El usuario pulsa el botón “Aceptar”. 5. El sistema comprueba que no hay datos erróneos. 6. El sistema muestra una pantalla de confirmación de cambios. 7. El usuario pulsa el botón “Aceptar”. 8. El sistema cambia los datos del usuario y carga la pantalla “Ver lista defectos”.
Escenario alternativo	1-4a. El usuario pulsa el botón cancelar. 5a. El sistema carga la pantalla “Ver lista defectos”. 6b. El sistema muestra en rojo los datos erróneos. 7b. El usuario modifica los datos erróneos. 8b. El usuario pulsa el botón “Aceptar”. 9b. El sistema comprueba que no hay datos erróneos. 7c. El usuario pulsa el botón “Cancelar”. 8c. El sistema vuelve a la pantalla “Ver lista defectos”.

5. Diseño

En este apartado abarcaremos todo lo relacionado con la implementación de la aplicación junto a algunos detalles del código.

5.1 Paso de tablas

Tras la fase de análisis, se realiza el paso a tablas. Con el resultado se obtienen 4 tablas representadas en la siguiente figura.

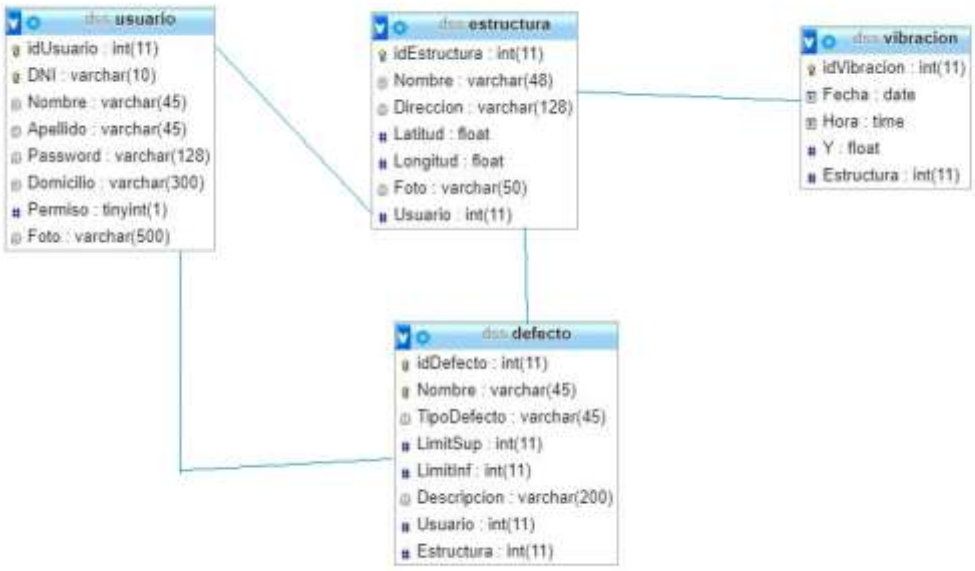

Ilustración 40 Paso de tablas

1. Usuario: Recoge la información de los usuarios de la aplicación. Se almacena:
    - idUsuario (clave numérica auto-incremental)
    - DNI (Que es único)
    - Nombre
    - Apellido
    - Password
    - Domicilio
    - Permiso

2. Estructura: Esta tabla almacena las estructuras de las que se verán los datos. Para cada estructura se almacena:
    - idEstructura (clave numérica auto-incremental)
    - Nombre
    - Dirección

- Latitud
- Longitud
- Foto
- Usuario (clave foránea procedente de la tabla usuario -> idUsuario)

3. Vibración: Esta tabla contiene todos los datos de vibración de cada estructura. Se almacena:
    - idVibracion (clave numérica auto-incremental)
    - Fecha
    - Hora
    - Y (Valor de la vibración)
    - Estructura (clave foránea procedente de la tabla estructura -> idEstructura)

4. Defecto: Tabla que recoge los distintos defectos que se analizaran en la estructura. Contiene:
    - idDefecto (clave numérica auto-incremental)
    - Nombre
    - Tipo de defecto
    - Límite Superior
    - Límite Inferior
    - Descripción
    - Usuario (Clave foránea procedente de la tabla usuario ->idUsuario)
    - Estructura (Clave foránea procedente de la tabla estructura ->idEstructura)

Ilustración 41 Imagen de la Base de datos

	idUsuario	DNI	Nombre	Apellido	Password	Domicilio	Permiso	Foto
Copiar Borrar	1	30636974G	Pedro	Gavilan Lozano	$2y$10$wbeHGmecNCyhY2AQ2EWG5uEbXNJU6Wn4TaXWWGvaf5qR.mwwvT4S	c/Eguiluz, nº2, 3ºA	0	NULL
Copiar Borrar	4	30536974G	Antonio	Rodriguez Brotons	$2y$10$4ocmMGKMD5q8GkVdstOKMOMVk4YSb9tmLQxO4Ebb5CSNbds Om	c/Heroes de Sostoa, nº2, 3ºA	1	NULL
Copiar Borrar	11	32061984c	Juan	Ramirez Fernandez	$2y$10$mCRdkR0B4udKJi8PtheR/x0ysK7HWGcBGBmQVTBE1JiPPuU9.i	c/Heroe de Sostoa 22	1	NULL
Copiar Borrar	12	31620446L	Juan	Ramirez Valaro	$2y$10$UZPAnagK/YDZccc1g01DX.qllyTmJ/p0BvWPIptF.xq/UYBpppOUg2	C/Maria Antonia Jesus Tirado 8, Jerez de la Frontera	0	NULL

Ilustración 42 Tabla Usuario

	idEstructura	Nombre	Direccion	Latitud	Longitud	Foto	Usuario
Editar Copiar Borrar	1	Auroras	Avenida de la Aurora, 15	36.7162	-4.4346	NULL	12
Editar Copiar Borrar	6	Partizan	Avenida de Partizan, 15	30.7163	-5.43434	NULL	12
Editar Copiar Borrar	7	ETSI Universidad de Malaga	Bulevar Luis Pasteur	36.7159	-4.47731	NULL	12
Editar Copiar Borrar	8	Nemo	C/Wallaby, 42, Sidney	35.24	-6.25	NULL	NULL

Ilustración 43 Tabla Estructura

	idVibracion	Fecha	Hora	Y	Estructura
Editar Copiar Borrar	9	2017-10-28	07:05:07	904	1
Editar Copiar Borrar	8	2017-10-28	07:05:06	920	1
Editar Copiar Borrar	7	2017-10-28	07:05:05	904	1
Editar Copiar Borrar	6	2017-10-28	07:05:04	912	1
Editar Copiar Borrar	5	2017-10-28	07:05:03	864	1
Editar Copiar Borrar	4	2017-10-28	07:05:02	904	1
Editar Copiar Borrar	3	2017-10-28	07:05:01	792	1

Ilustración 44 Tabla Vibración

	idDefecto	Nombre	TipoDefecto	LimitSup	LimitInf	Descripcion	Usuario	Estructura
Editar Copiar Borrar	16	Prueba11	Entre	925	900	Prueba11	12	1
Editar Copiar Borrar	17	Prueba2	Por Encima	950	NULL	Prueba2	12	1
Editar Copiar Borrar	18	Prueba3	Por Debajo	NULL	850	Prueba3	12	7
Editar Copiar Borrar	19	Prueba4	Por Encima	1000	NULL	Prueba4	12	1
Editar Copiar Borrar	23	Prueba5	Por Debajo	985	920	Prueba5	12	1

Ilustración 45 Tabla Defecto

5.2 Controladores, Vistas y Modelos utilizados

En este apartado indicaremos el modelo que se utiliza y la vista asociada para cada controlador:

- **datosUsuarioCtrl**: Este controlador realiza un acceso a la base de datos para mostrar los datos y las estructuras asociadas del usuario que el administrador desea ver. Dichos datos se almacenan en la propiedad “model” para los datos del usuario y en la propiedad “estruct” para el listado de las estructuras. El archivo “ver.html” es la vista asociada.

- **usuariosListadoCtrl**: Este controlador accede a la base de datos para mostrar el listado de usuarios que hay registrados. Los datos se almacenan en la propiedad “model”. También hace la función de eliminar a un usuario del listado. La vista asociada a este controlador es “listado.html”.

- **listadoEstructurasCtrl**: Este controlador accede a la base de datos para mostrar el listado de estructuras que hay registrados. Los datos se almacenan en la propiedad “model”. También hace la función de eliminar a una estructura del listado. La vista asociada a este controlador es “listEstructura.html”.

- **verEstructurasCtrl**: Este controlador se encarga de mostrar los datos de una estructura del listado de estructuras. Realiza un acceso a la base de datos y almacena los datos en el modelo “model”. La vista asociada es “verEstructura.html”.

- **registrarUserCtrl**: Este controlador se encarga de realizar la creación de un usuario en la aplicación. La vista asociada a este controlador es “crearUser.html”.

- **modificarUserCtrl**: Este controlador realiza la modificación de los datos de un usuario. La vista asociada es “modificarUser.html”.

- **registrarEstructuraCtrl**: Este controlador se encarga de realizar la creación de una estructura en la aplicación. La vista asociada a este controlador es “crearEstructura.html”.

- **modificarEstructuraCtrl**: Este controlador realiza la modificación de los datos de un usuario. La vista asociada es “modificarEstructura.html”

- **loginCtrl**: Se encarga de mostrar el formulario de acceso a la aplicación. Dependiendo de si es un trabajador o un administrador irá a una vista u a otra y guarda el nombre y permiso en el scope global. El archivo “login.html” es la vista asociada.

- **useriniCtrl**: Se encarga de mostrar el mensaje de bienvenida al trabajador además del listado de estructuras que tiene asociadas a su cuenta. El archivo “userini.html” es la vista asociada.

- **admininiCtrl**: Se encarga de mostrar el mensaje de bienvenida al administrador del sistema. El archivo “botones.html” es la vista asociada.

- **addestrucCtrl**: Este controlador se encarga de asociar una estructura libre a un trabajador. Se hace un acceso a la base de datos para recoger las estructuras “libres” (que no tienen un trabajador asociado). La vista asociada es “addEstruc.html”.

- **delestrucCtrl**: Este controlador se encarga de eliminar estructuras que tiene asociadas un trabajador. Se hace un acceso a la base de datos para guardar en la propiedad “model”. La vista asociada es “delestrucCtrl.html”.

- **userverEstructurasCtrl**: Este controlador se encarga de mostrar los datos de la estructura elegida al trabajador. Realiza un acceso a la base de datos donde recoge los datos de la estructura y de las vibraciones que ha recogido a lo largo del mes y los guarda en el modelo “model”. La vista asociada es “userverstruct.html”.

- **defectosCtrl**: Este controlador realiza un acceso a la base de datos para mostrar los defectos que tiene creados el trabajador. El modelo de datos es una propiedad del scope llamada “model”. La vista asociada es “defectos.html”.

- **addDefectosCtrl**: Este controlador permite crear un defecto en la base de datos. La información se recoge de un formulario y se almacena en la propiedad “model”. La vista asociada es “addDefecto.html”.

- **modificarDefectoCtrl**: Este controlador permite modificar un defecto de la base de datos. Recoge los datos de un formulario y

los almacena en la propiedad “model”. La vista asociada es “modificarDefecto.html”.

- **userelecdefectoCtrl**: Este controlador realiza un acceso a la base de datos para mostrar los defectos que ha creado el trabajador y elegirlo para realizar el análisis. La vista asociada es “userelecdefecto.html”.

- **analisisCtrl**: Este controlador realiza el análisis de la estructura, accede a la base de datos y muestra el análisis realizado con los datos del defecto elegido y las vibraciones que ha guardado. También crea el informe en pdf y la descarga en el equipo. La vista asociada es “UserAnalisis.html”.

5.3 Ejemplos de diseño y del código desarrollado

5.3.1 Aplicación web en una sola pagina

Angular JS permite diseñar aplicaciones en una sola página. Para lograrlo, el archivo “index.html” tendrá X partes diferenciadas:

Cuando el navegador cargue la URL, cargara estas X partes. Pero de aquí en adelante solo las partes del marco negro () no se recargarán otra vez. Solo se recargará la nueva vista, por tanto, ahorramos trabajo al navegador y tendremos más fluidez a la hora de movernos por la página.

Para cargar las vistas en la página utilizamos la directiva ng-view:

```
<body>

        <ng-view />
```

Ilustración 46 Código ubicado en index.html

5.3.2 Ejemplo de uso de configuración de rutas

Gracias al menú, podremos acceder a varias partes de la página. Por eso cada opción del menú está vinculado a una ruta. En la siguiente imagen podemos ver las rutas vinculadas al menú. Por ejemplo, la opción “Perfil” tiene vinculada la ruta “”#!/perfil/”.

```
<div id="sidebar-wrapper">
    <ul class="sidebar-nav">
        <li class="sidebar-brand">
            <a href="#!/userini">Principal</a>
        </li>
        <li>
            <a href="#!/perfil/{{globals.usuario.id}}">Perfil</a>
        </li>
        <li>
            <a href="#!/userini">Estructuras</a>
        </li>
        <li>
            <a href="#!/defectos/{{globals.usuario.id}}">Defectos</a>
        </li>
        <li>
            <a ng-click="logout()">Salir</a>
        </li>
    </ul>
</div>
```

Ilustración 47 Código ubicado en el archivo "userini.html"

En la siguiente imagen veremos la configuración de rutas. Por ejemplo, la ruta “/estructuras”, se cargará la vista “listEstructuras.html” y el controlador que se encargará de manejar la lógica de esta vista será “listadoEstructurasCtrl”:

```
when('/user', {
  templateUrl: 'htmls/listado.html',
  controller: 'usuariosListadoCtrl'
})
    .when('/user/:idUsuario', {
    templateUrl: 'htmls/ver.html',
    controller: 'datosUsuarioCtrl'
})
    .when('/estructuras', {
        templateUrl: 'htmls/listEstructuras.html',
        controller: 'listadoEstructurasCtrl'
})
    .when('/estructuras/:idEstructura', {
        templateUrl: 'htmls/verEstructura.html',
        controller: 'verEstructurasCtrl'
})
```

Ilustración 48 Código ubicado en el archivo "app.js"

5.3.3 Ejemplo de uso del scope

El scope sirve para establecer la comunicación entre controlador y vista. En la siguiente imagen veremos un ejemplo de ello:

```
$http.post('http://localhost/apiPhp/V1/usuarios/obtenerUsuariosId', $idUsuario).then(function (r) {
    console.log(r.data);
    $scope.model = r.data;
     estructuser($idUsuario);
```

Ilustración 49 Código ubicado en controllers.js

```
<p>{{ model.datos.Nombre }}</p><br>

<p>{{ model.datos.Apellido }}</p><br>

<p>{{ model.datos.DNI }}</p><br>

<p>{{ model.datos.Direccion }}</p><br>

<p ng-if="model.datos.Permiso == 1">Administrador</p>
<p ng-if="model.datos.Permiso == 0">Trabajador</p>
```

Ilustración 50 Código ubicado en perfil.html

5.3.4 Ejemplo de uso del rootScope

Además del scope, también existe el rootScope cuyo ámbito es accesible desde cualquier lugar de la aplicación.

```
$rootScope.globals = {
    usuario: {
        id: r.data.usuario.idUsuario,
        permiso: r.data.usuario.Permiso,
        perm: per
    }
};
```

Ilustración 51 Código ubicado en el archivo controllers.js

Como se puede observar, puedo acceder a las variables rootscope desde cualquier vista, aunque sea otro controlador quien lo gestione.

```
<ul class="nav navbar-top-links navbar-right">
    <li class="dropdown"> Conectado como: {{globals.usuario.perm}}.
    </li>

    <!-- /.dropdown -->
    <li class="dropdown">
        <a href="#!/perfil/{{globals.usuario.id}}"><i class="glyphicon glyphicon-user"></i></a>
    </li>
    <!-- /.dropdown -->
</ul>
```

Ilustración 52 Código ubicado en addDefecto.html

5.3.5 Ejemplo de uso de inyección de dependencias

En Angular JS existe un servicio que permite la comunicación con el servidor usando el protocolo http, este servicio es $http.

En la siguiente imagen veos la inyección de la dependencia en el controlador para su uso posterior.

```
empleadoControllers.controller('usuariosListadoCtrl', ['$scope', '$http', 'auth', function ($scope, $http, auth) {
```

Ilustración 53 Código ubicado en controllers.js

```
function listadoUsuario(){
    $http.get('http://localhost/apiPhp/V1/usuarios/obtenerUsuarios').then(function (r) {
        console.log(r.data);
        $scope.model = r.data;
    });
}
```

Ilustración 54 Código ubicado en controller.js

5.4 Configuración de la parte del servidor

En el servidor, el trabajo realizado contra la base de datos está en la carpeta vistas:

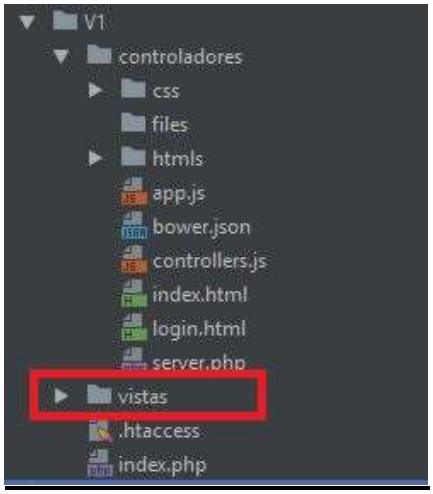

Ilustración 55 Imagen de la estructura de carpetas

Si accedemos a esta carpeta, encontramos los archivos donde se realizan las consultas y modificaciones de la base de datos:

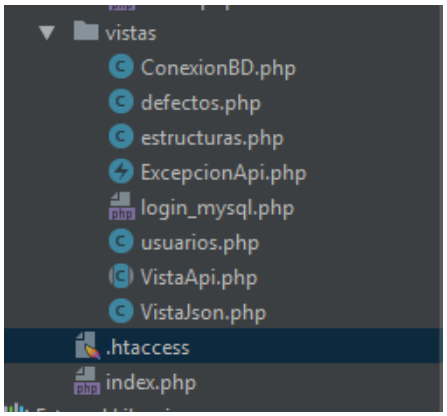

Ilustración 56 Imagen de la estructura de carpetas

Las conexiones a la base de datos se han realizado mediante PDO:

```php
public function obtenerBD(){
    if(self::$pdo == null){
        self::$pdo = new PDO(
            dsn: 'mysql:dbname=' . BASE_DE_DATOS .
            ';host=' . NOMBRE_HOST . ";",
            username: USUARIO, passwd: CONTRASENA,
            array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8")
        );

        self::$pdo->setAttribute( attribute: PDO::ATTR_ERRMODE, value: PDO::ERRMODE_EXCEPTION);
    }
    return self::$pdo;
}
```

Ilustración 57 Código ubicado en ConexionBD.php

Las operaciones de consulta/modificación de la base de datos se han hecho combinando lenguaje MySQL y php:

```php
$comando = "SELECT * FROM " . self::NOMBRE_TABLA;

$sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($comando);

if($sentencia->execute()){
    http_response_code( response_code: 200);
    return
        [
            "estado" => self::ESTADO_EXITO,
            "datos" => $sentencia->fetchAll( fetch_style: PDO::FETCH_ASSOC)
        ];
```

Ilustración 58 Código ubicado en usuarios.php

6. Guía de Usuario

En las siguientes páginas se mostrará una guía de la aplicación para que el usuario conozca paso a paso las opciones disponibles. Como la aplicación es diferente para los dos tipos de usuario que hay, esta guía estará dividida en dos partes. La parte “Guía del Administrador” y la parte “Guía del Trabajador”.

6.1 Login

Esta es la única parte común de la aplicación. Se introduce usuario y contraseña y se pulsa en “Ingresar”. Si ambos datos son correctos, el usuario accederá a la aplicación.

Ilustración 59 Login

Ilustración 60 Bienvenida Administrador

Ilustración 61 *Bienvenida usuario*

6.2 Guía del administrador

Comenzaremos con el administrador, como vemos en la siguiente imagen, vemos las opciones que puede realizar en el lateral izquierdo de la pantalla.

Ilustración 62 Menú lateral izquierdo

6.2.1 Perfil

El administrador puede consultar sus datos personales en esta sección. Se accede por el control lateral de la aplicación.

Ilustración 63 Perfil usuario

6.2.2 Estructuras

En esta sección el administrador puede ver las estructuras creadas en la aplicación. Se accede desde el menú lateral y podremos registrar, modificar y eliminar una estructura.

Ilustración 64 Listado estructuras

6.2.3 Registrar estructura

Para realizar este proceso, hay que clicar en el botón “Registrar” de la ventana de Estructuras. Se nos mostrará una nueva ventana con los datos a introducir. Introduciremos los datos solicitados y pulsaremos el botón “Agregar”, nos pedirá confirmación y la estructura se añadirá a la aplicación.

Para cancelar este proceso podemos pulsar el boton “Volver” en cualquier momento.

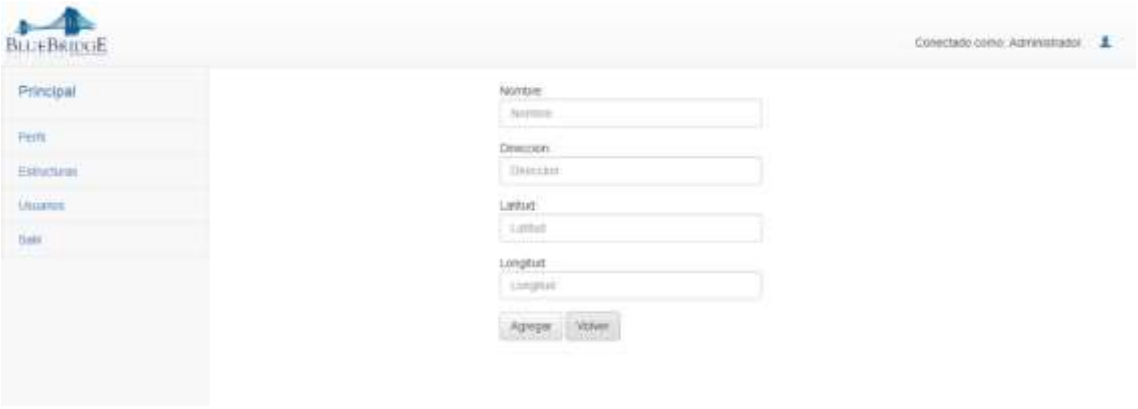

Ilustración 65 Creación estructura

6.2.4 Actualizar estructura

Para actualizar una estructura, debemos clicar en el botón del ojo  que se encuentra al lado de la estructura a modificar. Se abrirá una nueva página con los datos de la estructura y clicaremos el botón “Modificar”.

Esto nos llevara a un formulario donde podemos cambiar los datos de la estructura. Cuando terminemos de modificar, clicaremos el botón actualizar. Nos pedirá confirmación y al aceptar la estructura estará modificada.

Para cancelar el proceso podemos pulsar el botón volver en cualquier momento.

Ilustración 66 Botón ver estructura

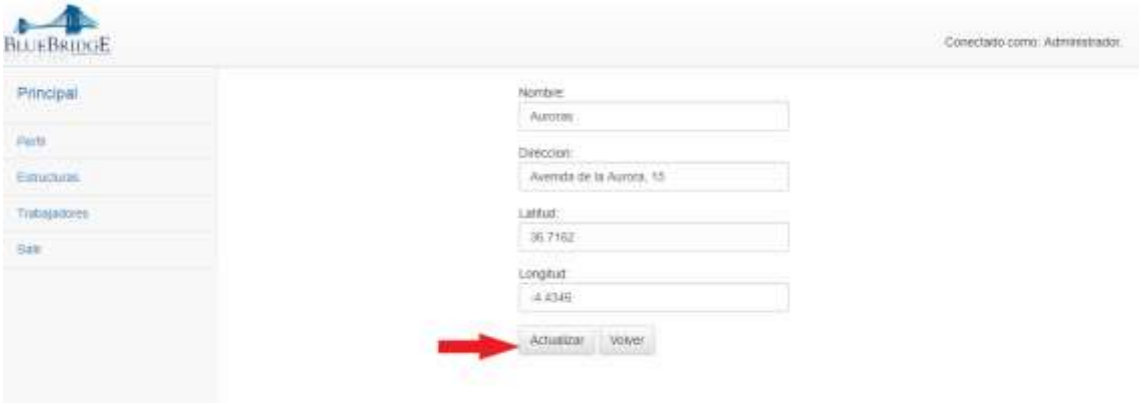

Ilustración 67 Modificar estructura

6.2.5 Eliminar estructura

Para eliminar una estructura, debemos clicar en el botón rojo de papelera que se encuentra al lado de la estructura a eliminar.

Nos pedirá confirmación de la eliminación y entonces la estructura desaparecerá del listado.

Ilustración 68 Eliminar estructura

6.2.6 Trabajadores

En esta sección el administrador puede ver los trabajadores creados en la aplicación. Se accede desde el menú lateral y podremos registrar, modificar y eliminar un trabajador.

Ilustración 69 Listado trabajadores

6.2.7 Registrar trabajador

Para realizar este proceso, hay que clicar en el botón “Registrar” de la ventana de Trabajadores. Se nos mostrará una nueva ventana con los datos a introducir.

Cuando terminemos de introducir los datos, clicaremos el botón “Agregar”, nos pedirá confirmación y se añadirá al trabajador a la aplicación.

Ilustración 70 Botón Registrar

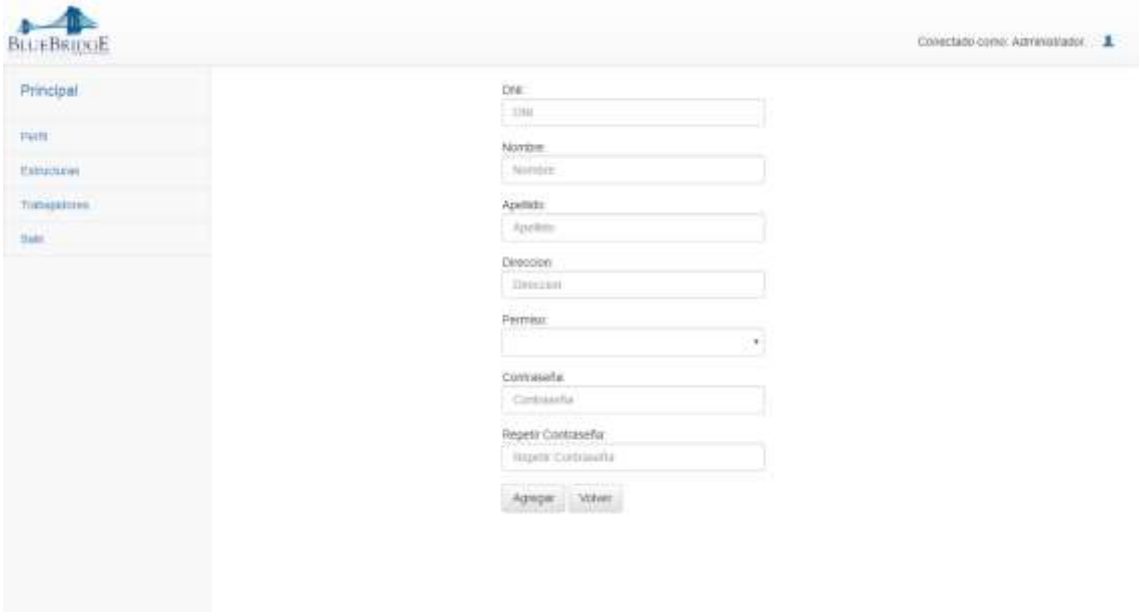

Ilustración 71 Formulario registrar trabajador

6.2.8 Ver trabajador

Para acceder a los datos de un trabajador de la aplicación hay que clicar en el botón del ojo que se encuentra al lado de trabajador elegido. Nos aparecerá una ventana nueva con los datos de dicho trabajador.

Ilustración 72 Datos trabajador

6.2.9 Modificar trabajador

Para actualizar una estructura, debemos clicar en el botón del ojo que se encuentra al lado de la estructura a modificar. Se abrirá una nueva página con los datos de la estructura y clicaremos el botón “Modificar”.

Esto nos llevara a un formulario donde podemos cambiar los datos de la estructura. Cuando terminemos de modificar, clicaremos el botón actualizar. Nos pedirá confirmación y al aceptar la estructura estará modificada.

Para cancelar el proceso podemos pulsar el botón volver en cualquier momento.

Ilustración 73 Botón modificar

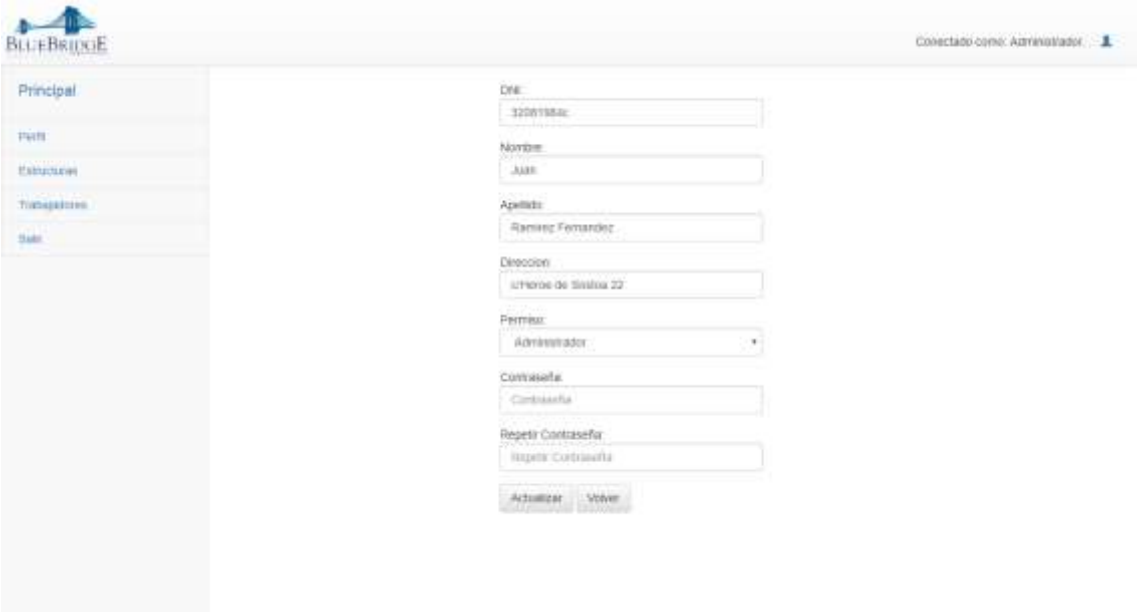

Ilustración 74 Modificar trabajador

6.2.10 Asignar estructuras a un trabajador

Para asignar estructuras a un trabajador, debemos estar en la pantalla de ver un trabajador. En el aparecerá una tabla con las estructuras asociadas. Si no aparece ninguna es que no tiene asignadas estructuras.

Clicaremos el botón “+”  para acceder a una pantalla con las estructuras no asignadas a ningún trabajador. Clicaremos nuevamente en el botón “+” para añadir la estructura a dicho trabajador y cuando acabemos pulsaremos el botón “volver” para salir a la pantalla de ver un trabajador.

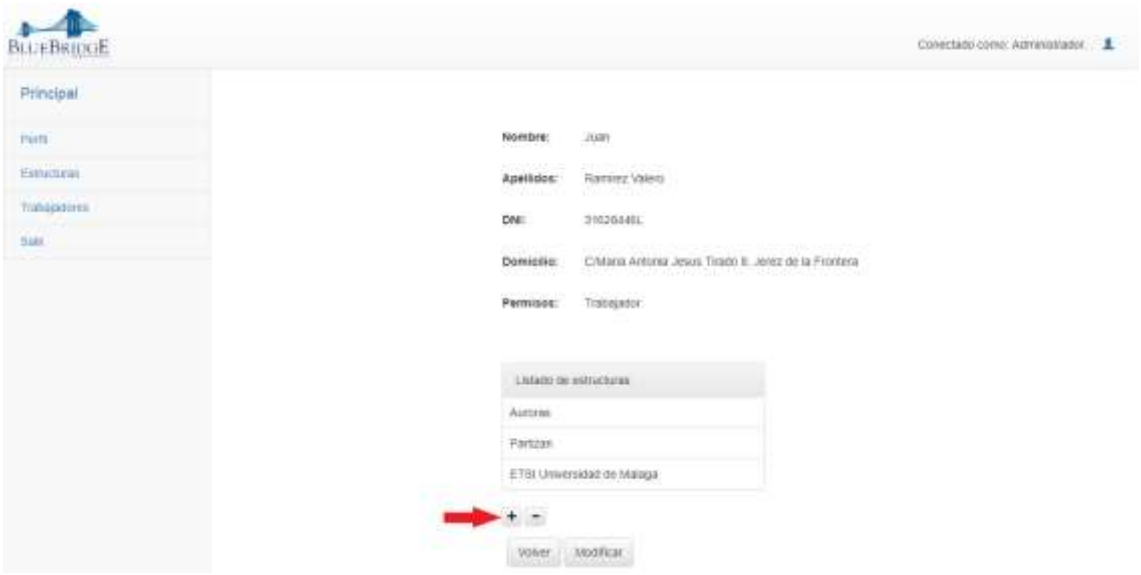

Ilustración 75 Ver trabajador

Ilustración 76 Estructuras sin asignar

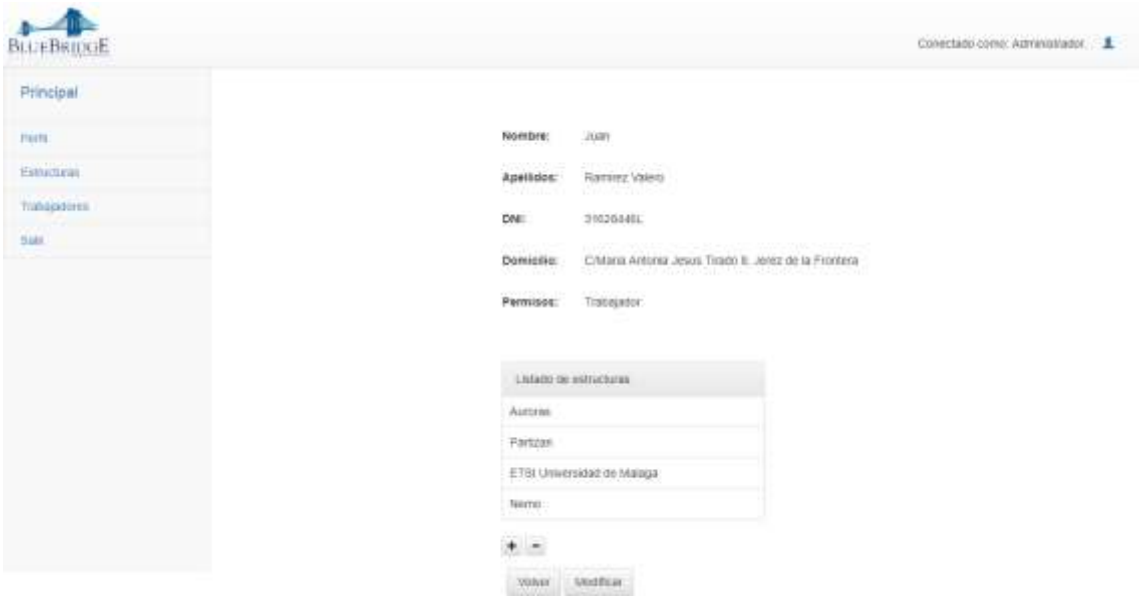

Ilustración 77 Trabajador con más estructuras asignadas

6.2.11 Desasignar estructuras a un trabajador

Para desasignar estructuras a un trabajador, debemos estar en la pantalla de ver un trabajador. En el aparecerá una tabla con las estructuras asociadas. Si no aparece ninguna es que no tiene asignadas estructuras.

Clicaremos el botón "-" para acceder a una pantalla con las estructuras asignadas a ningún trabajador. Clicaremos en el botón de papelera  para quitar la estructura a dicho trabajador y cuando acabemos pulsaremos el botón "volver" para salir a la pantalla de ver un trabajador.

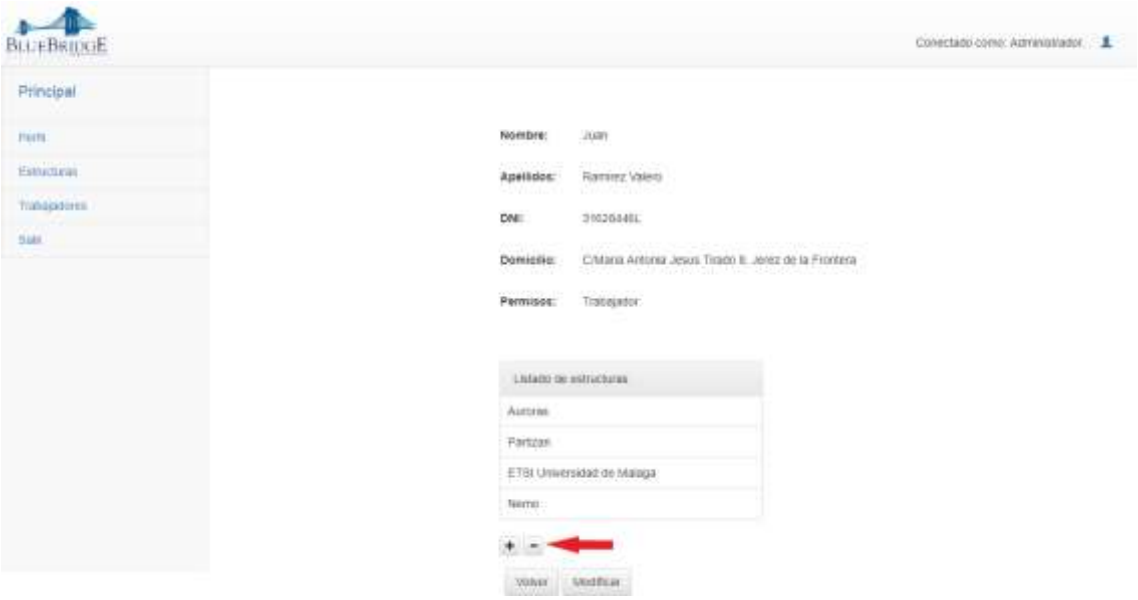

Ilustración 78 Ver trabajador

Ilustración 79 Listado de estructuras asignadas

Ilustración 80 Trabajador con menos estructuras asignadas

6.2.12 Eliminar trabajador

Para eliminar un trabajador, debemos clicar en el botón rojo de papelera que se encuentra al lado del trabajador que queremos eliminar.

Nos pedirá confirmación de la eliminación y entonces el trabajador desaparecerá del listado.

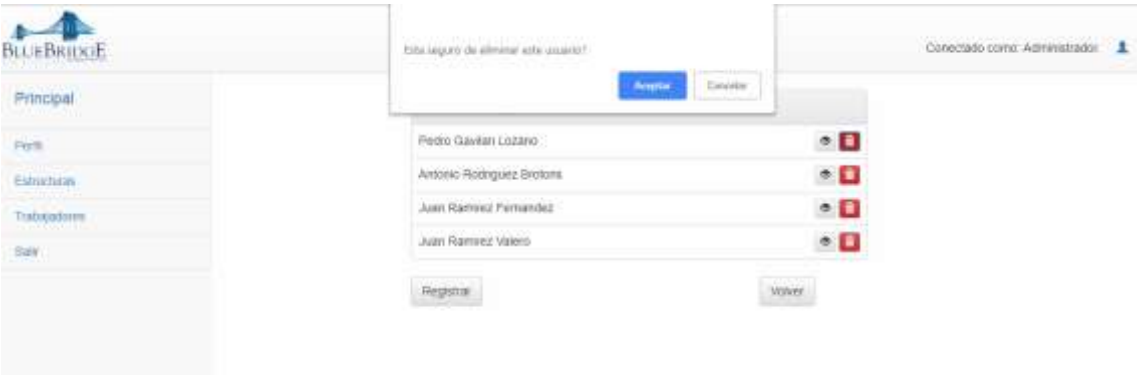

Ilustración 81 Eliminar trabajador

6.3 Guía del trabajador

Continuamos con el trabajador, como vemos en la imagen, automáticamente aparecen las estructuras asociadas a su usuario, por lo que puede acceder rápidamente a lo que necesite ver.

Ilustración 82 Inicio trabajador

También podemos observar que tiene varias opciones en el lateral izquierdo de la pantalla.

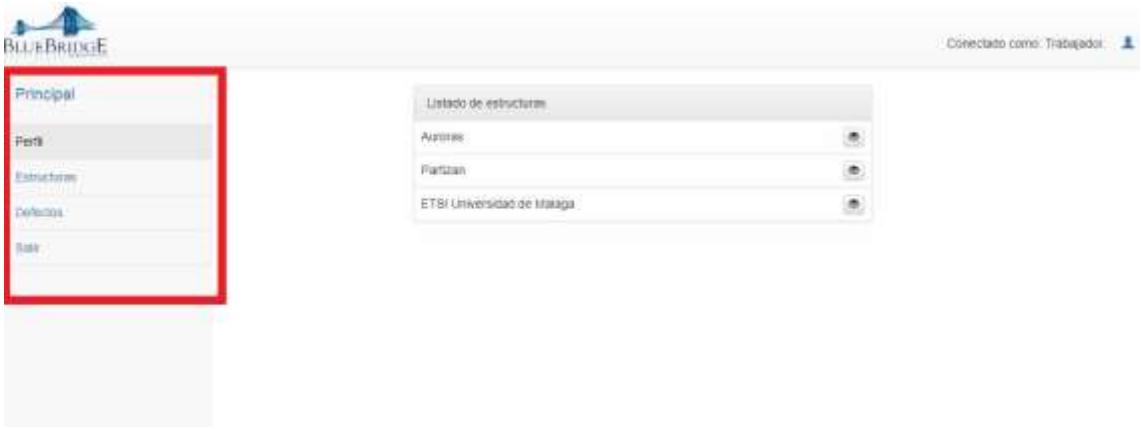

Ilustración 83 Menú lateral izquierdo trabajador

6.3.1 Perfil

El trabajador puede consultar sus datos personales en esta sección. Se accede por el control lateral de la aplicación.

Ilustración 84 Perfil trabajador

6.3.2 Estructuras

En esta sección el usuario puede ver las estructuras asociadas a su cuenta y poder acceder a ver los datos de esta.

Ilustración 85 Listado estructuras

Para ello clicamos en el icono del “ojo” al lado del nombre de la estructura. Y veremos la siguiente pantalla.

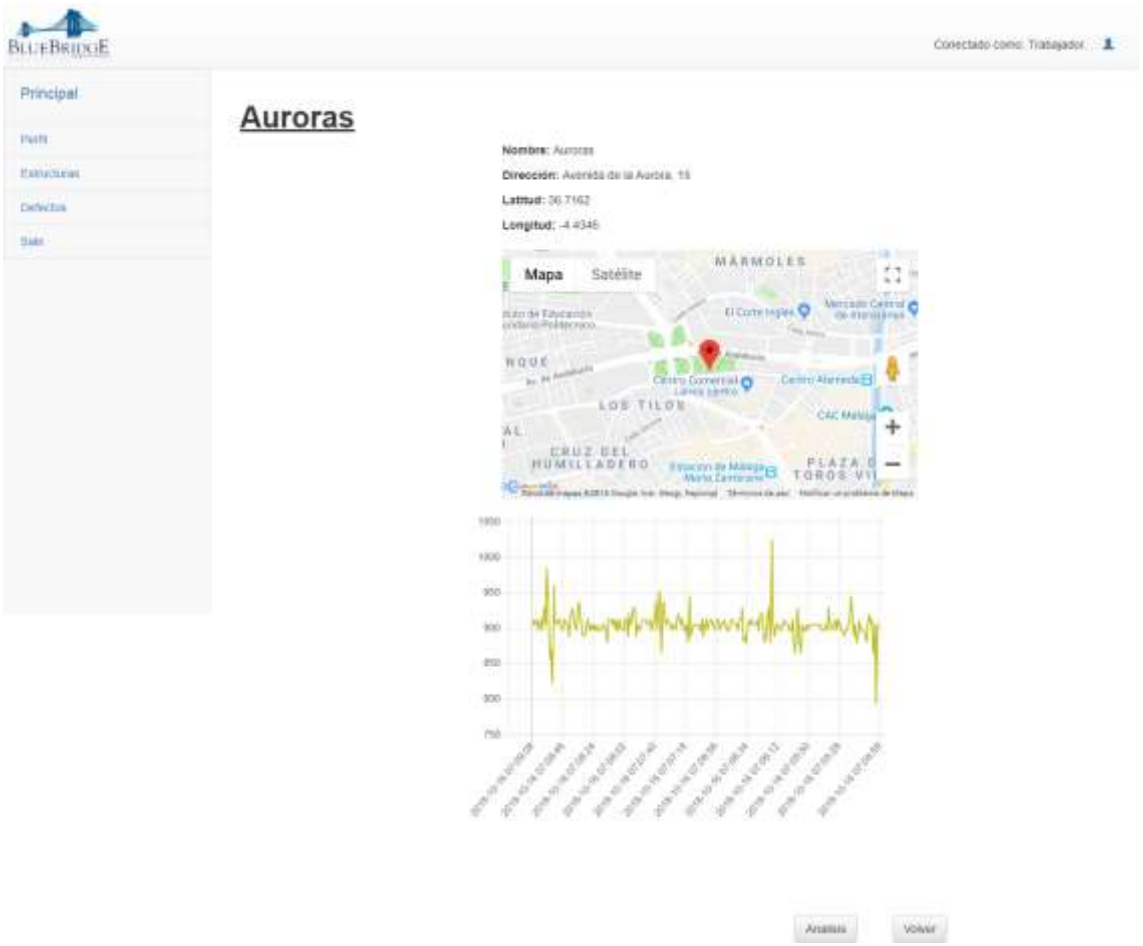

Ilustración 86 Datos de la estructura

En él se encuentran todos sus datos, la ubicación en el mapa y una gráfica con los datos recogidos el último mes.

6.3.3 Análisis

Desde la página de la estructura, podemos acceder al análisis de los datos obtenidos clicando en el botón “Análisis”

Ilustración 87 Botón análisis estructura

Accederemos a elegir el intervalo de tiempo y el tipo de defecto que queremos buscar y clicaremos en la lupa para que la aplicación busque el defecto y nos marque en la gráfica los puntos donde acurre dicho defecto.

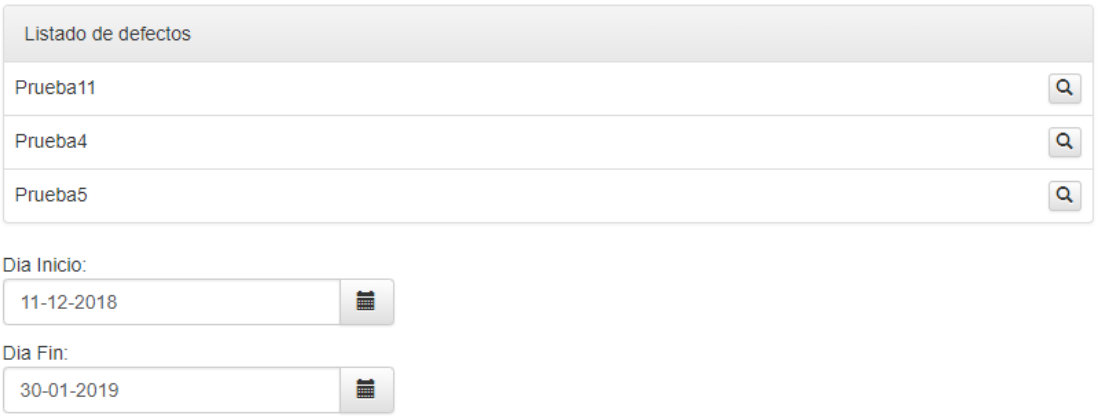

Ilustración 88 Elección de defecto

Ilustración 89 Grafica con defectos

En esta ventana también tenemos la opción de descargarnos un informe en PDF del análisis que acabamos de realizar.

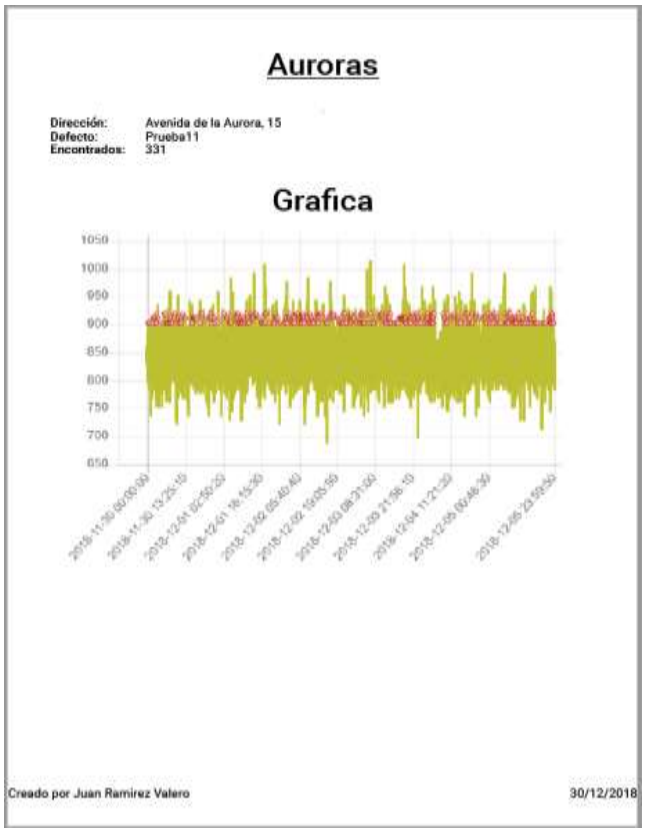

Ilustración 90 Informe generado

6.3.4 Defectos

A esta pestaña accedemos clicando en donde pone “Defectos” en el menú lateral izquierdo. Nos mostrará el listado de defectos que el usuario a creado y las opciones de modificar los datos del defecto, eliminarlo o añadir uno nuevo.

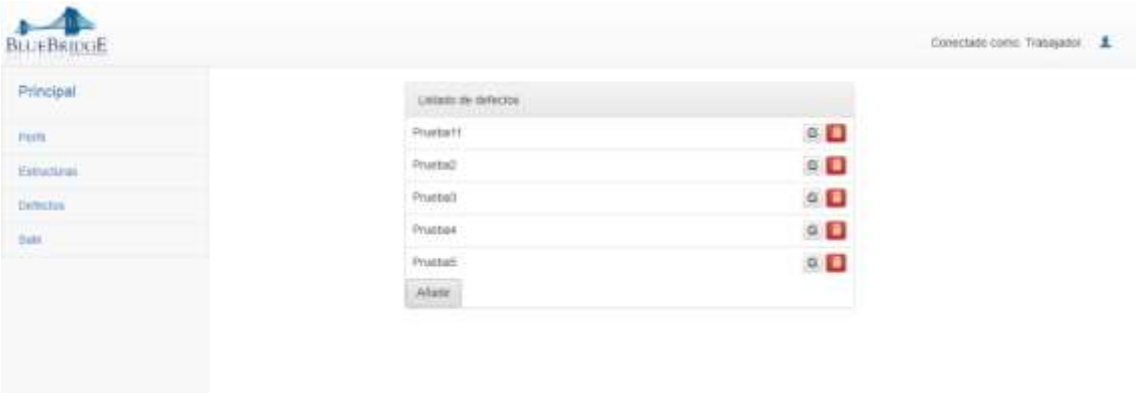

Ilustración 91 Listado de defectos

6.3.5 Añadir defecto

Para realizar este proceso, hay que clicar en el botón añadir de la ventana de Defectos. Se nos mostrará una nueva ventana con los datos a introducir.

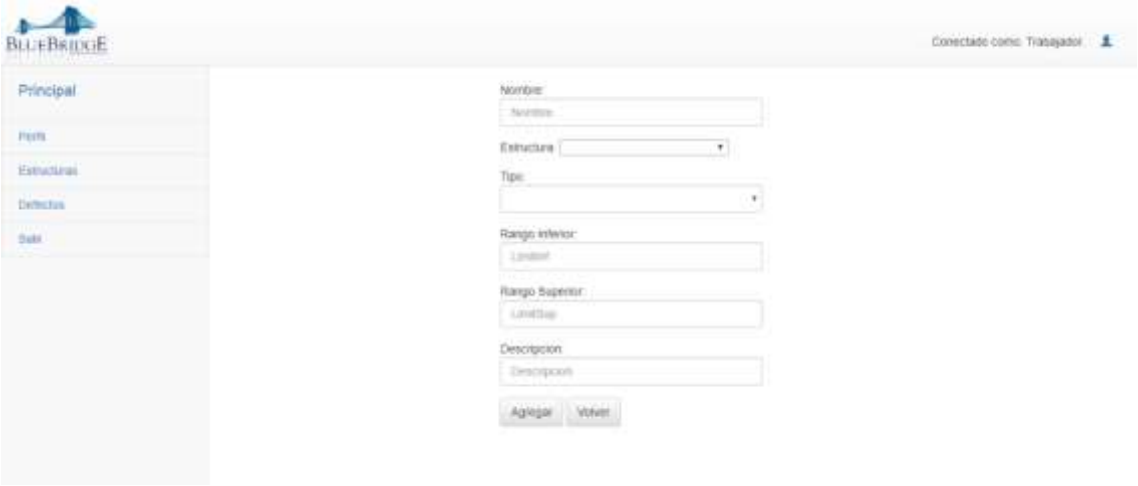

Ilustración 92 Añadir defecto

Podemos elegir la estructura a la que irá asociada y también el tipo de defecto que será a elegir “Entre”, “Por Encima” y “Por debajo”. Dependiendo de esta elección se activarán o desactivarán los campos Rango Inferior y Rango Superior.

Cuando acabemos de introducir todos los datos, clicaremos en el botón “Agregar”, nos pedirá confirmación y se añadirá el defecto a la aplicación.

También podemos cancelar el formulario clicando en el botón “Volver”.

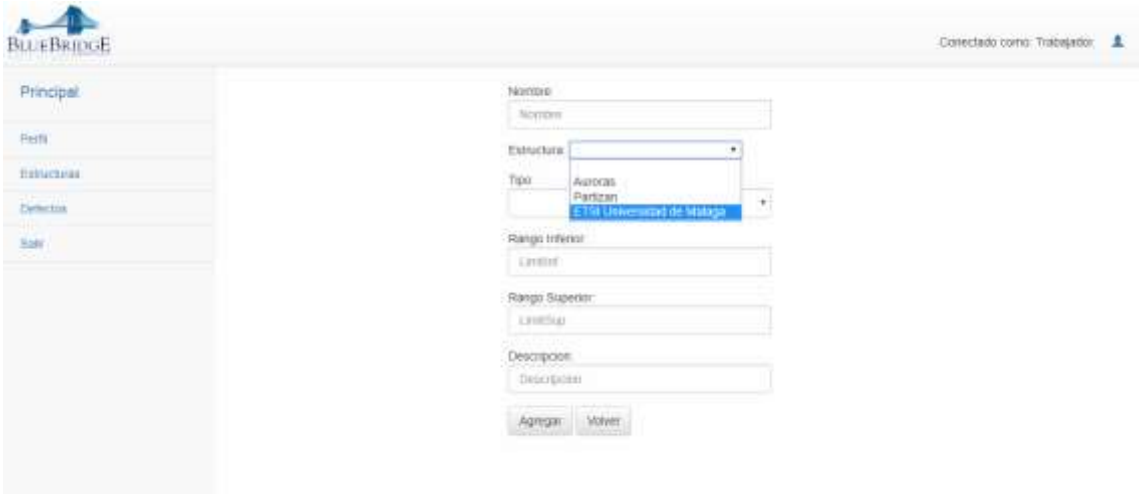

Ilustración 93 Listado estructuras defecto

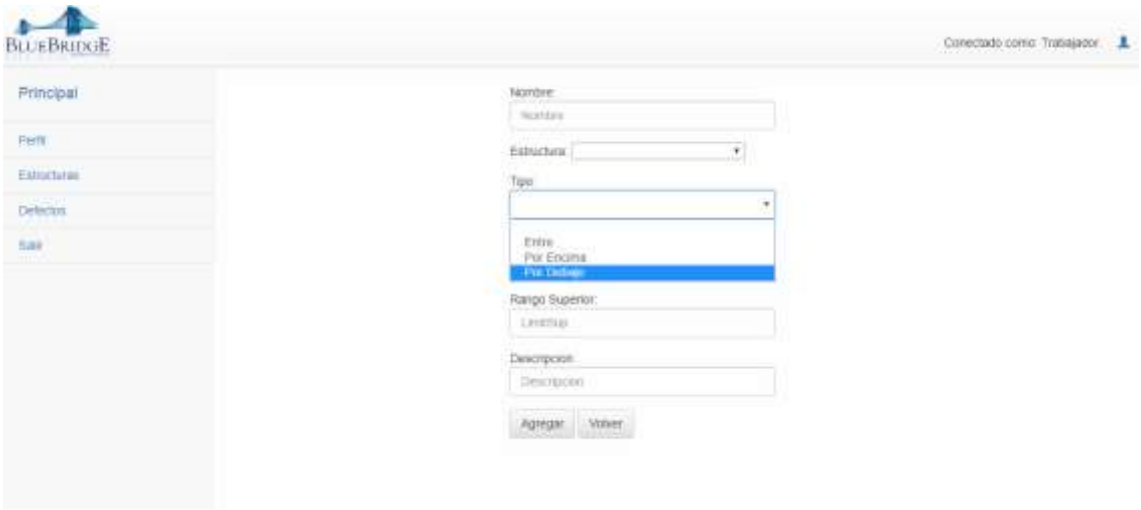

Ilustración 94 Listado tipo defecto

6.3.6 Actualizar defecto

Para actualizar el defecto debemos clicar el botón con el icono de editar que se encuentra al lado del defecto a cambiar. Se abrirá una nueva página con los datos actuales del defecto. Cambiaremos los datos que queramos y clicaremos el botón “Actualizar”, nos pedirá confirmar los cambios y se actualizará el defecto.

Como en añadir defecto, podemos cancelar el proceso pulsando el botón “Volver” en cualquier momento.

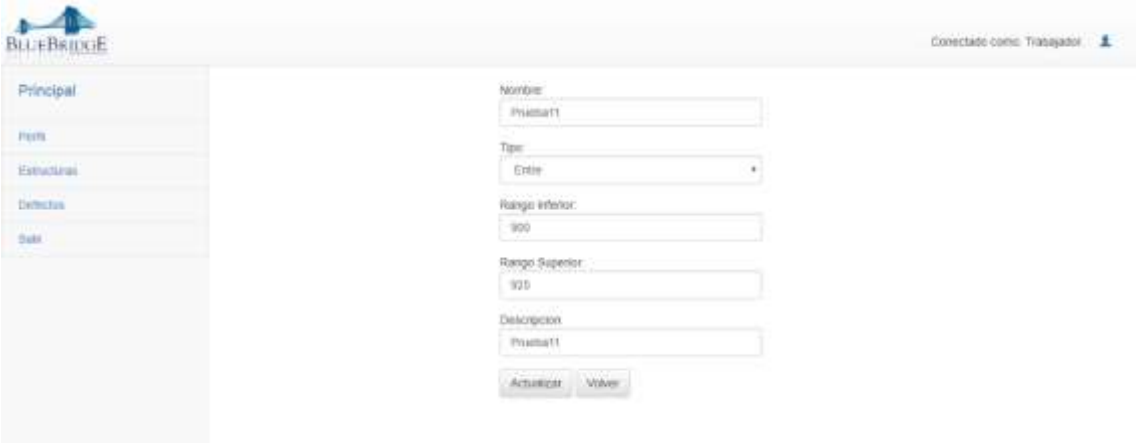

Ilustración 95 Actualizar defecto

6.3.7 Eliminar defecto

Para eliminar un defecto, debemos clicar en el botón rojo de papelera que se encuentra al lado del defecto a eliminar.

Nos pedirá confirmación de la eliminación y entonces el defecto desaparecerá del listado.

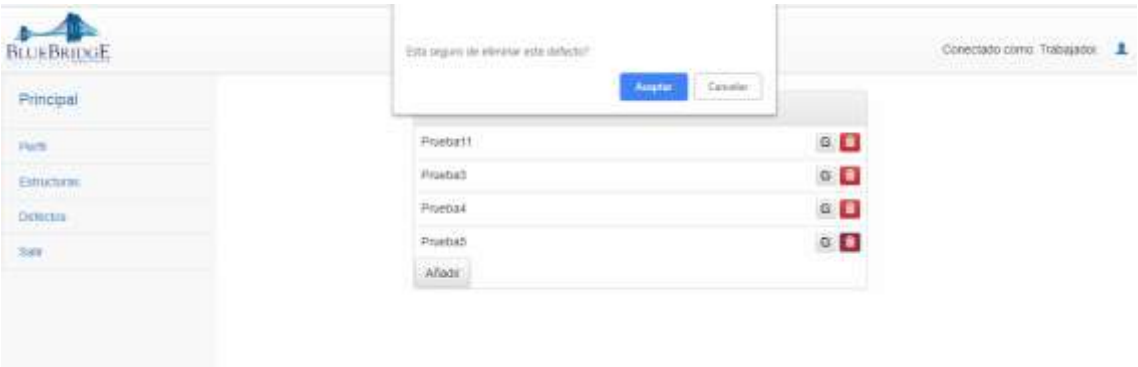

Ilustración 96 Eliminar defecto

7. Pruebas

En este apartado, se han realizado 11 pruebas, en las que se aportarán evidencias de la interfaz de la aplicación y de la base de datos.

Las primeras pruebas verificaremos que solamente los usuarios registrados puedan loguearse en la aplicación.

Las demás pruebas tienen como objetivo comprobar que el resultado es el esperado.

N.º de Prueba	Nombre de la prueba	Resultado esperado
1	Login con usuario correcto y contraseña incorrecta	El sistema no permite el acceso.

idUsuario	DNI	Nombre	Apellido	Password
4	30536974G	Antonio	Rodriguez Brotons	$2y$10$4ocmMGKMDfq9GkVdstOKMOfVk4YSb/8htnLQxQ4Etlb...
1	30636974G	Pedro	Gavilan Lozano	$2y$10$wbeHGmecNCyhY2AQ2EWGSuEbXNJU6Wm4TeXWW6vaf5...
12	31628446L	Juan	Ramirez Valero	$2y$10$UZPAnxgK/YDZccc1g01DX.q6yTmJ/p0BvWP/ptF.xq/...
11	32081984c	Juan	Ramirez Fernandez	$2y$10$mC1/dkR0B4udKJ/i8PthreR/x0yvK7HWGzBGBmQVTBE

Ilustración 97 BD usuario

Ilustración 98 Datos de usuario introducidos

Pulsamos el botón ingresar:

Ilustración 99 Error inicio sesión

N.º de Prueba	Nombre de la prueba	Resultado esperado
2	Login con usuario incorrecto	El sistema no permite el acceso.

idUsuario	DNI	Nombre	Apellido	Password
4	30536974G	Antonio	Rodriguez Brotons	$2y$10$4ocmMGKMDfq9GkVdstOKMOfVk4YSb/8htnLQxQ4Etlb...
1	30636974G	Pedro	Gavilan Lozano	$2y$10$wbeHGmecNCyhY2AQ2EWGSuEbXNJU6Wn4TeXWW6vaf5...
12	31628446L	Juan	Ramirez Valero	$2y$10$UZPAnxgK/YDZccc1g01DX.q6yTmJ/p0BvWP/ptF.xq/...
11	32081984c	Juan	Ramirez Fernandez	$2y$10$mC1/dkR0B4udKJ/i8PthreR/x0yvK7HWGzBGBmQVTBE

Ilustración 100 BD usuario

Ilustración 101 Datos de usuario introducidos

Pulsamos el botón ingresar:

Ilustración 102 Error inicio sesión

N.º de Prueba	Nombre de la prueba	Resultado esperado
3	Añadir un defecto	Se agrega el defecto a la base de datos

Ilustración 103 Datos defecto

Al pulsar en el botón “Agregar”, comprobaremos que se guarda en la base de datos y aparece en el listado de defectos del usuario.

idDefecto	Nombre	TipoDefecto	LimitSup	LimitInf	Descripcion	Usuario	Estructura
16	Prueba11	Entre	925	900	Prueba11	12	1
24	Brecha	Por Encima	980	*NULL*	Limite vibración	12	7
18	Prueba3	Por Debajo	*NULL*	850	Prueba3	12	7
19	Prueba4	Por Encima	1000	*NULL*	Prueba4	12	1
23	Prueba5	Por Debajo	985	920	Prueba5	12	1

Ilustración 104 BD con datos agregados

Ilustración 105 Listado de defectos

N.º de Prueba	Nombre de la prueba	Resultado esperado
4	Eliminar un defecto	El defecto es eliminado de la base de datos

Ilustración 106 Listado antes de eliminar

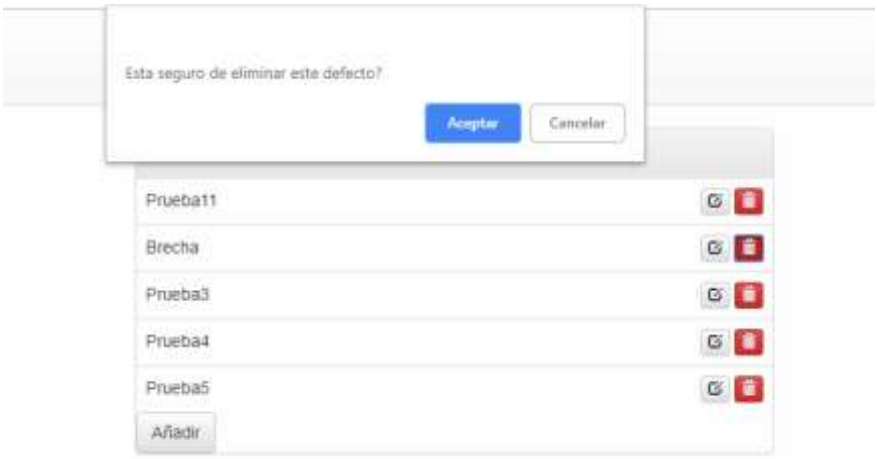

Ilustración 107 Confirmación de eliminar

Tras pulsar en “eliminar” y aceptar la confirmación, vemos que ha desaparecido de la base de datos y del listado.

idDefecto	Nombre	TipoDefecto	LimitSup	LimitInf	Descripcion	Usuario	Estructura
16	Prueba11	Entre	925	900	Prueba11	12	1
18	Prueba3	Por Debajo	*NULL*	850	Prueba3	12	7
19	Prueba4	Por Encima	1000	*NULL*	Prueba4	12	1
23	Prueba5	Por Debajo	985	920	Prueba5	12	1

Ilustración 108 Base de datos después de eliminar

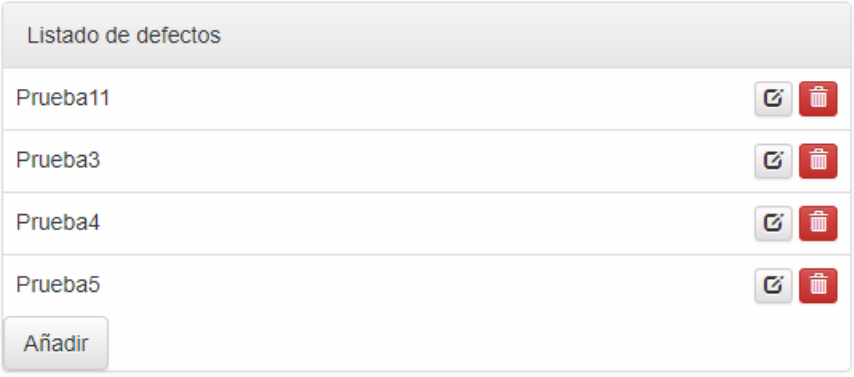

Ilustración 109 Listado des pués de eliminar

N.º de Prueba	Nombre de la prueba	Resultado esperado
5	Modificar un defecto	El defecto es modificado de la base de datos

idDefecto	Nombre	TipoDefecto	LimitSup	LimitInf	Descripcion	Usuario	Estructura
16	Prueba11	Entre	925	900	Prueba11	12	1
18	Prueba3	Por Debajo	*NULL*	850	Prueba3	12	7
19	Prueba4	Por Encima	1000	*NULL*	Prueba4	12	1
23	Prueba5	Por Debajo	985	920	Prueba5	12	1

Ilustración 110 Base de datos antes de modificar

Ilustración 111 Defecto antes de modificar

Ilustración 112 Defecto modificado

Al pulsar el botón “Actualizar” cambiaran los datos modificados en la base de datos.

idDefecto	Nombre	TipoDefecto	LimitSup	LimitInf	Descripcion	Usuario	Estructura
16	Prueba11	Entre	925	900	Prueba11	12	1
18	Prueba3	Entre	1000	700	Normalización	12	7
19	Prueba4	Por Encima	1000	*NULL*	Prueba4	12	1
23	Prueba5	Por Debajo	985	920	Prueba5	12	1

Ilustración 113 Base de datos después de modificar

N.º de Prueba	Nombre de la prueba	Resultado esperado
6	Añadir un trabajador	Se agrega el trabajador a la base de datos

idUsuario	DNI	Nombre	Apellido	Password	Domicilio	Permiso
4	30536974G	Antonio	Rodriguez Brotons	$2y$10$4ocmMGKMDfq9GkVdxtOKMOfVk4YSb/8htmLQxQ4Etlb	c/Heroes de Sostoa, nº2, 3ºA	1
1	30536974G	Pedro	Gavilan Lozano	$2y$10$xbeHGmecNCyhY2AQ2EWGSuEbXNJU6Wx4TaXiWW6vaf5	c/Eguiluz, nº2, 3ºA	0
12	31628446L	Juan	Ramirez Valero	$2y$10$UZPAnxgK/YDZccc1g01DX.q6yTmJp0BvWPptF.xq/	C/Maria Antonia Jesus Tirado 8, Jerez de la Fronte...	0
11	32081984c	Juan	Ramirez Fernandez	$2y$10$mCIdkR0B4udKJI8PfmeRvi0yyK7hWGzBGBmQVTBE	c/Heroe de Sostoa 22	1

Ilustración 114 Base de datos antes de añadir trabajador

Ilustración 115 Datos del trabajador introducido

Ilustración 116 Listado con el trabajador añadido

Al pulsar en el botón “Agregar”, comprobaremos que se guarda en la base de datos y aparece en el listado de defectos del usuario.

idUsuario	DNI	Nombre	Apellido	Password	Domicilio	Permiso
4	30536974G	Antonio	Rodriguez Brotons	$2y$10$4ocmMGKMDfq9GkVdxtOKMOfVk4YSb/8htmLQxQ4Etlb	c/Heroes de Sostoa, nº2, 3ºA	1
1	30536974G	Pedro	Gavilan Lozano	$2y$10$xbeHGmecNCyhY2AQ2EWGSuEbXNJU6Wx4TaXiWW6vaf5	c/Eguiluz, nº2, 3ºA	0
12	31628446L	Juan	Ramirez Valero	$2y$10$UZPAnxgK/YDZccc1g01DX.q6yTmJp0BvWPptF.xq/	C/Maria Antonia Jesus Tirado 8, Jerez de la Fronte	0
72	32081984B	Unai	Urkixo	$2y$10$G2pOochXWAXTQQgCX.xNx.NK3CXBM44MuYWM1F4gMZ.3	Calle Logroño nº 3 -1ºB, Donostia	0
11	32081984c	Juan	Ramirez Fernandez	$2y$10$mCIdkR0B4udKJI8PfmeRvi0yyK7hWGzBGBmQVTBE	c/Heroe de Sostoa 22	1

Ilustración 117 Base de datos después de añadir al trabajador

N.º de Prueba	Nombre de la prueba	Resultado esperado
7	Eliminar un trabajador	El trabajador es eliminado de la base de datos

Ilustración 118 Base de datos antes de eliminar un trabajador

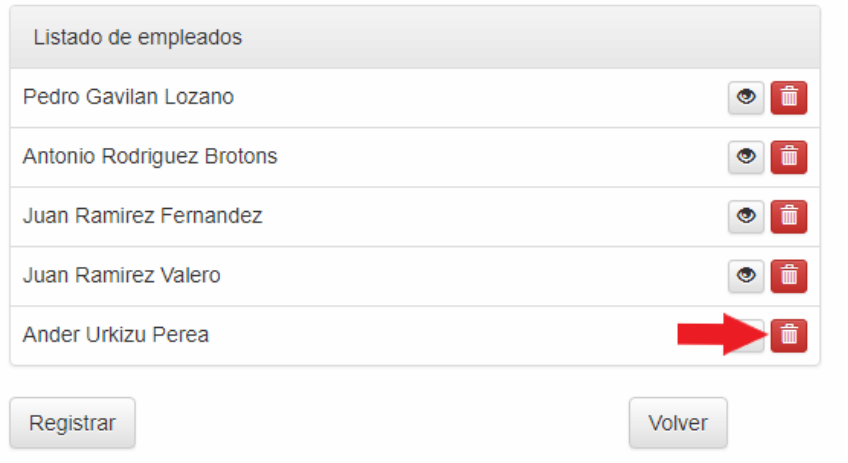

Ilustración 119 Listado de trabajadores

Tras pulsar en “eliminar” y aceptar la confirmación, vemos que ha desaparecido de la base de datos y del listado.

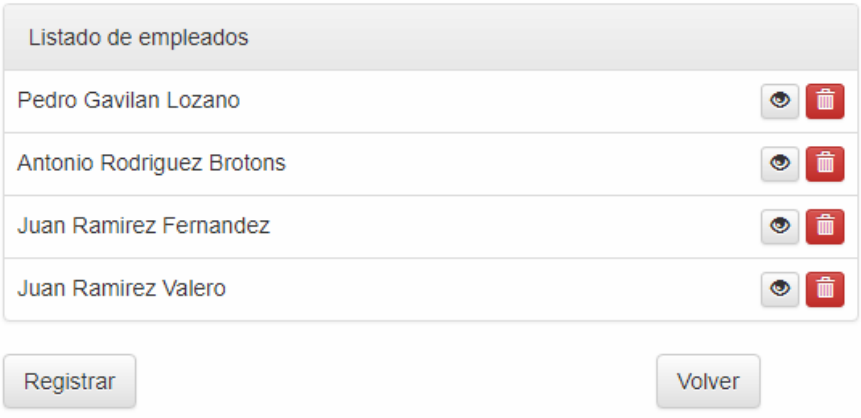

Ilustración 120 Listado después de eliminar un trabajador

Ilustración 121 Base de datos después de eliminar un trabajador

N.º de Prueba	Nombre de la prueba	Resultado esperado
8	Modificar un trabajador	El trabajador es modificado de la base de datos

Ilustración 122 Trabajador sin modificar

Ilustración 123 Trabajador con los datos cambiados

Al pulsar el botón “Actualizar” cambiaran los datos modificados en la base de datos.

Ilustración 124 Base de datos después de modificar

N.º de Prueba	Nombre de la prueba	Resultado esperado
9	Añadir una estructura	Se agrega la estructura a la base de datos

idEstructura	Nombre	Direccion	Latitud	Longitud	Usuario
1	Auroras	Avenida de la Aurora, 15	36.7162	-4.4346	12
6	Partizan	Avenida de Partizan, 15	30.7163	-5.43434	12
7	ETSI Universidad de Malaga	Bulevar Luis Pasteur	36.7159	-4.47731	12
8	Nemo	C/Wallaby, 42 ,Sidney	35.24	-6.25	*NULL*

Ilustración 125 Base de datos antes de añadir una estructura

Ilustración 126 Datos introducidos

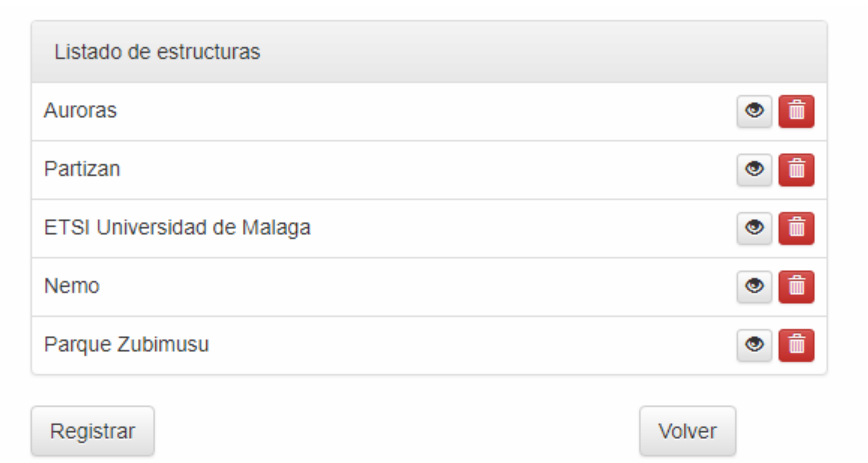

Ilustración 127 Listado de estructuras con la nueva estructura

idEstructura	Nombre	Direccion	Latitud	Longitud	Usuario
1	Auroras	Avenida de la Aurora, 15	36.7162	-4.4346	12
6	Partizan	Avenida de Partizan, 15	30.7163	-5.43434	12
7	ETSI Universidad de Malaga	Bulevar Luis Pasteur	36.7159	-4.47731	12
8	Nemo	C/Wallaby, 42 ,Sidney	35.24	-6.25	*NULL*
15	Parque Zubimusu	Calle Vitoria-Gasteiz	43.3138	-2.00663	*NULL*

Ilustración 128 Base de datos con la estructura nueva

N.º de Prueba	Nombre de la prueba	Resultado esperado
10	Eliminar una estructura	La estructura es eliminada de la base de datos

idEstructura	Nombre	Direccion	Latitud	Longitud	Usuario
1	Auroras	Avenida de la Aurora, 15	36.7162	-4.4346	12
6	Partizan	Avenida de Partizan, 15	30.7163	-5.43434	12
7	ETSI Universidad de Malaga	Bulevar Luis Pasteur	36.7159	-4.47731	12
8	Nemo	C/Wallaby, 42 ,Sidney	35.24	-6.25	*NULL*
15	Parque Zubimusu	Calle Vitoria-Gasteiz, nº 5	43.31	-2.006	*NULL*

Ilustración 129 Base de datos antes de eliminar estructura

Tras pulsar en “eliminar” y aceptar la confirmación, vemos que ha desaparecido de la base de datos y del listado.

Ilustración 130 Listado después de eliminar estructura

idEstructura	Nombre	Direccion	Latitud	Longitud	Usuario
1	Auroras	Avenida de la Aurora, 15	36.7162	-4.4346	12
6	Partizan	Avenida de Partizan, 15	30.7163	-5.43434	12
7	ETSI Universidad de Malaga	Bulevar Luis Pasteur	36.7159	-4.47731	12
8	Nemo	C/Wallaby, 42 ,Sidney	35.24	-6.25	*NULL*

Ilustración 131 Base de datos después de eliminar estructura

N.º de Prueba	Nombre de la prueba	Resultado esperado
11	Modificar una estructura	La estructura es modificada de la base de datos

idEstructura	Nombre	Direccion	Latitud	Longitud	Usuario
1	Auroras	Avenida de la Aurora, 15	36.7162	-4.4346	12
6	Partizan	Avenida de Partizan, 15	30.7163	-5.43434	12
7	ETSI Universidad de Malaga	Bulevar Luis Pasteur	36.7159	-4.47731	12
8	Nemo	C/Wallaby, 42 ,Sidney	35.24	-6.25	NULL
15	Parque Zubimusu	Calle Vitoria-Gasteiz	43.3138	-2.00663	NULL

Ilustración 132 Base de datos antes de eliminar estructura

Ilustración 133 Datos de la estructura

Ilustración 134 Datos de la estructura modificado

Al pulsar el botón “Actualizar” cambiaran los datos modificados en la base de datos.

idEstructura	Nombre	Direccion	Latitud	Longitud	Usuario
1	Auroras	Avenida de la Aurora, 15	36.7162	-4.4346	12
6	Partizan	Avenida de Partizan, 15	30.7163	-5.43434	12
7	ETSI Universidad de Malaga	Bulevar Luis Pasteur	36.7159	-4.47731	12
8	Nemo	C/Wallaby, 42 ,Sidney	35.24	-6.25	NULL
15	Parque Zubimusu	Calle Vitoria-Gasteiz, nº 5	43.31	-2.006	NULL

Ilustración 135 Base de datos después de modificar estructura

8. Conclusiones

Se ha realizado una aplicación web que pueda sustituir a los típicos softwares de escritorio. Esta trata la monitorización de datos y el apoyo a la decisión de estructuras como pavimentos o puentes.

La aplicación ha sido realizada con Angular JS para el controlador y la vista (apoyado con HTML y CSS). Se ha realizado una api en PHP que hace de comunicador entre la base de datos y el controlador.

La aplicación nos mostrará, dependiendo de los permisos del usuario, unas vistas específicas para administrador o trabajador. Por tanto, el usuario "Administrador" se encargará de toda la gestión de creación, modificación y eliminación de trabajadores, de gestionar las estructuras de la que se obtienen los datos y también de asociar estas estructuras a los trabajadores.

El usuario "Trabajador" solo tendrá que encargarse de ver que estructuras que tiene, pudiendo realizar un análisis acotando por fecha y los niveles de vibración a analizar. Luego con el resultado podrá decidir qué medidas tomar con la estructura y generar un informe en formato pdf.

He podido aprender Angular JS, un Framework de JavaScript que desconocía. Recomiendo dicho lenguaje porque es un lenguaje amigable y evita tener que realizar diferentes vistas según el dispositivo utilizado.

Respecto al lenguaje elegido para el modelo (php), para modelos mucho más ambiciosos, sería recomendable cambiarlo por otro lenguaje como por ejemplo Node JS o C# ya que están mejor optimizado para una cantidad mayor de datos.

Se podría mejorar el algoritmo de búsqueda de análisis, por uno más eficiente, a más cantidad de datos a analizar, mayor es el tiempo que tarda en mostrar un resultado.

Para completar el análisis se podrían realizar otras pruebas de rendimiento y tiempos de ejecución, pero por falta de tiempo (ya que estoy compaginándolo con un trabajo a tiempo completo) no se han podido realizar.

Aun así, esta aplicación proporciona un análisis y monitorización de vibraciones sobre pavimentos suficiente para poder prever daños en pavimentos. Por tanto, al anticiparnos a que se degrade la estructura reduciremos costes en reparaciones, aumentando la vida de la estructura.

Para una empresa pequeña, este desarrollo es suficiente para cubrir las necesidades básicas de monitorización y prevención. Pero si se pretende un desarrollo mucho más ambicioso con un mayor número de datos y monitorización automática, habría que mejorar el modelo y tener un equipo de desarrollo dedicado exclusivamente al proyecto.

9. Bibliografía

1. Manual PHP en español: http://secure.php.net/manual/es/
2. MySQL : edición especial / Paul Dubois.
3. Web para descargar el editor PhpStorm: https://www.jetbrains.com/phpstorm/
4. Página oficial de Angular JS: https://angularjs.org/
5. API de Angular JS: http://docs.angularjs.org/api
6. UML y patrones : una introducción al análisis y diseño orientado a objetos y al proceso unificado / Craig Larman.