UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA POLITECNICA SUPERIOR DE GANDIA

Grado en Ing. Sist. de Telecom., Sonido e Imagen

"Desarrollo de una aplicación web para la planificación temporal de proyectos mediante diagramas"

TRABAJO FINAL DE GRADO

Autor/a:

Diego Olmos Moreno

Tutor/a:

Jordi Bataller Mascarell

GANDIA, 2018

Agradecimientos

Jordi Bataller Mascarell, León Troitski, Ricardo Martínez Baeza, Jimena Acosta Argueta, Raúl Ripollés García, Alvaro Muñoz Caballero, Vicenta Moreno Algaba, Juan José Olmos Sanchis, Andrea Olmos Moreno.

RESUMEN

Actualmente el uso de aplicaciones web está completamente integrado en todos los ámbitos de la sociedad, y por supuesto en el laboral. La mayoría de grandes y medianas empresas utilizan estas aplicaciones para la gestión de información relacionada con todas las áreas de su estructura organizativa. Entre las múltiples herramientas existentes para la gestión y organización de proyectos se encuentra el conocido como Diagrama de Roy, que consiste en la división de un proyecto en actividades, con relaciones entre ellas, duraciones y tiempos calculados. Este proyecto pretende desarrollar una aplicación para poder gestionar proyectos mediante este diagrama on-line. La estructura del aplicativo se ha diseñado mediante la arquitectura Cliente / Servidor, pues esta organización posibilita que las partes del sistema, equipos o aplicaciones, se especialicen en tareas diferentes y favorece la robustez del sistema. El modelo de organización elegido es el Modelo-Vista-Controlador que separa y ordena la lógica de la aplicación por su cometido. El lenguaje utilizado es fundamentalmente Javascript, que es el lenguaje más extendido en este tipo de aplicaciones. Por último, utilizaremos una base de datos SQL para el almacenamiento de la información con la que operemos.

ABSTRACT

Nowadays, the use of web applications is completely integrated in all sectors of society, and, clearly, at the workplace. Most of large and medium sized companies use these applications in order to handle information related to all areas of their organizational structure. Amongst the multiple available tools to manage projects we can find one known as Roy's diagram which consist in dividing a project into activities related among themselves and with controlled timing. This project aims to develop a web application allowing the management of a project using Roy diagram on-line. The structure of the applicative has been designed following the client/server architecture. This approach allows different parts of a system, equipment or application to specialize in different tasks, thus enhancing the system strength. The chosen organizational model is Model - View – Controller. The language used for coding is mainly JavaScript, very commonly used in this type of applications. We will also use a SQL database for storing the data we work with.

PALABRAS CLAVE / KEYWORDS

Aplicación Web, Cliente / Servidor, Modelo-Vista-Controlador, Base de datos, Diagrama de Roy.

Web application, Client / Server architecture, Model – View - Controller, Database, Roy's Diagram.

Índice

Capítulo 1
Introducción

Ante el vertiginoso auge de las tecnologías digitales de los últimos años han ido apareciendo paralelamente nuevos problemas y necesidades, Internet ha transformado completamente nuestra capacidad de gestión del tiempo, nuestra forma de trabajo y nuestras posibilidades. Hoy en día es común el trabajo en equipo desde diferentes localizaciones, la extrema necesidad de maximizar el tiempo y la optimización de las herramientas necesarias para la consecución de dichos objetivos.

La mayoría de las grandes empresas cuentan con herramientas eficaces para la organización de trabajos colaborativos divididos en tareas adjudicadas a diferentes personas o equipos mediante plataformas digitales en Internet.

El propósito principal de este proyecto ha sido más formativo que estrictamente funcional, consiste en una sencilla aplicación web que implementa uno de los tantos sistemas existentes con el propósito de la organización de proyectos, como es el diagrama de Roy. No obstante, implementa los principales propósitos de este: las relaciones entre las actividades, y los cálculos del Tiempo Mínimo y Máximo de cada actividad, que representan lo más pronto y más tarde que puede llegarse a cada actividad respectivamente, así como la duración total del proyecto.

Por tanto, para la consecución de los objetivos descritos hemos estudiado en primer lugar el funcionamiento del Diagrama de Roy, posteriormente hemos investigado sobre la disponibilidad de tecnologías apropiadas para la implementación de aplicaciones web y por último hemos desarrollado la aplicación teniendo que ir adaptándonos en ocasiones a otras herramientas diferentes de las previstas en un primer momento.

El desarrollo de la siguiente Memoria está organizado de la siguiente forma: En el capítulo 2 se expone el análisis de requerimientos, un esbozo sobre las funcionalidades principales, una breve introducción teórica del funcionamiento del diagrama para una mejor comprensión por parte del lector y los objetivos finales de la aplicación. En el capítulo 3 describiremos de forma esquemática la arquitectura de la aplicación, las funcionalidades de las partes, la organización de los ficheros y las relaciones entre ellos. En el capítulo 4 hablaremos sobre las tecnologías elegidas para la construcción del aplicativo, la estructura de archivos y directorios y se analizarán los principales problemas a los que nos hemos enfrentado, así como las soluciones por las que hemos optado para solventarlos. Finalmente, en el capítulo 5, ofrecemos una guía de instalación pensada para un perfil técnico con detalles de todas las herramientas utilizadas y un Manual de Usuario pensado para un público no especializado en el que se le acompaña a través de la interfaz del aplicativo y se le explica el funcionamiento.

Capítulo 2
Análisis de requerimientos

En el presente capítulo presentamos el planteamiento inicial o boceto de una aplicación Web que nos permita la construcción de diagramas de Roy, para ello haremos una breve introducción teórica del funcionamiento del diagrama y esbozaremos los diferentes sitios de la interfaz que hemos considerado oportunos para la inserción de los datos necesarios y su posterior visualización. Hemos querido incorporar las principales operaciones que podemos realizar mediante la comunicación con bases de datos, esto es, inserción, lectura, operaciones con datos, actualización y borrado de registros.

Presentaremos la manera que hemos considerado más oportuna para implementar el diagrama a través de la aplicación. Podremos crear diferentes proyectos que contendrán actividades con información que el usuario introducirá mediante formularios. El aplicativo se encargará de realizar las consultas y operaciones pertinentes y mostrará los datos por pantalla.

Se describirán cada uno de los apartados y las operaciones a realizar en cada uno de ellos.

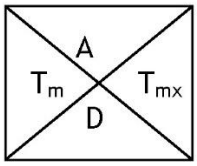
2.1. Introducción al Diagrama de Roy

2.1.1. Concepto

El diagrama de Roy es un método de organización y análisis de proyectos. Consiste en la división de un proyecto en actividades y el establecimiento de relaciones de precedencia entre ellas con el objetivo de la obtención de resultados a lo largo de un periodo de tiempo dependiente de las duraciones estimadas de las actividades y de sus relaciones.

Los elementos básicos de un Diagrama de Roy son los Nudos, que se representan mediante cuadrados, y las Flechas, que representan relaciones entre actividades.

Los cuadrados contienen información concerniente a la actividad que representan, como son el nombre de la actividad, la duración, el tiempo mínimo y el tiempo máximo.



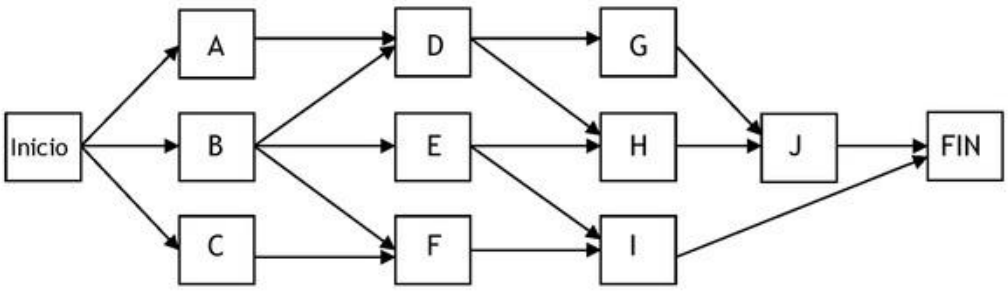
2.1.2. Construcción del Diagrama (Actividades Inicio y Fin)

Para la construcción del grafo de Roy se añaden dos actividades que posibilitarán los cálculos de los Tiempos Mínimo y Máximo, Actividad Inicio y Actividad Fin.

La Actividad Inicio precede a todas aquellas actividades que no tienen actividades precedentes, es decir, precede a todas las actividades que representan el punto de partida del proyecto. Su duración es 0 y su tiempo mínimo se establece también en 0.

La Actividad Fin está representada por un nudo al que llegan aquellas actividades que no preceden a ninguna otra. Su duración también es 0 y su tiempo mínimo tiene el mismo valor que su tiempo máximo.

De esta manera todas las actividades, a excepción de Inicio, tendrán actividades precedentes.



2.1.3. Cálculo de Tiempos Mínimo y Máximo

La duración total del proyecto es el tiempo que transcurre desde que comienza la primera actividad hasta la última. Para calcular la duración total del proyecto necesitamos definir dos conceptos fundamentales que ya se han mencionado previamente, el Tiempo Mínimo y el Tiempo Máximo de cada actividad.

Tiempo Mínimo

El Tiempo Mínimo de una actividad representa lo más pronto que se puede llegar a esa actividad. Se calcula de izquierda a derecha del diagrama eligiendo el mayor valor del binomio $D_i + T_{min(i)}$ (Duración más Tiempo Mínimo) de cada una de sus actividades precedentes, es decir, de las actividades que confluyen en ella.

$$Tiempo\ Mínimo = Max\ \Sigma\ D_i + T_{min(i)}$$

Tiempo Máximo

El $T_{max(i)}$ de una actividad representa lo más tarde que se puede llegar a esa actividad. Se calcula al contrario que el T_{min}, es decir, de derecha a izquierda, comenzando por la Actividad Fin. Elegimos el menor de los binomios $T_{max(i)}$ - D (Tiempo Máximo de su actividad siguiente menos la duración de la propia actividad sobre la que se está calculando el tiempo máximo).

$$Tiempo\ Máximo = Min\ \Sigma\ T_{max(i)} - D$$

Denominamos actividades críticas a aquellas actividades cuyo Tiempo Mínimo y Tiempo Máximo coinciden, son actividades que no permiten retrasos pues su demora implicaría el retraso de la realización total del proyecto.

A partir de estos datos determinamos el calendario de ejecución del proyecto:

- La fecha de comienzo más temprana: Tiempo mínimo de cada actividad (T_{min}).
- La fecha de comienzo más tardía: Tiempo máximo (T_{max}) de cada actividad.

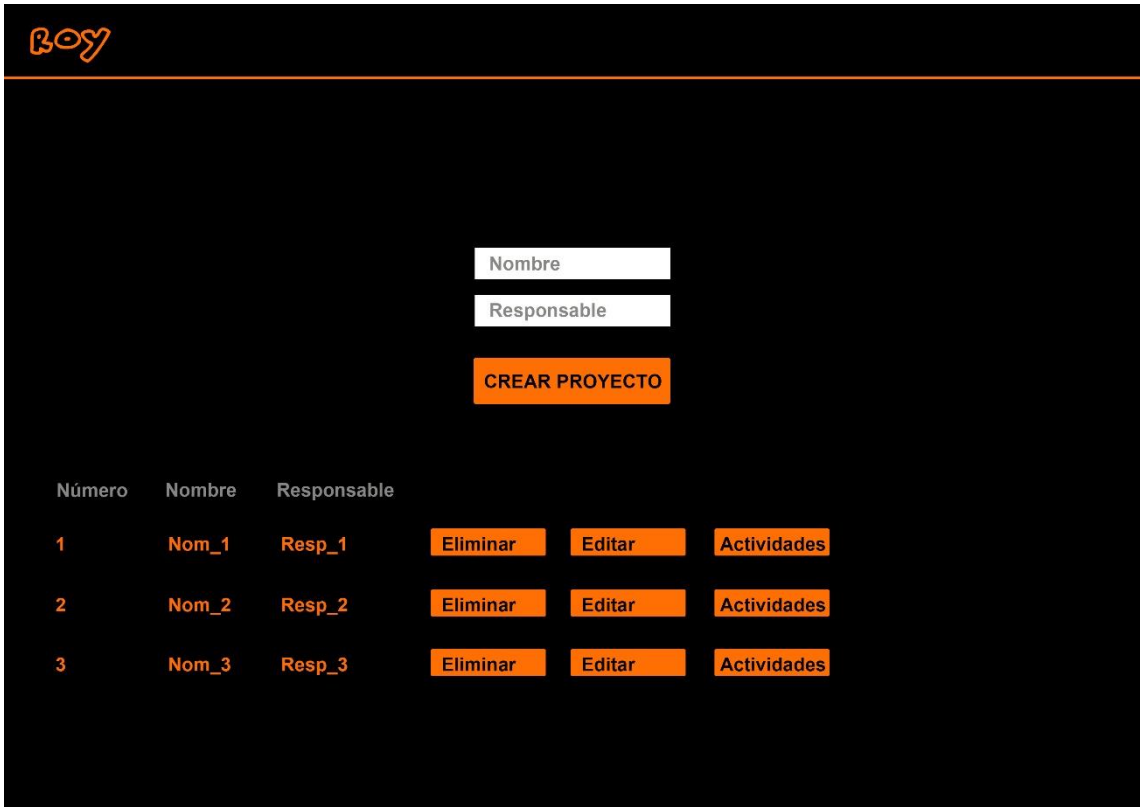
2.2. Requerimientos de la aplicación

Vamos a describir los requerimientos de la aplicación que se desea construir mediante la descripción de la interfaz de usuario de la misma.

2.2.1. Pantalla de inicio

En este apartado el usuario podrá crear un nuevo proyecto mediante un sencillo formulario en el que podrá introducir el nombre y el responsable del proyecto, una vez creado aparecerá en el listado de proyectos, donde tendrá acceso a más operaciones como describiremos a continuación. A través de este formulario comenzarán las primeras operaciones de comunicación con Base de Datos, se insertará un registro en la tabla ‘proyectos’ con los datos presentados y un identificador de proyecto, sobre esto se hablará más detalladamente en el apartado en el que se describen las tablas y sus respectivos campos de la Base de Datos.

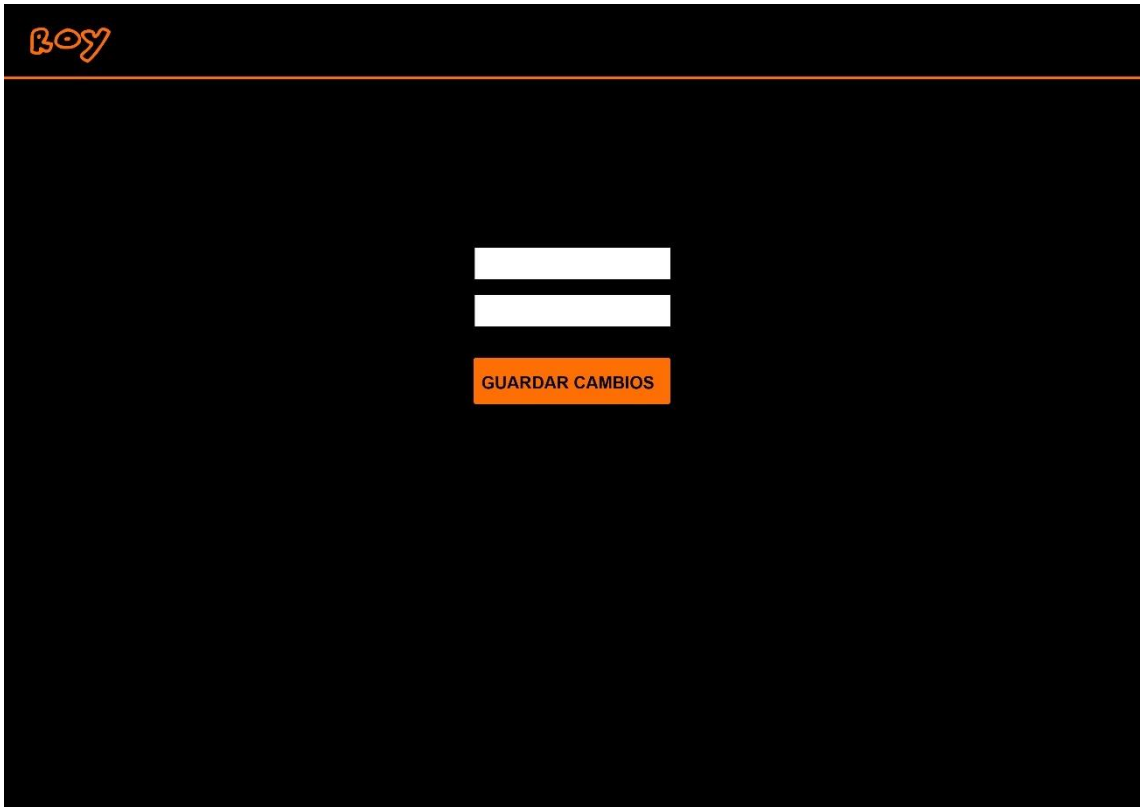
2.2.2. Listado de Proyectos



Cuando el usuario introduce los datos iniciales de un Proyecto estos se muestran automáticamente en la página ‘listado de proyectos’ como se muestra en la imagen.

Cada línea representa un Proyecto, un registro de la tabla ‘proyecto’ de la Base de Datos. Además, creamos tres botones que realizan diferentes operaciones, podremos eliminar datos, actualizarlos mediante el botón ‘editar’ e ingresar actividades, así como visualizarlas si ya estuvieran creadas. Desde esta interfaz el usuario tiene acceso a cualquiera de los proyectos creados y realizar las operaciones permitidas.

2.2.3. Editar Proyecto



En esta interfaz el Usuario podrá actualizar los datos principales del proyecto, Nombre y Responsable. Al pulsar el botón se actualizará el registro ya creado en la tabla ‘Proyectos’. Se cambiará el Nombre y/o el Responsable del proyecto, pero no la clave primaria que lo identifica, será una operación de actualización. Mantendremos el mismo proyecto, seguirá siendo el mismo registro.

2.2.4. Ingresar actividades

A este apartado accederemos mediante el botón Actividades de cada proyecto que se muestra en el listado de proyectos. Mediante este formulario podremos ingresar los datos de cada actividad, nombre, Duración, descripción y actividades precedentes.

Las operaciones que realizaremos serán de inserción en base de datos, una vez insertada una actividad se mostrará en el listado de actividades de manera análoga a como hicimos en el primer apartado de inserción de proyectos y como describimos en el siguiente apartado.

2.2.5. Listado de actividades

A medida que ingresamos actividades aparecerán ordenadamente en el listado de actividades con todos sus datos: número de actividad, nombre, duración, actividades precedentes, descripción, tiempo mínimo y tiempo máximo.

Capítulo 3
Diseño de la Aplicación

El objetivo del siguiente capítulo es proveer al lector de una visión general de la aplicación, para ello se ofrece una breve introducción teórica sobre el tipo de arquitectura empleada, Cliente / Servidor, y el modelo de organización de ficheros elegido, MVC o Modelo-Vista-Controlador. A continuación, se describe el aplicativo mediante un mapa o esquema en el que se muestran las relaciones entre Cliente, Aplicativo y Base de Datos, así como las diferentes interacciones entre los propios ficheros de la aplicación y las instrucciones o funciones más importantes de cada uno de ellos. Por último, se describen los campos principales de las tablas que conforman la Base de Datos.

3.1. Modelo Cliente / Servidor

La arquitectura de esta aplicación está basada en el modelo Cliente / Servidor, este modelo permite repartir las tareas entre diferentes equipos o entre diferentes aplicativos como en el caso de este proyecto. De esta manera se dividen las exigencias de procesamiento entre máquinas situadas en los puntos finales de las redes de comunicación, encargadas de realizar tareas como la visualización de datos a través de interfaces gráficas, y otras dedicadas a la realización de distintos servicios como el almacenamiento de datos o la gestión y tratamiento de estos.

Esto permite que cada equipo o aplicación involucrada se especialice en tareas concretas y utilice hardware y software específico.

3.1.1. Principios de funcionamiento

El principio fundamental de este tipo de arquitectura es la distinción entre los conceptos de cliente y servidor. Suele denominarse Cliente al equipo del usuario que solicita información a otras máquinas, enviando peticiones y esperando respuestas. Estas peticiones llegan a otros equipos llamados Servidores que responden a las demandas del cliente.

Pero la definición anterior no es del todo exacta, pues un cliente y un servidor pueden estar situados en un mismo equipo, como es el caso de instalación de servidores locales, para un uso formativo o de testeo como en el diseño de esta aplicación. No obstante, la funcionalidad última de esta arquitectura sí que reside en la división de distintas tareas entre equipos diferentes.

Clientes y servidores pueden formar parte de cualquier tipo de red, desde redes locales a Internet.

Cliente

Es un equipo o aplicación informática que solicita datos o servicios a otro equipo denominado servidor, generalmente mediante una red de telecomunicaciones, pero como ya hemos explicado anteriormente cliente y servidor pueden estar localizados en un mismo equipo. El cliente también procesará de alguna manera los datos recibidos del servidor.

Los servidores pueden utilizarse para la obtención de multitud de servicios diferentes como la interacción con otros usuarios o la utilización de recursos externos no disponibles en el propio equipo, pero en el caso que nos atañe nos centraremos en la obtención de páginas web y bases de datos. El navegador web es el cliente de esta aplicación y es muy utilizado porque pueden requerirse multitud de servicios a través suyo sin la necesidad de instalación de programas específicos.

Servidor

Normalmente se entiende por Servidor como una máquina o conjunto de ellas cuyo cometido es la realización de tareas destinadas a proveer de servicios a otros equipos cliente, pero un servidor puede ser también una aplicación que ofrezca servicios a otras.

Algunas de las funciones más importantes de los servidores son el almacenamiento de archivos o los servicios de aplicaciones.

3.1.2. Arquitectura Cliente / Servidor

Clientes y servidores se encuentran separados desde un punto de vista lógico, los clientes realizan peticiones y los servidores las reciben y procesan para posteriormente responder, los servicios de respuesta pueden consistir en el acceso a bases de datos, a dispositivos hardware como impresoras o ejecución de aplicaciones.

La relación cliente servidor suele ser desigual en el sentido de que un servidor suele ofrecer sus servicios a multitud de clientes.

Un servidor puede actualizarse sin que ello deteriore su comunicación con el cliente mientras el sistema de conexión entre ambos permanezca intacto.

La división lógica entre ambos hace que el sistema completo de comunicación sea más robusto y más tolerante a fallos de cualquiera de las partes. Un cliente puede no verse afectado por la caída de un servidor si esos mismos servicios pueden ofrecerse por otro servidor al que esté conectado.

La arquitectura cliente servidor óptima debería ser independiente de las tecnologías involucradas, tales como el hardware o los sistemas operativos empleados, permitiendo la utilización de distintas plataformas.

Permiten escalabilidad tanto horizontal como vertical, esto es, deberían poderse aceptar multitud de conexiones de clientes sin afectar a la calidad de la comunicación ni al tiempo de respuesta (escalabilidad horizontal), también deberían soportarse las actualizaciones o migraciones de equipos Servidores (escalabilidad vertical).

De esta manera un equipo servidor debe ofrecer servicios a multitud de clientes simultáneamente y gestionar el acceso de estos a los recursos que ofrece.

Existen multitud de tipos de servidores según las funciones que desempeñen, pero vamos a nombrar solamente a los que competen al cometido de este trabajo, como son los servidores de Base de Datos, los Servidores de aplicación y los servidores Web.

- **Servidores de Base de Datos**:

  Almacenan grandes cantidades de datos estructurados. El cliente realiza consultas y el servidor responde solo con la información especificada.

- **Servidores Web**:

  Almacenan páginas HTML, archivos CSS, PHP, Javascript…etc.

- **Servidores de Aplicación**:

  Servidores dedicados a aplicaciones, Los clientes podrán acceder a ellas de forma remota y solicitar sus servicios sin necesidad de tenerlas instaladas localmente.

Arquitectura Cliente Servidor de Tres Capas

La arquitectura de tres capas se divide entre la capa de presentación, la de lógica y la de base de datos. La lógica está aislada de la base de datos, por lo que los cambios en una de las partes no deben afectar a las demás.

Algunas ventajas e inconvenientes

Este tipo de arquitectura facilita la integración entre diferentes tecnologías y sistemas, y permite que puedan mantenerse las estructuras de bases de datos y lógica ante, por ejemplo, cambios de interfaces más amigables. Además, favorece el uso en sistemas operativos diferentes.

Las aplicaciones construidas bajo estas reglas favorecen el uso de interfaces más interactivas, razón por la cual resultan más atractivas e intuitivas para los usuarios finales, ahorran ancho de banda porque no se hace tan necesario enviar información gráfica, pues esta puede residir en el equipo cliente.

Facilitan la integración de nuevas tecnologías en el sistema.

Como principal inconveniente, destacaremos que el mantenimiento de los sistemas puede resultar bastante complejo, pues al estar distribuidos en diferentes equipos y ser proporcionados por distintos proveedores es más difícil integrar las soluciones.

Protocolo HTTP

HTTP (Protocolo de Transferencia de Hipertexto) es un protocolo de la capa de aplicación encargado de transferir páginas de hipertexto entre el cliente Web y el servidor.

En esta aplicación se utiliza este protocolo para efectuar las comunicaciones entre el cliente y el servidor, las peticiones que haremos serán de tipo GET y tipo POST.

- GET: Solicitud de consulta a los recursos.
- POST**:** insertar nuevos recursos.

Software Intermedio o Middleware

El Middleware es la capa de software intermedio para la comunicación entre cliente y servidor, el middleware direcciona las peticiones del cliente determinando la ubicación de los recursos requeridos, el servidor web devuelve la información al middleware y este de nuevo la reenvía al cliente.

3.1.7. Modelo vista controlador (MVC)

El Modelo-Vista-Controlador divide la lógica del aplicativo de la lógica de la vista.

Como veremos un poco más adelante, la estructura de esta aplicación está basada en este modelo, dividiendo los archivos principales en un archivo encargado de establecer las rutas para las peticiones, otro que contiene la lógica del negocio, y una carpeta con las vistas que se entregarán al Cliente para mostrar los datos.

Utilizamos este modelo para continuar con la filosofía de dividir el problema en partes y que los cambios en una de las partes de la aplicación no afecten, o afecten lo menos posible, al resto.

El controlador responde a las solicitudes del cliente, una vez que ha recibido una solicitud, este a su vez solicita al modelo o lógica del negocio el recurso requerido, el modelo se comunica con la base de datos y responde al controlador entregando los datos que se han solicitado.

Cuando el controlador dispone de los datos solicitados los entrega a la Vista que se encarga de mostrarlos en el navegador.

Modelo

En esta aplicación el modelo lo constituye el objeto ‘logica.js’, que es el encargado de consultar a la base de datos y realizar las operaciones pertinentes.

Controlador

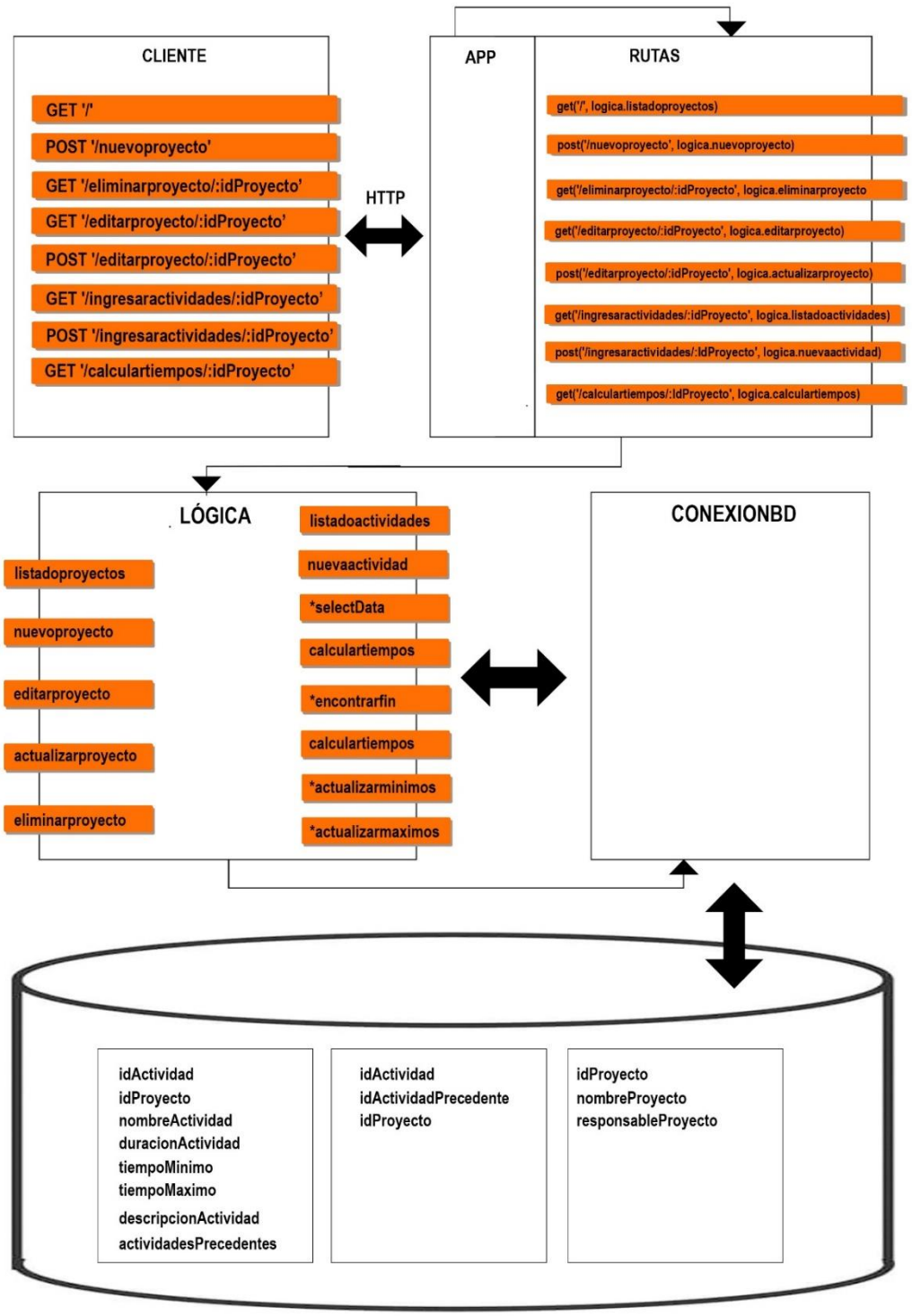
El controlador está implementado entre el fichero ‘rutas.js’ y ‘app.js’, el primero se encarga de enrutar las peticiones y el segundo es quien se comunica con las Vistas.

Vista

Estos archivos, ubicados en la carpeta ‘views’ son los responsables últimos de mostrar los datos por pantalla.

3.2. Diseño de la Aplicación

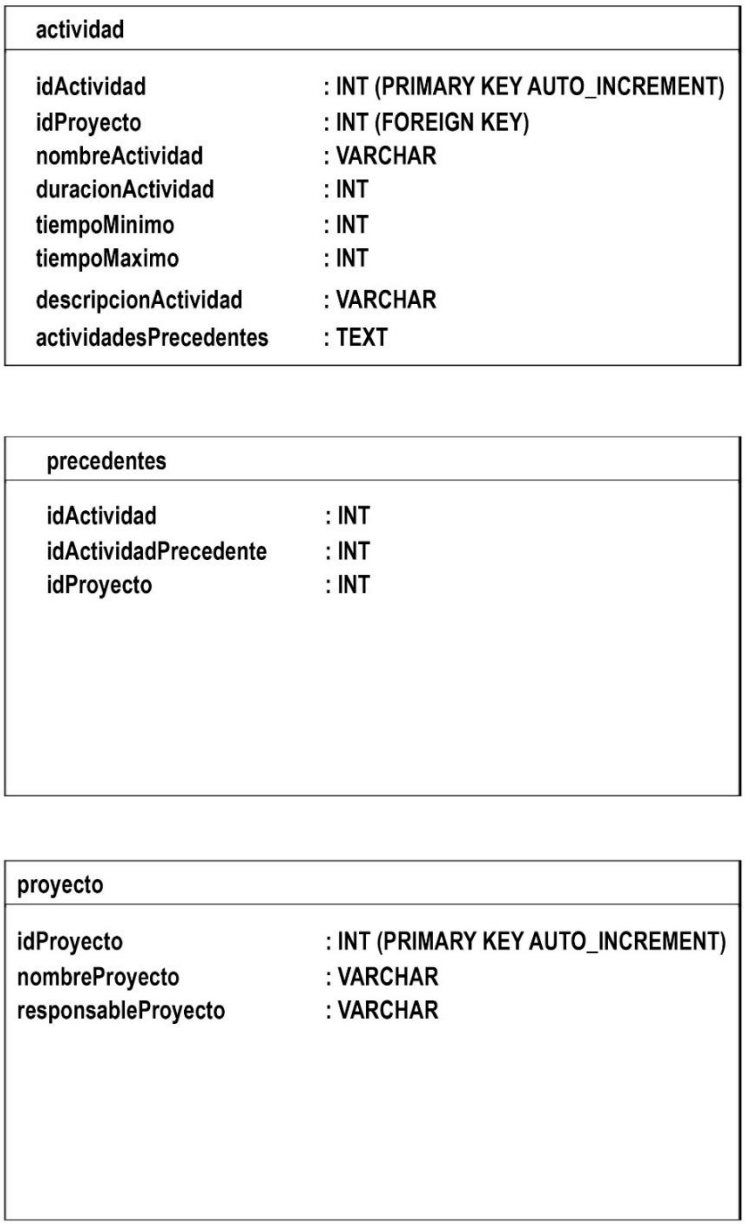
La siguiente figura muestra de forma esquemática la relación entre el Cliente y el Servidor, por una parte, y la relación entre el aplicativo y la Base de Datos por otra, así como las diferentes interacciones que ocurren entre los diferentes ficheros de la aplicación.



Como se observa en la figura, el Cliente realiza peticiones HTTP al Servidor, dentro de este, el encargado de gestionar el sistema general es app.js que se comunica con rutas.js, este a su vez direcciona las peticiones HTTP referenciando cada una de ellas a las funciones pertinentes descritas en ‘logica.js’. Finalmente, ‘logica.js’ se comunica con ‘conexionBD.js’ que es quien describe la conexión a la Base de Datos.

3.3. Estructura de las Tablas de la Base de Datos

En la siguiente imagen mostramos la estructura de las tablas que componen la base de datos y la definición de sus campos, destacamos el tipo de dato que define a cada campo, así como si constituye o no Clave Primaria o Secundaria.



Las Tablas ‘proyecto’ y ‘actividad’ contienen un id o identificador único, las relaciones entre tablas se hacen mediante sus claves primarias, no suele ser recomendable relacionar tablas mediante campos que no constituyan claves primarias. En la tabla ‘precedentes’ no fue necesario crear Claves Primarias para los registros.

Capítulo 4
Implementación

Llegados a este punto el lector ya dispone de un mapa general que le permite comprender la estructura general del aplicativo y su funcionamiento. Es por ello por lo que en el siguiente capítulo describiremos la implementación definitiva de la aplicación, describiendo el software utilizado para su construcción, la estructura definitiva de directorios y ficheros explicándose los archivos más relevantes y haciendo hincapié en las instrucciones o funciones fundamentales de cada uno de ellos.

Por último, expondremos algunos de los problemas más relevantes con los que nos hemos encontrado y explicaremos detalladamente las soluciones por las que optamos para sortearlos.

4.1. Descripción de Tecnologías

4.1.1. NodeJS

NodeJS es un framework Open Source basado en Javascript de código abierto, consta de un servidor web y un entorno de desarrollo para aplicaciones web.

Una de las particularidades de NodeJS es el uso de Javascript en el lado del servidor, lenguaje tradicionalmente ejecutado desde el lado del cliente.

La adaptabilidad de Javascript para el uso de funciones asíncronas y su robustez frente a conexiones múltiples al servidor son otras cualidades que hacen a NodeJS un framework muy popular entre los desarrolladores de este tipo de aplicaciones.

Contiene un gestor de paquetes NPM (Node Package Manager) que ofrece acceso a múltiples librerías de código abierto con multitud de funcionalidades necesarias para el desarrollo de aplicaciones, algunas de las más importantes son express, body-parser, Jade, Pug, Mongoose o mysql entre otras.

4.1.2. ExpressJS

Es quizás el framework de Node.js más utilizado para el desarrollo de aplicaciones web, provee al desarrollador de la estructura básica de la aplicación, de hecho, tras la instalación se dispone de una aplicación sencilla montada que muestra un archivo con el clásico “Hello World” por pantalla. Consta de todas las funcionalidades disponibles en NodeJS. Y provee a la aplicación de una serie de estructuras de funcionamiento predefinidas que facilitan mucho el funcionamiento de la aplicación:

-Gestores de peticiones HTTP en diferentes localizaciones URL (Rutas).

-Motores de renderización vistas para la integración de datos en pantalla mediante plantillas creadas generalmente con motores de HTML como Jade o Pug.

-Ajustes predefinidos de funcionamiento de aplicaciones web como configuraciones sobre que puerto usar para las conexiones y la localización de las plantillas que se usan para la renderización de las respuestas del servidor.

4.1.3. MySQL

MySQL es una estructura de base de datos relacional, su sintaxis es bastante sencilla y además es multihilo y multiusuario, esto es, permite múltiples conexiones simultaneas, es, por tanto, un sistema de base de datos muy eficaz para aplicaciones Web.

4.1.4. WampServer

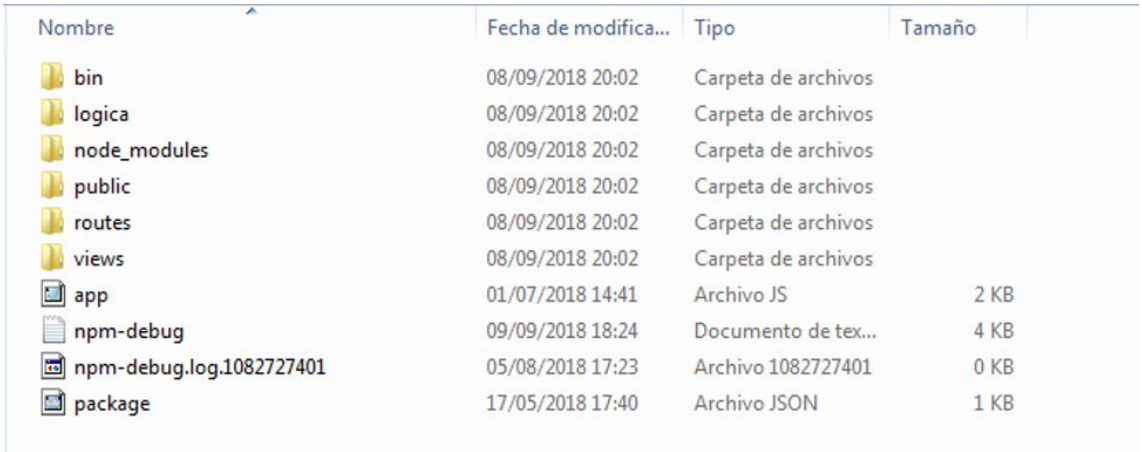
Es una estructura que suele usarse para la implementación de servidores locales. Utiliza Windows como sistema operativo y MySQL como base de datos, Apache como servidor y PHP como lenguaje de programación.

4.2. Estructura de la aplicación

En este apartado describiremos esquemáticamente la estructura de directorios y ficheros del aplicativo. Explicaremos brevemente el cometido de cada uno y destacaremos sus instrucciones más importantes para obtener una visión general de su funcionamiento.

4.2.1. Descripción general de Directorios y Ficheros

La estructura de directorios definitiva del aplicativo es la que vemos en la siguiente imagen, una adaptación del modelo que crea express por defecto. Hemos realizado los cambios pertinentes para crear una estructura de Modelo-Vista-Controlador estricta.



Nombre	Fecha de modifica...	Tipo	Tamaño
bin	08/09/2018 20:02	Carpeta de archivos	
logica	08/09/2018 20:02	Carpeta de archivos	
node_modules	08/09/2018 20:02	Carpeta de archivos	
public	08/09/2018 20:02	Carpeta de archivos	
routes	08/09/2018 20:02	Carpeta de archivos	
views	08/09/2018 20:02	Carpeta de archivos	
app	01/07/2018 14:41	Archivo JS	2 KB
npm-debug	09/09/2018 18:24	Documento de tex...	4 KB
npm-debug.log.1082727401	05/08/2018 17:23	Archivo 1082727401	0 KB
package	17/05/2018 17:40	Archivo JSON	1 KB

Haremos una breve descripción de algunos de los directorios que express crea por defecto para posteriormente centrarnos en los más importantes para el funcionamiento de la aplicación.

El directorio ‘bin’ es el responsable de decirle a la aplicación el puerto en el que debe escuchar, por defecto se configura en el puerto 3000 de localhost.

El directorio ‘node_modules’ contiene módulos por defecto de express que dotan a la aplicación de muchísimas funcionalidades predeterminadas muy útiles para el buen funcionamiento de esta.

El archivo ‘app’ como veremos a continuación es el encargado de arrancar el aplicativo.

El fichero ‘package’ contiene metadatos como el nombre, la versión, el archivo para la ejecución del programa y podría contener otros datos importantes. Además, se encarga de describir las dependencias del sistema general de otros módulos secundarios.

4.2.2. Descripción de los ficheros principales

APP.JS

En este archivo se definen en primer lugar los módulos necesarios para el correcto funcionamiento de la aplicación mediante funciones ‘require’, algunos de los más importantes se muestran a continuación:

```
var express = require('express');
var app = express();
var rutas = require('./routes/rutas');
```

La primera instrucción requiere el módulo ‘express’ que nos proporcionará todas sus funcionalidades. Lo podremos utilizar creando la variable ‘app’ de la segunda línea.

La tercera instrucción nos permitirá acceder al módulo ‘rutas’ donde están definidas las rutas para las peticiones.

Este archivo dispone de otros fragmentos de código para la configuración de la aplicación donde se describe, por ejemplo, que motor de vistas usar, se configura el uso de un módulo para entender los datos de los formularios y otras funcionalidades implementadas por defecto para el correcto funcionamiento de una aplicación web. También aquí se le indicará que módulo debe usar cuando el cliente se conecte a la aplicación a través del puerto 3000 de localhost, que es el que viene por defecto implementado en el framework.

```
app.use(‘/’, rutas);
```

RUTAS.JS

Este archivo es el encargado de direccionar las peticiones HTTP del Cliente.

Algunas de las instrucciones más importantes se muestran a continuación:

```
var express = require('express');
var router = express.Router();
```

```
var logica = require('../logica/logica');
```

Requerimos ‘express’ y creamos una variable ‘router’ llamando a su módulo ‘router’, de esta manera podemos gestionar las peticiones htttp del cliente.

También requerimos el módulo ‘logica.js’ donde está implementada la lógica del negocio de la aplicación, por tanto, podremos llamar a sus métodos.

El direccionamiento de las peticiones HTTP se efectúa mediante instrucciones de este tipo:

```
router.get(‘/’, lógica.listadoproyectos);
```

Esto quiere decir que cuando se haga una petición http de tipo GET desde el index (‘/’) se llamará a la función ‘listadoproyectos’ de ‘logica.js’

Mediante la siguiente instrucción podremos llamar al módulo ‘rutas.js’ desde ‘app.js’, u otro módulo que lo requiriese, donde lo hemos importado mediante la función ‘require’:

```
module.exports = router;
```

LOGICA.JS

En este fichero se implementa la lógica del negocio, el conjunto de funciones que definen el propósito de la aplicación, también aquí se creará una variable, ‘conexionBD’, que llamará al archivo donde se define la conexión a la Base de Datos, por tanto podremos utilizar sus métodos de consulta. A continuación, definimos brevemente las funciones e instrucciones principales:

La siguiente instrucción crea una variable que apunta al módulo ‘conexionBD’:

```
var conexionBD = require("./ConexionBD.js");
```

Función ‘listadoproyectos’:

realiza una consulta SELECT a la base de datos, obtiene los registros de la tabla ‘proyecto’ y los muestra en el index.

Función ‘nuevoproyecto’:

Insertará un registro en la tabla ‘proyecto’ de la base de datos con los datos ‘nombreProyecto’, ‘responsableProyecto’ e ‘idProyecto’ mediante una instrucción SQL de tipo INSERT a la base de datos.

Función ‘editarproyecto’:

Selecciona el registro que contiene el dato idProyecto de la tabla ‘proyecto’ que le pasamos y nos lleva a la página de actualización de proyectos.

Función ‘actualizarproyecto’:

Actualiza un proyecto en la tabla ‘proyecto’ mediante la instrucción UPDATE con los datos que le enviamos a través del formulario.

Función ‘eliminarproyecto’:

Elimina un proyecto mediante la instrucción DELETE.

Función ‘listadoactividades’:

Funciona de manera análoga a la función ‘listadoproyectos’ pero mostrando la lista de actividades para un proyecto dado.

Función ‘nuevaactividad’:

Ingresa los datos insertados a través del formulario correspondiente de una actividad en la tabla ‘actividad’.

Función ‘calculartiempos’:

Calcula los Tiempos Mínimo y Máximo de cada actividad, finalmente actualiza los datos por pantalla en el listado de actividades.

Finalmente, para cada función crearemos una instrucción de este tipo para que puedan ser llamadas por el módulo ‘rutas.js’:

```
exports.nuevoproyecto=nuevoproyecto;
```

CONEXIONBD.JS

En este fichero se implementa la conexión a la Base de Datos, como hemos visto se le llama desde ‘logica.js’ para poder realizar las consultas pertinentes a la Base de Datos.

Mediante la siguiente instrucción podremos llamar a los métodos de ‘mysql’, que es un módulo de express para conexiones con bases de datos mysql.

```
var mysql= require('mysql');
```

Creamos la variable ‘conexionBD’ llamando a un método del módulo ‘mysql’ con todos los datos de la conexión:

```
var conexionBD= mysql.createConnection({
host: '127.0.0.1',
  user: 'root',
  password: '',
  database: 'dbproyectoroyprueba'
});
```

Finalmente exportamos el módulo para poder utilizarlo desde ‘logica.js’

```
module.exports = conexionBD;
```

4.3. Principales problemas resueltos

4.3.1. Problema de interpretación de Arrays

Al obtener los datos de las actividades precedentes a través del formulario de tipo ‘checkbox’ nos topamos con un problema un tanto extraño, cuando ingresábamos solamente una actividad precedente, obteníamos un vector con tantas posiciones como cifras tenía el dato ‘idActividad’ que recibíamos, es decir para una actividad con un valor de ‘idActividad’ de valor 100 deberíamos obtener un vector de la forma [100], pero obteníamos un vector de tres posiciones: [1,0,0]. En cambio, cuando ingresábamos más de una actividad precedente nos devolvía vectores de la forma esperada.

La solución fue crear una actividad nueva, a la que llamamos ‘actividadTrampa’, activarla en el ‘checkbox’ e invisibilizarla, de manera que el vector ‘actividadesPrecedentes’ siempre recogiera al menos dos actividades.

Mediante las siguientes instrucciones obtenemos los datos ‘idActividad’ del checkbox.

```
var actividadesPrecedentes=[];
actividadesPrecedentes=req.body.actividadesPrecedentes;
```

Creación de la Actividad Trampa:

```
var datosActividadTrampa = {IdProyecto: idProyecto,
nombreActividad:’Actividad Trampa', descripcionActividad:'Actividad
Trampa’};
```

En la siguiente imagen observamos la implementación del código su ‘view’ correspondiente para sortear la Actividad Trampa:

```
45  <% if (dataact) { %>
46
47   <li class="checkbox keep-open">
48          <label>
49                  <input type="checkbox" name="actividadesPrecedentes" value="<%= dataact[1].idActividad %>">
50                  <%= dataact[1].nombreActividad %>
51          </label>
52   </li>
53
54  <% for(var i = 3; i < dataact.length; i++) { %>
55
56  <li class="checkbox keep-open">
57          <label>
58                  <input type="checkbox" name="actividadesPrecedentes" value="<%= dataact[i].idActividad %>">
59                  <%= dataact[i].nombreActividad %>
60          </label>
61  </li>
62
63  <% } %>
64
65  <li class="checkbox keep-open">
66          <label>
67                  <input type="checkbox" name="actividadesPrecedentes" value="<%= dataact[0].idActividad %>"  checked="checked" style="opacity:0;">
68                  <!--<%= dataact[0].nombreActividad %>-->
69          </label></li>
70
71  <% } %>
```

La primera actividad que se ingresa en un proyecto es siempre ‘actividadTrampa’, por tanto, corresponde siempre a la posición 0 del registro, ‘dataact’ contiene los datos devueltos por la consulta a la Base de Datos desde ‘logica.js’, por tanto, ‘dataact[0]’ corresponde al registro que contiene la ‘actividadTrampa’ y el valor para el atributo ‘checked’ es “checked”, esto es, la actividad siempre está seleccionada.

Mediante el atributo ‘style’ le damos un valor de opacidad 0 para invisibilizarla al usuario, aunque esto es redundante, porque como puede verse en el listado de actividades precedentes a mostrar, la primera línea de la lista siempre será ‘actividadInicio’ (‘dataact[1]’), al crear un proyecto esta actividad siempre se ingresa en la posición 1.

En la siguiente línea mediante la condición de la instrucción ‘for’ le indicamos que comience a mostrar las actividades a partir de la posición 3, es decir, a partir de la primera actividad ingresada por el usuario. Y por último, en la última línea, le indicamos que muestre el dato ‘dataact[2]’, que corresponde a ‘actividadFin’.

4.3.2. Problemas de sincronía

Los problemas más importantes han surgido a la hora de encadenar consultas a la Base de Datos, consultas dentro de bucles ‘for’ o consultas encadenadas dentro de las funciones de otras consultas. El tiempo que puedan demorarse estas consultas no es previsible y podemos necesitar datos de funciones que estén ejecutándose todavía, por lo que no es válida la programación secuencial en estos casos. Estos problemas comenzaron en la función ‘nuevaactividad’, a la hora de mostrar los datos actualizados por pantalla, operaba correctamente, pero no actualizaba en el momento correcto el campo Actividades Precedentes.

A partir de este momento se abordó de nuevo el diseño de la función desde una perspectiva diferente, se estudió el funcionamiento de las funciones ‘callback’ y se encadenaron las consultas. Para entender el funcionamiento de la función he considerado necesario hacer una breve introducción teórica de la programación asíncrona y el uso de ‘callbacks’, y explicar el funcionamiento mediante un esquema, pues son muchas consultas encadenadas y la mejor forma de comprenderla es mediante la visualización de los encadenamientos.

PROGRAMACIÓN ASÍNCRONA

Este tipo de programación nos permite posponer la ejecución de una función dependiendo de haber concluido previamente alguna otra instrucción o conjunto de ellas. En nuestro caso particular nos hemos visto obligados a utilizarla en casos de consultas reiteradas a la base de datos como hemos comentado anteriormente. Básicamente consiste en pasar funciones como parámetros a otras funciones, a estas funciones se les denomina funciones ‘callbacks’.

SINCRONÍA Y ASINCRONÍA

Mediante un diseño de programación síncrona cada instrucción espera a la anterior para ejecutarse, en cambio mediante la programación asíncrona no se espera a la consecución de las instrucciones anteriores y el código continúa ejecutándose independientemente de que las instrucciones predecesoras se hayan implementado o no.

CALLBACKS

Este tipo de funciones son funciones recibidas como parámetros por otra función, a la que podemos llamar función principal, y se ejecutarán en algún punto de la secuencia de instrucciones definidas en su función principal.

FUNCIÓN NUEVAACTIVIDAD

```
var nuevaactividad = function (req, res, next){

  datosActividadStandard // contiene todos los datos menos idActividad y actividadesPrecedentes
  actividadesPrecedentes // contiene las 'idActividad' de cada precedente marcada en el checkbox
  conexionBD.query('INSERT into actividad SET?, datosActividadStandard, function(err, rows){

    idActividad // Ya tenemos la idActividad
    tiempoMinimot = 0
    encontrarFin(idProyecto, function(fin){

      Primero ejecuto las instrucciones de encontrarFin y luego haré el callback,
      conexionBD.query('SELECT* FROM actividad WHERE idProyecto... ')
      // Encuentro la actividadFin de este proyecto y paso el registro al callback (rows[0])
      Hago el callback:

      for(i=0; i<actividadesPrecedentes.length; i++) // recorro todas las actividades precedentes
      selectData(actividadesPrecedentes[i], function (err, actividad){

        Primero ejecuto las instrucciones de selectData
        conexionBD.query('SELECT* FROM actividad WHERE idActividad =?', actividadesPrecedentes[i])
        // cojo cada actividadPrecedente que le paso de la tabla 'actividad' y la paso al callback (rows[0])
        Hago el callback:

        conexionBD.query('INSERT INTO precedentes SET ?',
        {idActividad: idActividad, idActividadPrecedente: actividad.idActividad, idProyecto: idProyecto]})
        function (err, rows){

          conexionBD.query('DELETE from precedentes
          WHERE idActividad = ? and idActividadPrecedente = ?',[fin.idActividad, actividad,idActividad]f );
          //Elimino la actividad como precedente de Fin
          la añado a actividadesPrecedentesNames
          if (i===actividadesPrecedentes.length)
          // Si es la última del vector actividadesPrecedentes:
          conexionBD.query('UPDATE actividad SET? WHERE idActividad = ?',
          [{actividadesPrecedentesNames.join(',')}, idActividad]
          // Actualizo el string de actividades precedentes.
          function(err, rows){

            // selecciono las actividades del proyecto:
            conexionBD.query('SELECT* FROM actividad WHERE idProyecto = ?',[idProyecto], function(err,rows){

              // Muestro actividades por pantalla
              res.render('listadoactividades', {dataact: rows})

        conexionBD.query('INSERT into precedentes SET ?', {
        [idActividad: fin.idActividad, idActividadPrecedente: idActividad, idProyecto: idProyecto]}
```

Capítulo 5
Guía de Usuario

En el presente capítulo se ofrece al lector, de un perfil técnico o funcional, sendos manuales de Instalación y Usuario. En el primero de ellos se dan algunos detalles para la instalación del software implicado y la consecuente puesta en marcha del aplicativo, en el segundo se hace un recorrido detallado por todas las páginas de la Interfaz de Usuario explicando el funcionamiento de la aplicación.

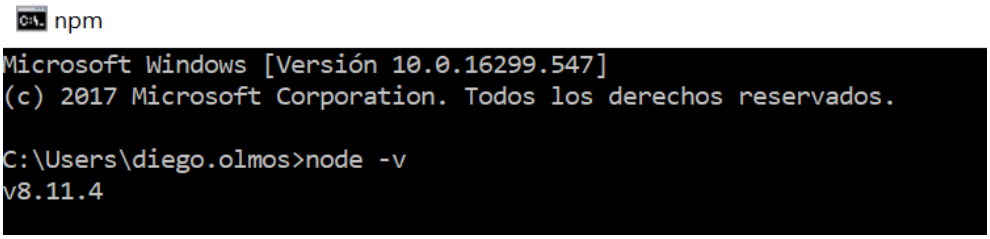
5.1. Manual de Instalación

5.1.1. Instalación de NodeJS y ExpressJS

Entramos en la página oficial de NodeJS *https://nodejs.org/en/* y descargamos el software.

Instalamos NodeJS como cualquier otra aplicación en Windows.

Abrimos la consola de comandos CMD, tecleamos `node -v` y pulsamos Intro, comprobamos que Node está instalado:

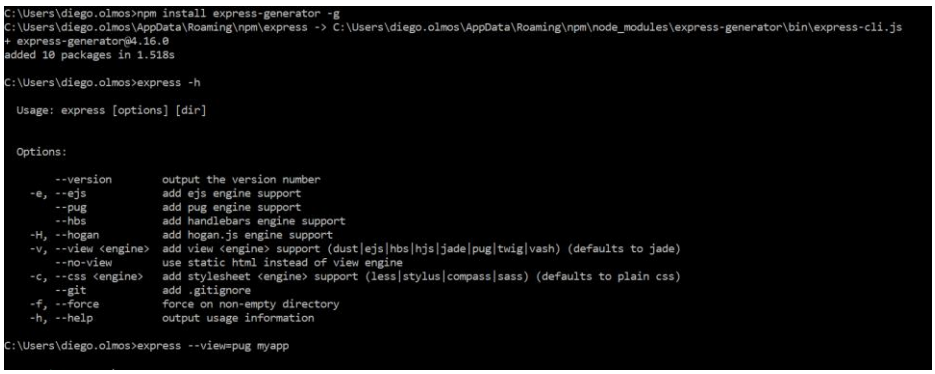


Tecleamos las siguientes instrucciones:

- `npm install express-generator -g`

- `express -h:`

Creamos una aplicación Express denominada myapp o el nombre que queramos darle a la aplicación.



Instalamos las dependencias: `cd my app > npm install ()`

En Windows, utilizamos este comando: `> set DEBUG = myapp:* & npm start`

En MacOS o Linux: `$ DEBUG = myapp:* npm start`

```
C:\Users\diego.olmos>express --view=pug myapp

   create : myapp\
   create : myapp\public\
   create : myapp\public\javascripts\
   create : myapp\public\images\
   create : myapp\public\stylesheets\
   create : myapp\public\stylesheets\style.css
   create : myapp\routes\
   create : myapp\routes\index.js
   create : myapp\routes\users.js
   create : myapp\views\
   create : myapp\views\error.pug
   create : myapp\views\index.pug
   create : myapp\views\layout.pug
   create : myapp\app.js
   create : myapp\package.json
   create : myapp\bin\
   create : myapp\bin\www

   change directory:
     > cd myapp

   install dependencies:
     > npm install

   run the app:
     > SET DEBUG=myapp:* & npm start


C:\Users\diego.olmos>cd myapp

C:\Users\diego.olmos\myapp>npm install
npm notice created a lockfile as package-lock.json. You should commit this file.
added 120 packages in 12.012s

C:\Users\diego.olmos\myapp>set DEBUG=myapp:* & npm start

> myapp@0.0.0 start C:\Users\diego.olmos\myapp
> node ./bin/www
```

Cargamos *http:///localhost:3000/* en el navegador para acceder a la aplicación.

Sustituir ficheros

Una vez instalado el esqueleto de la aplicación debemos sustituir algunos directorios y ficheros creados por express por defecto por los nuestros propios. En primer lugar, sustituiremos el archivo ‘app.js’ que viene por defecto por el de nuestra aplicación, a continuación haremos lo mismo con ‘package.json’ y con las carpetas ‘views’ y ‘routes’, Finalmente añadiremos la carpeta ‘logica’.

5.1.2. Instalación de Wampserver

Instalamos Wampserver de forma muy sencilla siguiendo su guía de instalación. Para funcionar de forma óptima Wamp ha de estar conectado a Internet, por tanto, debemos cerciorarnos de que el puerto 80 no esté siendo utilizado por ninguna otra aplicación. Un indicativo de que esto estuviera ocurriendo podría ser que el icono de Wamp situado en la barra de tareas no apareciera de color verde.

Importar la Base de Datos

Una vez instalado Wamp, desde localhost / phpMyAdmin podemos acceder a MySQL e importar la base de datos deseada. Para ello deberemos crear una base de datos y desde ella importar las tablas del archivo proporcionado (.sql) mediante el botón 'Importar', desde el botón 'Examinar' podremos acceder a la ubicación del archivo. Seleccionaremos el fichero sql con las tablas deseadas y pulsaremos 'continuar'.

5.1.3. Arrancar la aplicación

Desde la consola de comandos teclearemos la siguiente instrucción:

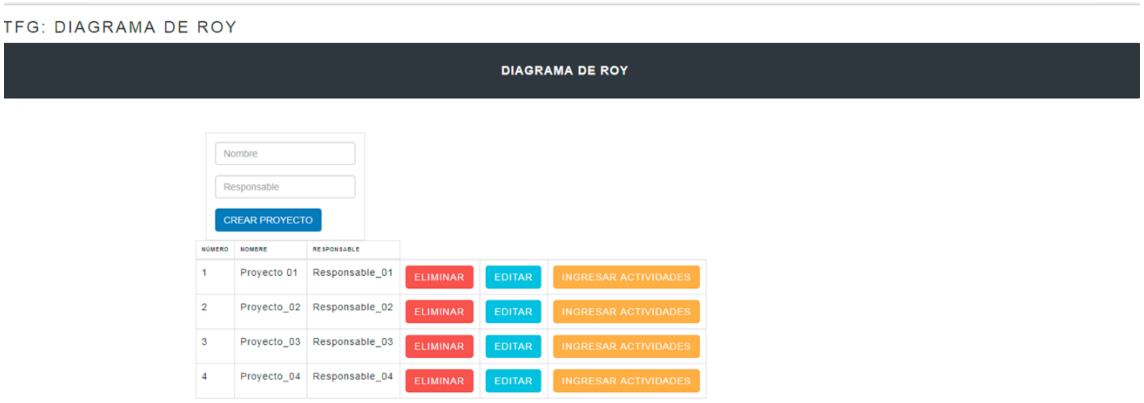
```
> set DEBUG = myapp:* & npm start
```

Y ya podremos utilizar la aplicación desde localhost:3000.

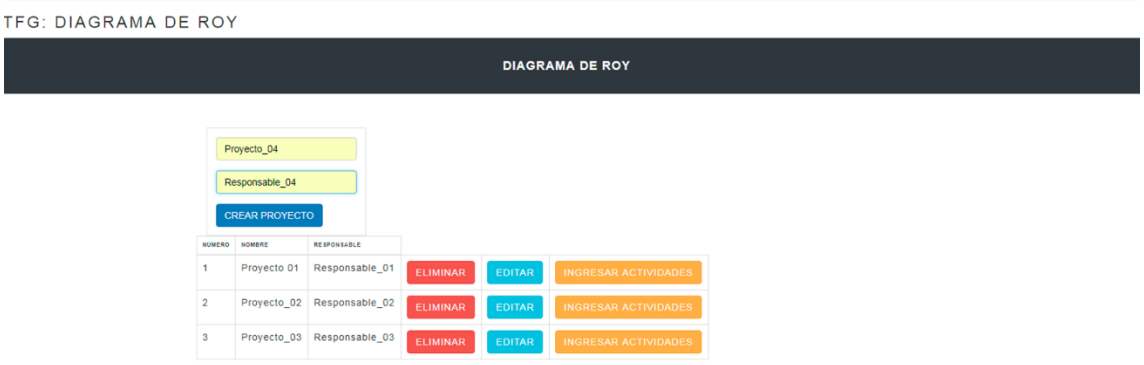
5.2. Manual de Usuario

5.2.1. Listado de proyectos

Al acceder a la aplicación por primera vez aparecerá una pantalla con un formulario con dos campos con el cometido de que el usuario introduzca el nombre y responsable del proyecto sobre el que se va a trabajar. En el caso de que existan más proyectos creados aparecerán listados debajo del formulario como se muestra en la siguiente imagen. Al accionar el botón ‘Crear Proyecto’ el Proyecto habrá sido creado y aparecerán sus datos identificativos en el listado junto al resto de proyectos existentes. A partir de este momento ya pueden realizarse acciones sobre el mismo a través de los tres botones que aparecen en cada línea de proyecto. Puede eliminarse el proyecto, pueden actualizarse los datos identificativos y se puede acceder a la pantalla de ingreso de actividades.



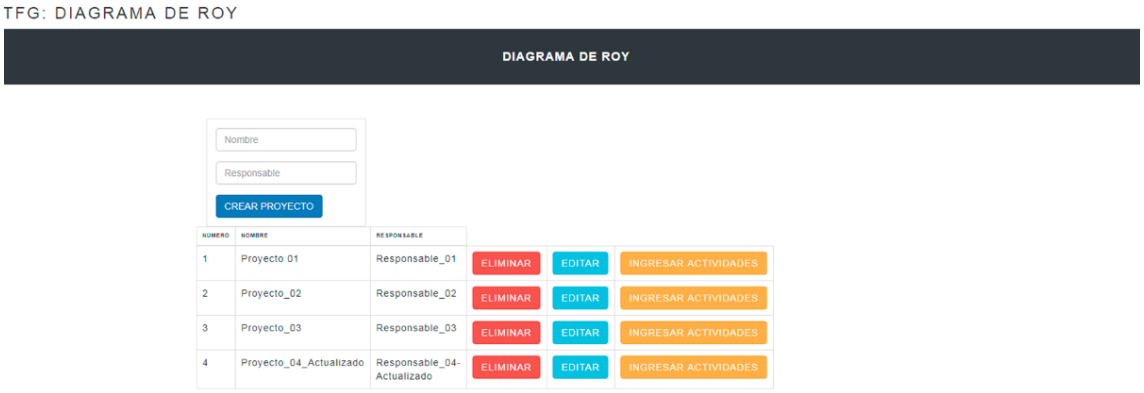
Dicho esto, comenzaremos a crear un proyecto nuevo ingresando su nombre y su responsable como se muestra en la siguiente imagen.



5.2.2. Editar Proyecto

Al pulsar el botón ‘EDITAR’ accederemos a la pantalla de actualización de los datos identificativos, aparecerá un nuevo formulario con dos campos en el que se podrá actualizar el Nombre y el Responsable del proyecto. Al pulsar el botón ‘GUARDAR CAMBIOS’ el proyecto quedará actualizado y accederemos de nuevo a la pantalla inicial con el listado de proyectos. A continuación, procedemos a mostrar cómo se cambian los datos mencionados.

Como puede verse los datos han sido actualizados y así aparece en el listado de proyectos.

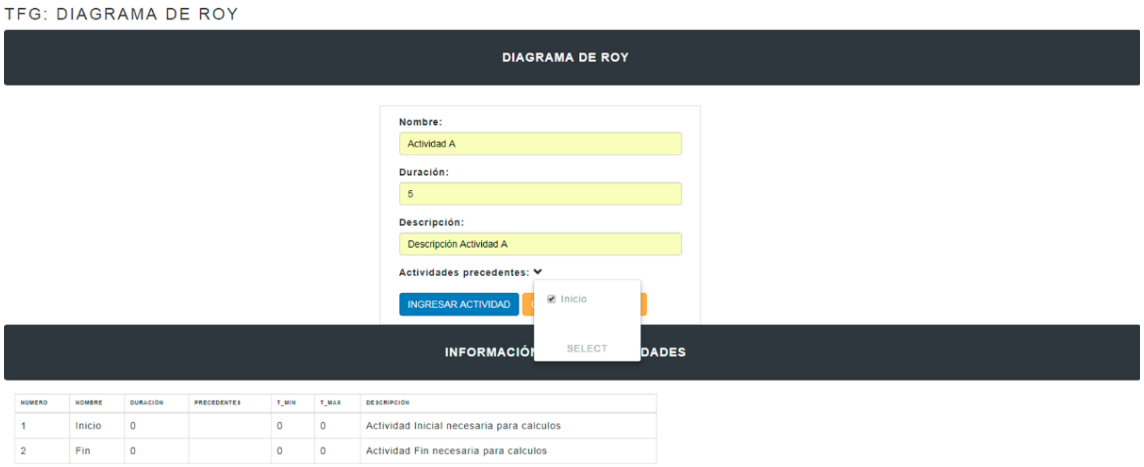


5.2.3. Ingresar Actividades

Al pulsar el botón ‘Ingresar Actividades’ el usuario accederá a una nueva pantalla en la que aparecerá un formulario con los datos necesarios para la realización de los cálculos del diagrama de Roy. Se le pedirá que ingrese un nombre para la actividad, su duración, una breve descripción y sus actividades precedentes. En el caso de que una o varias actividades no tengan actividades precedentes, es decir, las actividades iniciales del proyecto, el usuario deberá marcar la casilla ‘Inicio’ del desplegable ‘Actividades precedentes’

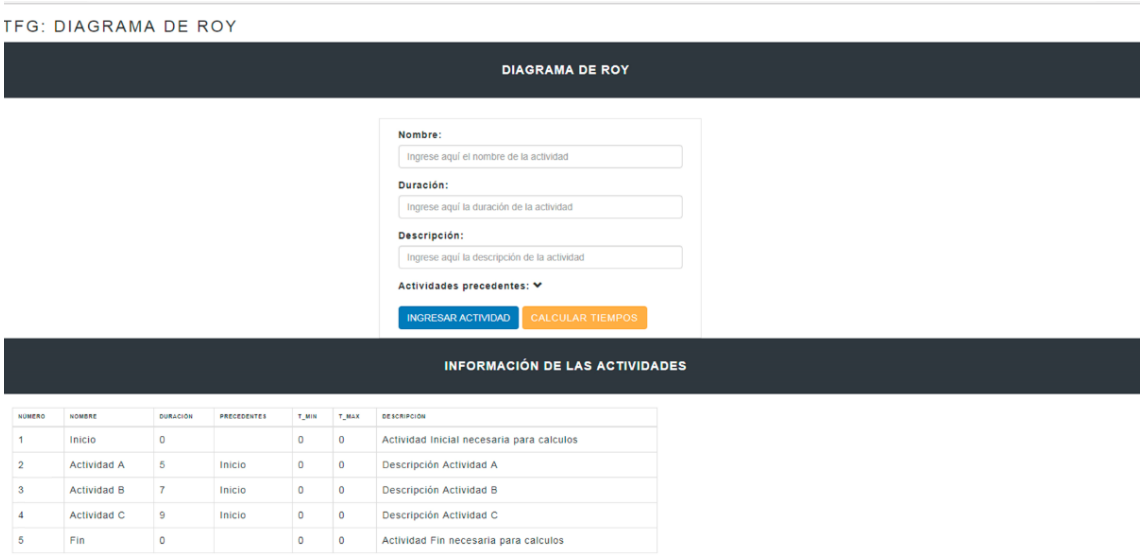
Debajo aparecerá un listado con las actividades Inicio y Fin que se incluyen por defecto en la creación de cualquier proyecto para la realización de los cálculos. A medida que el usuario vaya ingresando actividades estas aparecerán numeradas en el listado con todos los datos actualizados excepto los tiempos mínimo y máximo que se calcularán pulsando el botón ‘CALCULAR TIEMPOS’ una vez ingresadas todas las actividades del proyecto.

En la siguiente imagen se muestra como ingresar una actividad inicial, se rellenan los campos y se marca la casilla Inicio como se ha explicado en el párrafo anterior. En el ejemplo descrito para la explicación del proyecto existen tres actividades iniciales, por tanto, se ingresarán marcando la casilla ‘Inicio’ del desplegable

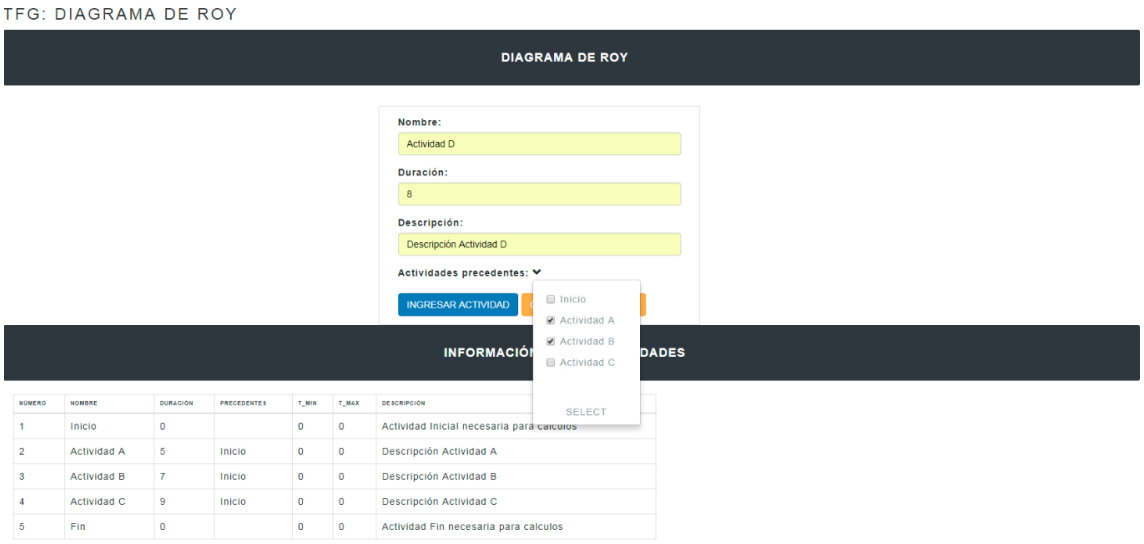


Como podemos observar la Actividad se ha ingresado correctamente y se muestra en el listado con su nombre, duración, actividades precedentes y descripción.

Procedemos de la misma manera para el resto de las actividades y observamos cómo van apareciendo en el listado.



Continuamos añadiendo actividades y marcando sus precedentes mediante el desplegable.



Y van actualizándose en el listado como hemos comentado.

NÚMERO	NOMBRE	DURACIÓN	PRECEDENTES	T_MIN	T_MAX	DESCRIPCIÓN
1	Inicio	0		0	0	Actividad Inicial necesaria para calculos
2	Actividad A	5	Inicio	0	0	Descripción Actividad A
3	Actividad B	7	Inicio	0	0	Descripción Actividad B
4	Actividad C	9	Inicio	0	0	Descripción Actividad C
5	Actividad D	8	Actividad A,Actividad B	0	0	Descripción Actividad D
6	Fin	0		0	0	Actividad Fin necesaria para calculos

En la siguiente imagen observamos que ya tenemos todas las actividades del ejemplo ingresadas.

NÚMERO	NOMBRE	DURACIÓN	PRECEDENTES	T_MIN	T_MAX	DESCRIPCIÓN
1	Inicio	0		0	0	Actividad Inicial necesaria para calculos
2	Actividad A	5	Inicio	0	0	Descripción Actividad A
3	Actividad B	7	Inicio	0	0	Descripción Actividad B
4	Actividad C	9	Inicio	0	0	Descripción Actividad C
5	Actividad D	8	Actividad A,Actividad B	0	0	Descripción Actividad D
6	Actividad E	10	Actividad B	0	0	Descripción Actividad E
7	Actividad F	10	Actividad B,Actividad C	0	0	Descripción Actividad F
8	Actividad G	5	Actividad D	0	0	Descripción Actividad G
9	Actividad H	4	Actividad D,Actividad E	0	0	Descripción Actividad H
10	Actividad I	3	Actividad E,Actividad F	0	0	Descripción Actividad I
11	Actividad J	5	Actividad G,Actividad H	0	0	Descripción Actividad J
12	Fin	0		0	0	Actividad Fin necesaria para calculos

5.2.4 Calcular Tiempos

Una vez ingresadas todas las actividades, aparecerán listadas con sus datos. Procederemos a pulsar el botón ‘CALCULAR TIEMPOS’ y observaremos como aparecen los cálculos del Tiempo Mínimo y Máximo para cada actividad.

NÚMERO	NOMBRE	DURACIÓN	PRECEDENTES	T_MIN	T_MAX	DESCRIPCIÓN
1	Inicio	0		0	0	Actividad Inicial necesaria para calculos
2	Actividad A	5	Inicio	0	3	Descripción Actividad A
3	Actividad B	7	Inicio	0	0	Descripción Actividad B
4	Actividad C	9	Inicio	0	4	Descripción Actividad C
5	Actividad D	8	Actividad A,Actividad B	7	8	Descripción Actividad D
6	Actividad E	10	Actividad B	7	7	Descripción Actividad E
7	Actividad F	10	Actividad B,Actividad C	7	13	Descripción Actividad F
8	Actividad G	5	Actividad D	15	16	Descripción Actividad G
9	Actividad H	4	Actividad D,Actividad E	17	17	Descripción Actividad H
10	Actividad I	3	Actividad E,Actividad F	17	23	Descripción Actividad I
11	Actividad J	5	Actividad G,Actividad H	21	21	Descripción Actividad J
12	Fin	0		26	26	Actividad Fin necesaria para calculos

Capítulo 6
Conclusiones y trabajo futuro

Este proyecto se ha realizado con la intención de conocer la estructura básica de una Aplicación Web Cliente / Servidor, adquirir una base relativamente sólida de los fundamentos de un lenguaje de programación relevante como es Javascript, comprender la integración de las diferentes partes implicadas en el sistema de comunicación y familiarizarse con el resto de los elementos que constituyen un sistema final de este tipo. Se decidió implementar el Diagrama de Roy por considerarse una herramienta útil en la gestión de proyectos y así dotar a la estructura de una funcionalidad atractiva e interesante.

El resultado es una aplicación intuitiva y amigable, de fácil interactividad, por la que puede navegarse con facilidad gracias a una interfaz gráfica simple y directa. El usuario puede crear sencillos proyectos, ingresar actividades y obtener los resultados de las operaciones que se realizan sobre los datos de las actividades ingresadas.

La arquitectura se abordó desde diferentes perspectivas hasta que se decidió que la arquitectura MVC (Modelo-Vista-Controlador) era la más apropiada. Toda la lógica de la aplicación ha quedado debidamente ordenada y estructurada, separada por responsabilidades, de manera que se ha conseguido el objetivo de separar las partes para proteger la estabilidad del sistema.

Como ha quedado patente a lo largo de esta memoria, los objetivos, más modestos en un principio por una razón de desconocimiento de la envergadura final del proyecto, se han cumplido a grandes rasgos, salvo algunos detalles secundarios sobre los que hemos tenido que sortear baches y adaptarnos al tiempo disponible y posibilidades, como el aspecto gráfico o respuestas a errores en la implementación del código.

Hemos comprendido la importancia del orden y la arquitectura en este tipo de proyectos, pues se trabaja con gran cantidad de datos, una sintaxis a menudo ardua, multitud de instrucciones relacionadas, referencias a funciones prediseñadas y desconocidas y muchos otros problemas incómodos que se vuelven cotidianos en esta disciplina.

Hemos descubierto nuevas tecnologías, conocido nuevos lenguajes y nuevas formas de entender el flujo de datos, como el complejo mundo de la programación asíncrona y el uso de funciones callback.

En definitiva, hemos aprendido los fundamentos básicos de la construcción de aplicaciones Web, hemos reforzado y mejorado los humildes conocimientos que poseemos sobre los lenguajes de programación que conocemos, fundamentalmente Javascript, y hemos adquirido una visión general de la compleja relación que mantienen las tecnologías involucradas en la construcción de este tipo de aplicativos.

Ahora que conocemos los principios básicos, nos sentimos capaces de enfrentarnos a proyectos más complejos, resultaría interesante indagar en los recovecos de NodeJS y Express, tener más control sobre los resultados de la interfaz gráfica, conocer Javascript en profundidad, investigar el potencial del lenguaje SQL y adquirir las capacidades para adaptar las tecnologías a nuestro servicio más que adaptarnos nosotros a ellas.

Capítulo 7
Bibliografía

· Gauchat, J.D. (2018). *El gran libro de HTML5, CSS3 y Javascript*.

Barcelona: Marcombo.

· Varios Autores. *Página Oficial de Node.JS*.

<https://nodejs.org/es/>

· Varios Autores. *Página Oficial de Express.JS*.

<https://expressjs.com/es/>

· Varios Autores. *Página Oficial de MySQL*.

<https://www.mysql.com>

· Moisset, D. *Tutoriales Programación Ya*.
<https://www.tutorialesprogramacionya.com/>

· Varios Autores. *Página oficial de W3schools*.

<https://www.w3schools.com/>

· Almunia, P. *Todo JS*.

<https://www.todojs.com/controlar-la-ejecucion-asincrona/>

· Dopacio, C.I. *Diccionario Empresarial*.
<http://diccionarioempresarial.wolterskluwer.es/Content/Documento.aspx?params=H4sIAAAAAAAAEAMtMSbF1jTAAASMTM0tDtbLUouLMDxbIwMDS0NDQ3OQQGZapUt-ckhlQaptWmJOcSoAUtnTMzUAAAA=WKE>