UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA INFORMÁTICA

SISTEMA DISTRIBUIDO DE NOTIFICACIONES PARA DESPACHOS

DISTRIBUTED MESSAGING SYSTEM FOR OFFICES

Realizado por
Carlos Martín Ríos

Tutorizado por
Dra. Ana Cruz Martín
Dr. Juan Antonio Fernández Madrigal

Departamento
Ingeniería de Sistemas y Automática

UNIVERSIDAD DE MÁLAGA

MÁLAGA, OCTUBRE DE 2019

Fecha defensa:

Fdo. El/la Secretario/a del Tribunal

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA

D/Dª.: Carlos Martín Ríos, con DNI 77194859M, estudiante del Grado en Ingeniería Informática de la Universidad de Málaga.

DECLARO QUE:

El Trabajo Fin de Grado denominado:

SISTEMA DISTRIBUIDO DE NOTIFICACIONES PARA DESPACHOS
es de mi autoría, inédito (no ha sido difundido por ningún medio, incluyendo internet) y original (no es copia ni adaptación de otra), no habiendo sido presentado anteriormente por mí ni por ningún otro autor ni en parte ni en su totalidad. Asimismo, se ha desarrollado respetando los derechos intelectuales de terceros, para lo cual se han indicado las citas necesarias en las páginas donde se usan, y sus fuentes originales se han incorporado en la bibliografía. Igualmente se han respetado los derechos de propiedad industrial o intelectual que pudiesen afectar a cualquier institución o empresa.

Para que así conste, firmo la presente declaración en Málaga,
a 20 de SEPTIEMBRE de 2019.

Fdo.: D/Dª CARLOS MARTÍN RÍOS

Resumen

Este trabajo de fin de grado relata el diseño y posterior desarrollo de un dispositivo para la notificación de mensajes en formato electrónico para los despachos de los profesores del sector universitario. Dicho aparato, acoplado en las respectivas puertas de los profesores, mostrará a través de una pequeña pantalla el mensaje que se decida conveniente en el caso de que deba ausentarse de su despacho en hora de tutorial, por ejemplo.

Junto a este dispositivo, se dispondrá de una aplicación de escritorio desde la cual el profesor será capaz de gestionar los mensajes a publicar. Para acceder a esta será necesaria una inscripción previa al sistema.

Además, se ha desarrollado una aplicación de escritorio para la administración de dispositivos y usuarios. No está enfocada a un tipo de personal universitario en específico sino a un grupo reducido de personas que vayan a asumir el cargo de la administración de este. Desde ella se podrá dar de alta y baja a usuarios, despachos y dispositivos y realizar las correctas asignaciones entre ellos.

Palabras clave:

Arduino, Java, WiFi, Distribuido, Notificaciones, Base de datos.

Abstract

This Bachelor Thesis shows the design and subsequent development of an electronic-messaging device for lecturers' offices at university. This gadget will be attached to their respective office doors, and it will display the proper message selected by the lecturer through a small screen in the event of an unexpected absence from the office.

Besides, a desktop application will be available for the professor, giving the ability to manage the to be published in the device. It will be necessary to fill an inscription in order to gain access to the system.

In addition to this, another desktop application has been developed for administration purposes. It is not focused on a certain group of university workers, but on a reduced group of people willing to assume the administration of the system. Through this application we could register and discharge users, offices and devices in addition to do the proper assignations between them.

Keywords:

Arduino, Java, WiFi, Distributed, Messaging, Database.

Índice

Índice de figuras

Índice de tablas

1

Introducción

Este capítulo introductorio recoge los siguientes apartados: la motivación principal por la cual ha sido desarrollado este proyecto, los objetivos que han de ser superados tras el desarrollo de este, la estructura que seguirá la memoria de principio a fin, así como la metodología de trabajo, y, por último pero no menos importante, las herramientas y el software que se ha utilizado para el diseño del proyecto y su posterior implementación.

1.1 Motivación

Una parte importante de la docencia universitaria se corresponde con las tutorías en los despachos de los profesores, los cuales fijan un horario para que los alumnos asistan a ellas. Sin embargo, pueden existir ocasiones en las que un profesor no pueda asistir a la tutoría por causa de fuerza mayor o tenga que abandonar el despacho por un breve periodo de tiempo y necesite informar a sus estudiantes de esta situación.

En algunos casos se dejan notificaciones escritas pegadas en las puertas de los despachos, que en ocasiones pueden despegarse o perderse, resultando en una solución poco elegante. La motivación principal de este trabajo de fin de grado es sustituir este método por uno más refinado y fiable, instalando una pequeña pantalla de LCD en las puertas de los despachos que mostrarán el nombre del profesor y el mensaje a dejar durante un periodo de tiempo, siendo este mensaje eliminado de la pantalla cuando el profesor lo desee. Además, cuando varios profesores compartan un mismo despacho, todos podrán usar la misma pantalla ya que los mensajes irán rotando entre ellos y serán eliminados cuando el profesor decida eliminarlo.

Estas notificaciones electrónicas serán enviadas y gestionadas desde una aplicación cliente en el ordenador que el profesor desee, siendo capaz de monitorizar desde la aplicación los mensajes de varios despachos, en el caso de que tenga un despacho en varios centros, desde la misma cuenta, resultando más fiable para el profesorado.

1.2 Objetivos

Este trabajo de fin de grado consiste en desarrollar un sistema que, a través de una aplicación cliente, un servidor intermedio y una placa Arduino [1] con conexión inalámbrica conectada a una pequeña pantalla LCD, pueda gestionar las notificaciones que se vayan a dejar cuando un profesor o profesores abandonen el despacho durante un cierto periodo de tiempo.

Los objetivos son:

- Diseño y programación de la placa Arduino de tal forma que sea capaz de gestionar la conexión WiFi con el servidor, la pantalla LCD por la que se mostrarán los mensajes, y la publicación de dichos mensajes.

- Diseño y programación de una aplicación cliente de escritorio para gestionar los mensajes que se envíen a un conjunto de placas Arduino. Se dispondrá de un control de acceso al sistema, siendo el administrador del sistema el único que puede introducir profesores a la base de datos y asignarles placas a estos.

- Configuración del sistema gestor de bases de datos y del servidor para manejar las peticiones de conexión entre los profesores y las placas.

1.3 Estructura de la memoria y metodología

Para el desarrollo del proyecto se siguió una metodología de trabajo en cascada [2]. Esta memoria seguirá una estructura lineal según el tiempo de desarrollo del sistema, la cual se dividirá en cinco capítulos, además de contar con tres apéndices a modo de manual para su uso e instalación.

Este primer capítulo sirve como introducción a lo que se pretende desarrollar en este trabajo de fin de grado tal y como se comentó al inicio de este.

En el capítulo dos se relatará la evaluación, elección y posterior montaje de los componentes electrónicos que formarán la parte hardware del sistema.

El tercer capítulo se centra en la evaluación de los requisitos y casos de uso de la placa, el servidor, el cliente y los actores que estarán presentes en el sistema, diseñándose posteriormente las soluciones software según el hardware elegido y de los roles del sistema.

En el capítulo cuatro se explicará el desarrollo del software de cada uno de los componentes que forman el sistema distribuido de notificaciones.

Por otro lado, el quinto capítulo abarcará el despliegue y las pruebas de los componentes desarrollados anteriormente para asegurar el correcto funcionamiento como sistema distribuido.

El capítulo seis se centrará en las conclusiones a las que se han llegado tras el desarrollo de este proyecto y de los trabajos futuros que pueden surgir a partir de él.

Por último, el capítulo siete se compone de la bibliografía que se ha utilizado como referencia a lo largo de todo el trabajo de fin de grado.

Hay que decir que durante las primeras fases del desarrollo del sistema se encontró en los foros de Arduino un sistema de notificaciones para laboratorios [3]. Sin embargo, este tipo de dispositivo no tiene nada que ver con este trabajo de fin de grado puesto que solamente mostraba por pantalla dos mensajes estáticos que cambiaban en función de la posición de un interruptor y tampoco dispone de conexión a Internet.

1.4 Software y tecnologías utilizadas

A la hora de escoger las herramientas y los lenguajes utilizados para la realización del sistema, se ha apostado tanto por software libre como por software propietario. A continuación se exponen las tecnologías usadas en el transcurso del trabajo por orden alfabético:

- **Arduino IDE** [4]

Figura 1.1: Logo de Arduino IDE

El entorno de desarrollo libre y oficial de Arduino se ha utilizado para implementar las funcionalidades de la placa debido a su sencillez a la hora de trabajar con distintas librerías como pueden ser WiFiNINA [5] o LiquidCrystal_I2C [6]. Se ha usado la versión 1.8.9 de este programa.

- **Eclipse IDE** [7]

Figura 1.2: Logo de Eclipse IDE

Otro entorno de desarrollo libre para Java. Utilizado en su versión Photon para implementar las funcionalidades de la aplicación cliente, realizar retoques en el código autogenerado para las distintas vistas de la interfaz y el desarrollar por completo el servidor intermedio.

- **Fritzing** [8][9]

Figura 1.3: Logo de Fritzing

Fritzing es un programa de diseño asistido por ordenador de código abierto destinado a simplificar el diseño esquemas de circuitos electrónicos con microcontroladores y protoboards. Además, cuenta con el apoyo de la web oficial de Arduino, pudiéndose descargar desde ella los esquemas de las placas para Fritzing. Esta herramienta ha sido utilizada para el diseño de las conexiones entre la Arduino y pantalla LCD en su versión BETA 0.9.3.

- **Java** [10]

Figura 1.4: Logo de Java

Ha sido el principal lenguaje de programación usado en este trabajo debido a la gran disponibilidad de librerías que simplifican el desarrollo de elementos, tales como la construcción de interfaces de usuario o implementación de servicios web. La versión que se ha utilizado durante toda la implementación ha sido Java 1.8.

- **Magic Draw UML** [11]

Figura 1.5: Logo de Magic Draw

Magic Draw es una herramienta de ingeniería de software asistido por ordenador con licencia propietaria para realizar diseños UML y los casos de uso de un determinado programa. Se ha usado la versión 19.0 para el diseño de la aplicación y cliente y del servidor intermedio.

- **MySQL** [12]

Figura 1.6: Logo de MySQL

El sistema gestor de bases de datos relaciones de código abierto MySQL es el responsable de la base de datos que el sistema utiliza para almacenar todos los datos de los usuarios y placas dados de alta en el sistema. Se ha utilizado la versión 8.0.17.0.

- **NetBeans IDE** [13]

Figura 1.7: Logo de NetBeans IDE

Entorno de desarrollo libre para Java. Usado en su versión 11.0 para el diseño de las interfaces de las aplicaciones cliente de usuario y administrador, debido a que dispone de un marco de trabajo que simplifica mucho el desarrollo de la parte visual de estas aplicaciones. Sin embargo, no permite la modificación manual del código generado automáticamente por la aplicación para la vista.

- **ORACLE Datamodeler** [14]

Figura 1.8: Logo de ORACLE Datamodeler

Programa de diseño de modelos entidad-relación para bases de datos relacionales. Permite además generar un archivo de órdenes con el código de la base de datos. Usado en su versión 18.3.0 para el modelo de la respectiva base de datos del sistema y de su respectivo código.

2

Elección y montaje de componentes hardware

Este capítulo expone una parte fundamental de este trabajo de fin de grado: la búsqueda, elección y posterior montaje de los componentes electrónicos que formarán la parte tangible del sistema.

2.1 Elección del hardware

A la hora de elegir los componentes electrónicos que formarían parte del hardware del sistema se tuvieron en cuenta dos elementos principales: la capacidad de cómputo y el tamaño de la memoria. Estos elementos son importantes ya que la placa ha de gestionar los mensajes en circulación y escuchar peticiones de clientes, evitando en lo posible que la memoria de la placa se desborde. Asimismo, la conectividad a Internet vía WiFi integrado también es un elemento muy a tener en cuenta.

Se han considerado las siguientes placas Arduino para la implementación del servidor empotrado en ella. En la tabla 2.1 se muestran los parámetros principales:

MODELO	***VOLTAJE***	***CPU***	***EEPROM***	***SRAM***	***FLASH***	***CONECTIVIDAD***
UNO [15]	5V	16MHz	1KB	2KB	32KB	No
ZERO [16]	3.3V	48MHz	-	32KB	256KB	No
MEGA 2560 [17]	5V	16MHz	4KB	8KB	256KB	No
MKR 1000 [18]	3.3V	48MHz	-	32KB	256KB	WiFi Integrado
MKR 1010 [19]	3.3V	48MHz	-	32KB	256KB	WiFi Integrado

Tabla 2.1: Comparativa de las distintas placas candidatas

De entre todas las placas preseleccionadas de la tabla 2.1, se eligió la Arduino MKR 1010 WiFi debido a su capacidad de cómputo y memoria, además de incorporar un módulo integrado WiFi ESP32 el cual nos ahorrará tener que buscar un módulo WiFi externo en caso de haber elegido otra placa diferente de la familia MKR.

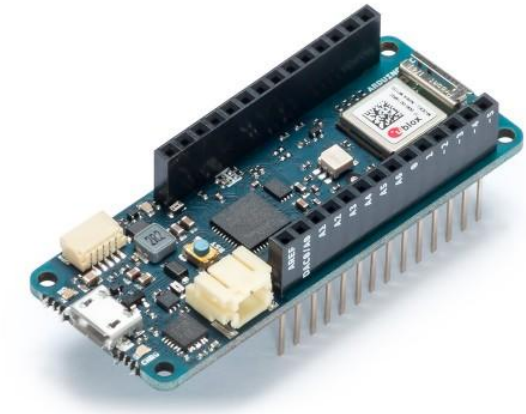

Figura2.1: Arduino MKR 1010 WiFi

Respecto a la elección de la pantalla que será conectada a nuestra placa, se apostó desde un principio por una simple con 16 columnas y 2 filas de resolución debido a que en la primera fila se mostrará el nombre del usuario y en la segunda fila el mensaje, por lo que la búsqueda de una pantalla más compleja supondría un desperdicio de recursos.

El parámetro principal que se tuvo en cuenta a la hora de elegir la pantalla fue la minimización del cableado entre la Arduino y esta, resultando en un componente simple de montar. En la tabla 2.2 se muestran los parámetros principales:

MODELO	***CHIPSET***	***CONEXIONES A CABLEAR***
LCD 16X02 [20]	HD44780	16
LCD 16X02 I2C [21]	HD44780	4

Tabla 2.2: Comparativa de las dos pantallas candidatas

Figura 2.2: Pantalla LCD 16x02 con módulo I2C presoldado

Tras haber visto las dos elecciones disponibles se decidió elegir la pantalla LCD con un módulo I2C [22] presoldado a ella, la cual nos permitirá realizar la conexión entre la placa y la pantalla de forma más sencilla debido a que solamente tendremos que conectar cuatro cables. Hay que remarcar que en la tabla 2.2 no se ha tenido en cuenta el fabricante, puesto que este tipo de pantallas son fabricadas por una gran cantidad de empresas y las características más importantes son su chipset y su resolución.

Una vez realizada la elección de los componentes electrónicos, podemos pasar al posterior montaje de ellos en lo que se convertirá en la parte visual del sistema.

2.2 Montaje del hardware

Para la conexión entre la placa y la pantalla necesitaremos cuatro cables con un extremo macho y otro hembra. Tendremos una pareja de cables rojos, una de cables negros, otra de cables amarillos y una última de cables verdes. Con este tipo de montaje lo que queremos evitar es que los cables se confundan entre ellos y nos quede un montaje sencillo de ver y seguro de manipular.

Figura 2.3: Cables con extremos macho-hembra

La Arduino MKR 1010 WiFi dispone de un módulo I2C integrado el dispone de los conectores SCL y SDA. Puesto que la nuestra pantalla LCD 16x02 incorpora otro módulo I2C, también tiene los conectores SCL y SDA, por lo que simplemente hemos de conectar cada conector con su homólogo.

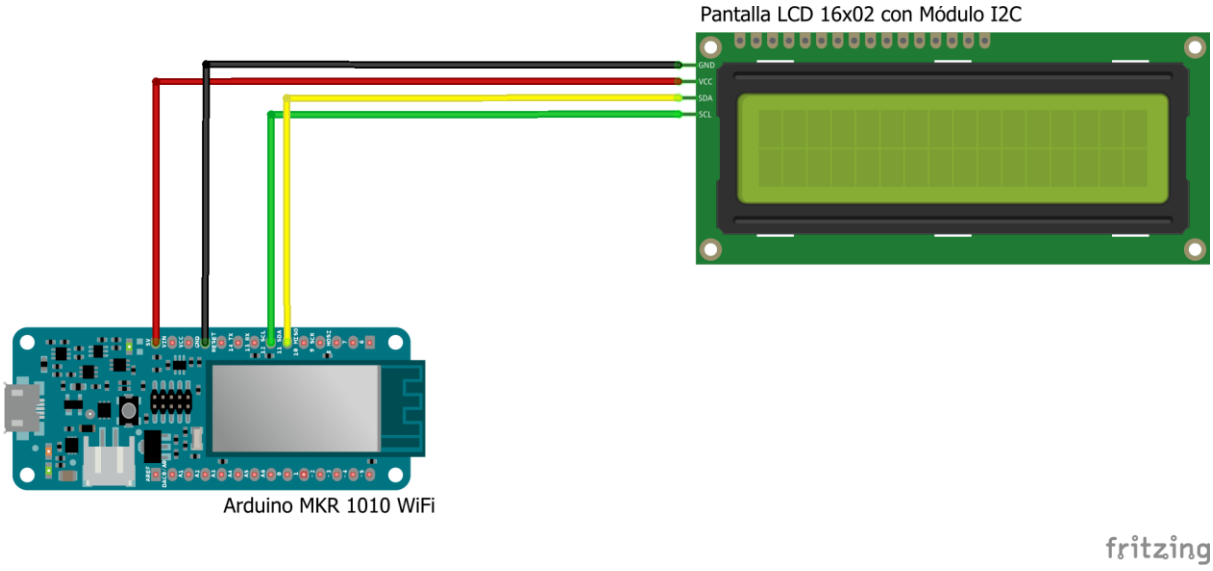

Figura 2.4: Esquema de conexión placa/pantalla en Fritzing

Como puede apreciarse en la figura 2.4, el cable de alimentación para la pantalla será un cable rojo proveniente del conector de 5 voltios de la placa que irá al conector VCC de la placa. El cable negro se corresponde con tierra, este conectará las tierras de la placa y la pantalla. Por último, el cable amarillo se corresponde con los conectores SDA y el cable verde con los conectores SCL. Para alimentar la Arduino usaremos un cable de tipo MicroUSB que posteriormente irá conectado a la red eléctrica.

Los conectores macho de cada uno de los cables irán conectados a su respectivo pin en la Arduino MKR 1010 WiFi, mientras que los extremos con la toma hembra estarán conectados en sus correspondientes pines del módulo I2C de la pantalla LCD.

Hay que remarcar que en la figura 2.4, el esquema de la placa se corresponde con una Arduino MKR 1000 en lugar de una Arduino MKR 1010 WiFi. Esto se debe a que Arduino aún no ha lanzado una versión del esquema de esta placa, estando disponible solamente el de la MKR 1000. Sin embargo, esto no supone un problema puesto que la disposición de los pines en sendas placas es idéntico, de ahí que la MKR 1000 se llame MKR 1010 WiFi en el esquema.

El presupuesto total que se ha utilizado en adquirir los diferentes componentes ha sido de 23.90€, el desglose de este podemos verlo en la tabla 2.3:

COMPONENTE	***PRECIO***
Arduino MKR 1010 WiFi	27.90€
Pantalla LCD 16x02 con módulo I2C	5€
Cableado	1€
TOTAL	***23.90€***

Tabla 2.3: Precio de cada uno de los componentes

Los componentes han sido comprados tanto en tienda online como en tienda física, por lo que el presupuesto puede variar en un futuro o según el vendedor.

En la figura 2.5 se muestra el ensamblaje de ambos componentes en físico. Hay que mencionar que la Arduino está montada sobre una protoboard a modo de soporte, ya que la placa tiene pines al aire libre en su parte inferior como se pudo ver en la figura 2.1, además, esta dispone de una parte inferior con cinta adherente que será utilizada para pegar el dispositivo a la puerta del despacho.

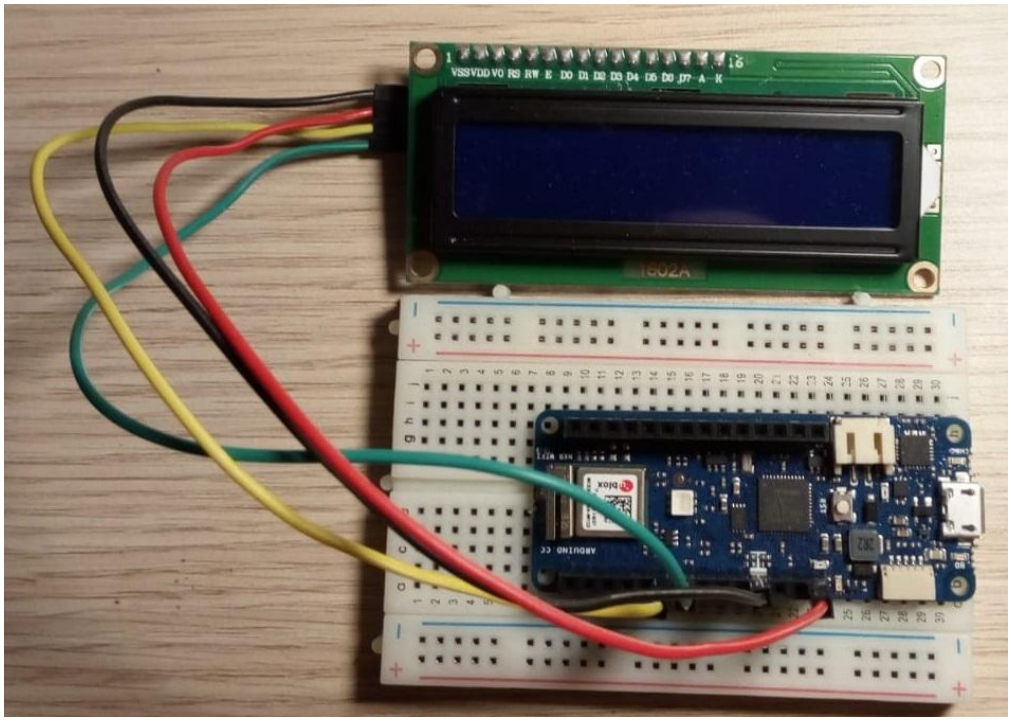

Figura 2.5: Ensamblaje de componentes

3

Diseño del sistema

Este capítulo se centrará en el diseño de cada uno de los componentes software que formarán el sistema distribuido de notificaciones electrónicas en función de los casos de uso [23] de este.

3.1 Casos de uso

A la hora de diseñar nuestro sistema tenemos que tener muy en cuenta los actores o roles que estarán presentes, ya que éstos serán los que le den uso y por tanto hemos de diseñar las distintas funcionalidades que pueda tener cada uno de ellos.

Como bien se menciona en el capítulo introductorio, este sistema dispondrá de dos roles principales: el usuario y el administrador. A continuación en las figuras 3.1 y 3.2 podemos ver los casos de uso de cada uno.

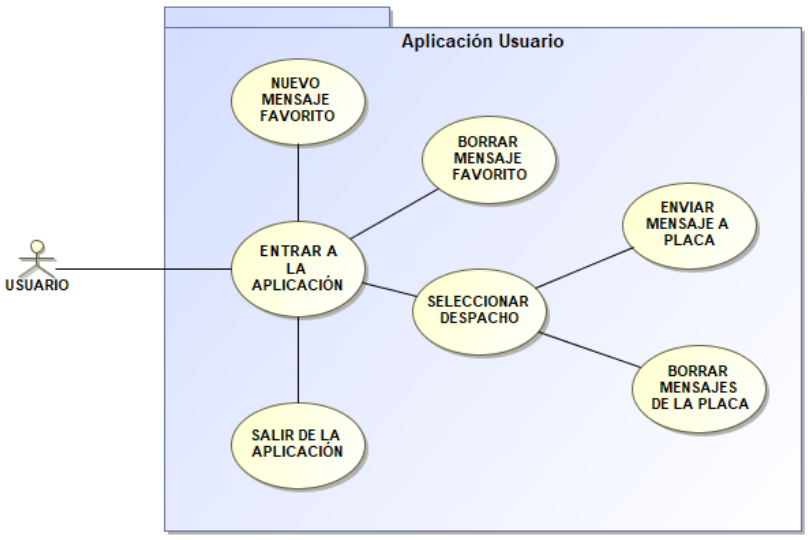

Figura 3.1: Casos de uso del usuario

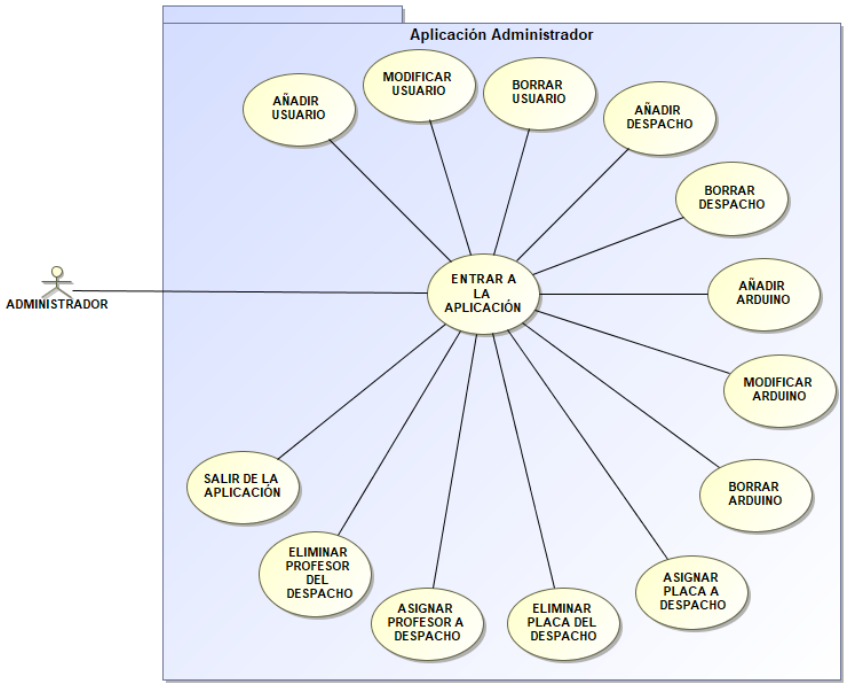

Figura 3.2: Casos de uso del administrador

Como bien se puede apreciar, cada uno de ellos tendrá una aplicación de cliente propia que será instalada en los respectivos ordenadores del usuario y administrador. En la figura 3.3 podemos ver una visión general del sistema en forma de diagrama de secuencia.

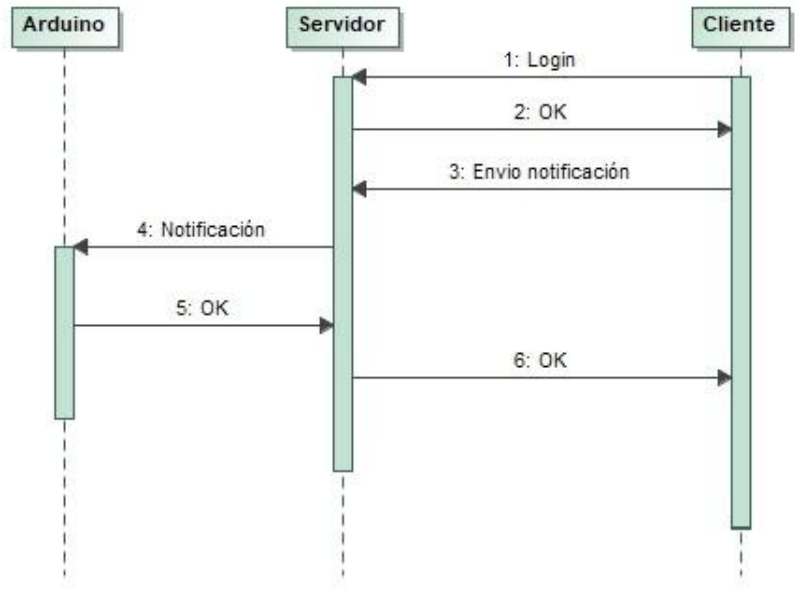

Figura 3.3: Visión general del paso de mensajes entre el cliente, el servidor y la placa

3.2 Base de datos

Antes de realizar cualquier tipo de diseño de la base de datos, debemos analizar qué entidades se van a almacenar en esta en forma de tablas, los atributos que dichas entidades que formarán las columnas de las tablas, y, por último, las relaciones que existirán entre dichas entidades. Para ello procederemos a realizar un análisis de requisitos funcionales y no funcionales de la base de datos.

- **Requisitos funcionales**
  - La base datos ha de ser capaz de almacenar los datos de los profesores dados de alta en el sistema y los mensajes favoritos de cada uno de estos.

  - También ha de guardar las placas Arduino que formen parte del sistema juntos con los despachos a los que estas estén asignadas.

  - Cada despacho deberá tener una única placa asignado a él, sin embargo, un mismo despacho podrá tener varios profesores o un mismo profesor podrá tener varios despachos asignados.

- **Requisitos no funcionales**
  - Como medida de seguridad adicional, se deberá de almacenar el valor resultante de aplicar una función de tipo hash a la contraseña del usuario

  - Para controlar las acciones que se llevan a cabo dentro de la base de datos será necesario disponer de una tabla de auditoría donde se almacenaran estas acciones.

Con estos requisitos podremos generar el modelo entidad-relación [24] correspondiente a la base de datos necesaria para almacenar y gestionar los usuarios, placas y despachos que luego serán accedidas por las aplicaciones cliente. A continuación, en las figuras 3.4 y 3.5 podemos ver con detalle este modelo y su versión lógica.

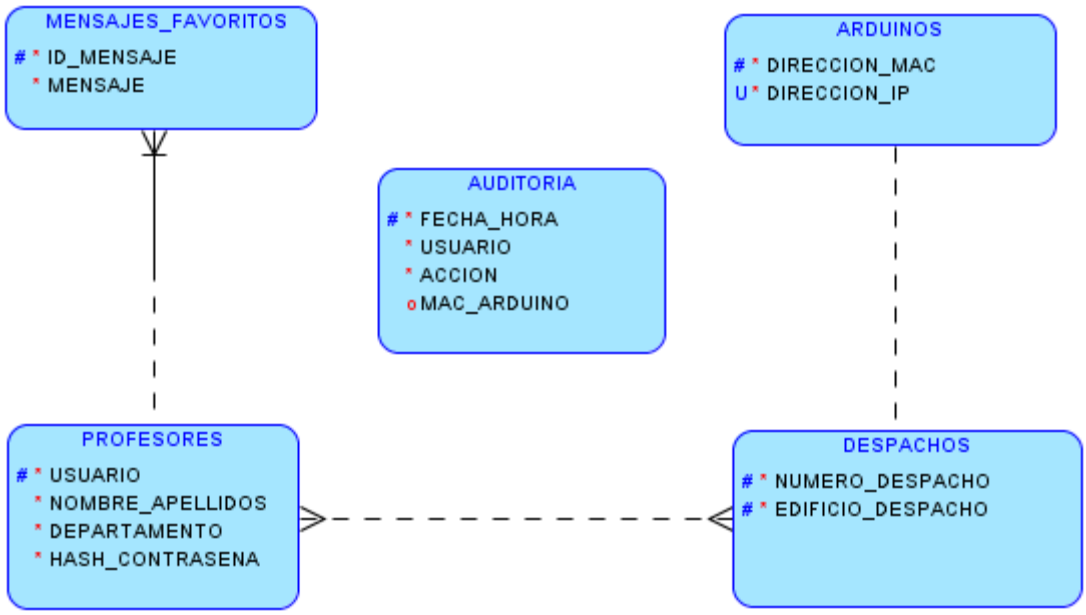

Figura 3.4: Modelo entidad-relación de la base de datos del sistema

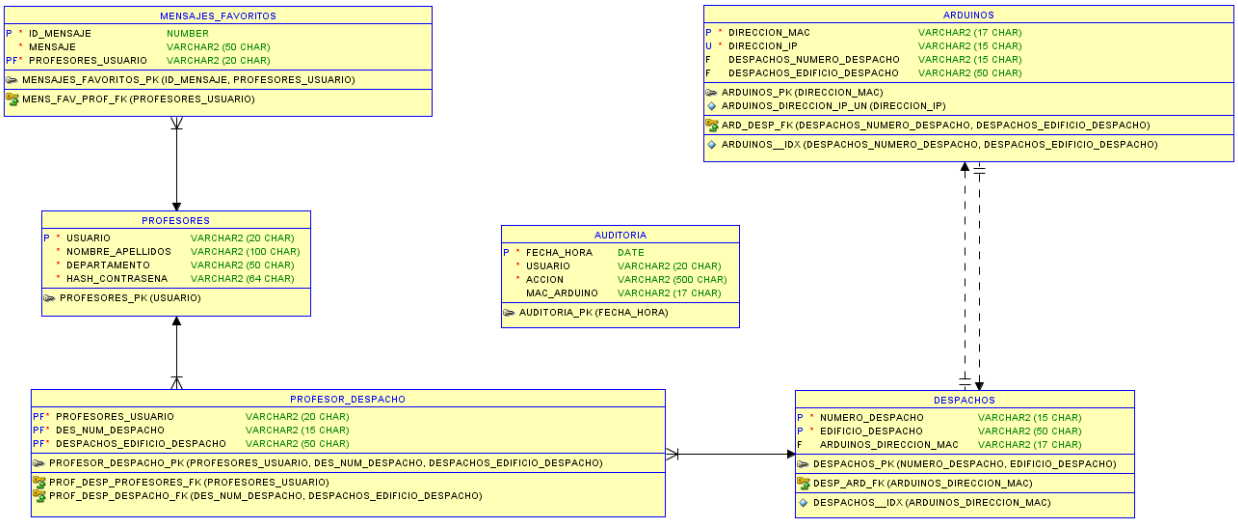

Figura 3.5: Modelo lógico de la base de datos del sistema

A continuación, se procederá a explicar cada una de las entidades y relaciones existentes en los modelos de las figuras anteriores.

- **Entidades**
  - *MENSAJES_FAVORITOS*: Aquí almacenaremos un código único como la clave primaria del mensaje y el contenido del mismo. El mensaje no puede estar vacío.

  - *PROFESORES*: Esta entidad almacenará el nombre de usuario, que será la clave primaria de la entidad, seguida del nombre y apellidos del profesor junto con el departamento al que pertenece y el valor hash de su contraseña.

  - *DESPACHOS*: Se guardarán el número del despacho y el edificio al que pertenece. Estos valores formarán la clave primaria de la tabla. El propósito fundamental de esta es normalizar la base de datos para que a la hora de modificar el despacho de una Arduino, no haya que modificar todos los despachos de los profesores.

  - *ARDUINOS*: Almacenará tanto la dirección MAC, que será la clave primaria de la tabla, como la dirección IP estática de la placa, que deberá ser única.

  - *AUDITORÍA*: Guardará toda la información que pase por el servidor intermedio. Se registrará la fecha y hora del suceso junto con el usuario que la llevó a cabo y la acción que realizó.

- **Relaciones**
  - *MENSAJES_FAVORITOS-PROFESOR (Entidad débil)*: Cada profesor deberá tener asociado únicamente a él una serie de mensajes favoritos.

  - *PROFESORES-DESPACHOS (Muchos a muchos)*: Como se menciono en los requisitos funcionales, un profesor puede disponer de varios despachos (situados en diferentes edificios) y a su vez un despacho puede albergar a varios profesores.

  - *DESPACHOS-ARDUINO (Uno a Uno)*: Las placas dadas de alta en el sistema pueden estar acopladas en un único despacho o estar sin asignación.

3.3 Servidor Intermedio Java

Este componente fundamental del sistema tiene como misión ser el intermediario entre las placas Arduino dadas de alta en la base de datos y asignadas a un despacho y los usuarios que tengan acceso a ellas. Para ello, el servidor se ejecutará en una dirección IP y puerto fijo, escuchando todas las peticiones que le lleguen y tengan el formato adecuado para su posterior trámite entre la placa y el usuario. A partir de aquí, nos referiremos a este como servidor intermedio.

Atendiendo a uno de los requisitos básicos que debe de tener todo servidor es el ser concurrente, esto es, ser capaz de gestionar múltiples conexiones de usuarios o administradores en el tiempo sin que éstas se bloqueen entre sí, por ello será necesaria la utilización de hebras de ejecución siendo cada hebra una conexión única.

Sin embargo, el servidor no sólo ha de atender a las diferentes conexiones de los usuarios del sistema, sino que también ha de crear conexiones con las placas Arduino que dichos usuarios les envíen vía protocolo interno. Por tanto, se deberá implementar en Java una clase principal con funciones y métodos para crear hilos y conexiones con la placa, además de disponer de una clase para crear la conexión con la base de datos que hemos diseñado en el capítulo anterior. En la figura 3.6 podemos ver una visión general del diagrama de clases sin entrar muy a fondo en detalle de variables.

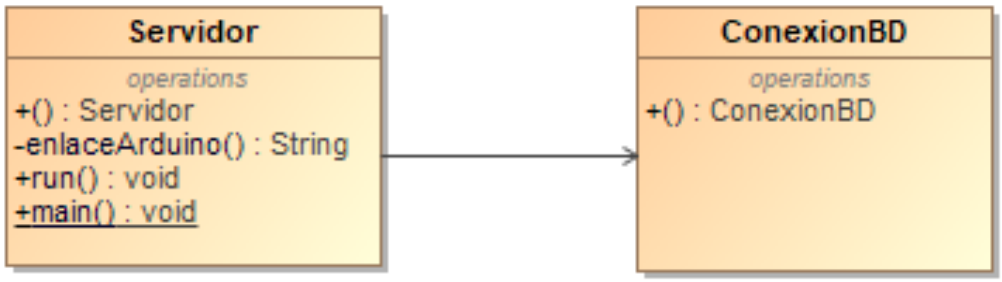

Figura 3.6: Diagrama de clases del servidor intermedio

3.4 Servidor Arduino

Como ya se comentó anteriormente en la introducción de este trabajo de fin de grado, el servidor empotrado en la placa Arduino ha de ser capaz de escuchar peticiones de envíos de mensajes por parte de los usuarios y mostrar dichos mensajes por la pantalla LCD de forma simultánea.

Debido a que los mensajes han de estar mostrándose por la pantalla durante una pequeña cantidad de tiempo, si seguimos el modelo de programación tradicional basado en un bucle infinito, nuestra solución no será efectiva puesto que durante el tiempo que el mensaje esté en pantalla, el servidor de la Arduino no será capaz de recibir nuevas peticiones debido a que la pantalla estará realizando una espera activa para que pueda mostrarse dicho mensaje.

Para solucionar este problema se utilizará un modelo de programación basado en un bucle ejecutivo cíclico [25][26], cuyo funcionamiento podremos ver en la figura 3.7. Este modelo de programación permite realizar una multitarea en tiempo real gobernado por un bucle de ciclos que se reinician una vez alcancen un límite preestablecido.

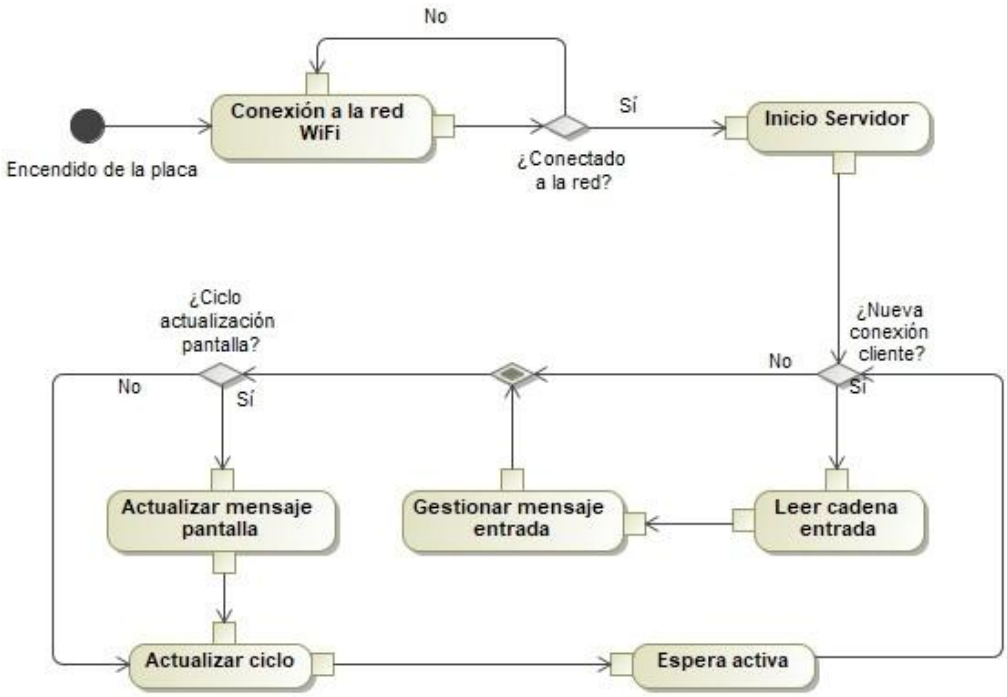

Figura 3.7: Diagrama de flujo del bucle en ejecutivo cíclico de la placa

3.5 Aplicación cliente

Como hemos podido ver en las figuras 3.1 y 3.2, los casos de uso del sistema presentan a dos actores distintos: usuario básico, que en este caso sería un profesor, y usuario administrador. Desde un primer momento se estableció que estas serían aplicaciones de escritorio, ya que el profesor utilizará la aplicación principalmente cuando esté frente a su ordenador personal del despacho y tenga que abandonarlo por algún motivo. En cualquier otro caso, también se podrá utilizar la aplicación desde cualquier ordenador personal con conexión a Internet, por ejemplo, se puede utilizar desde casa si el profesor ve que va a llegar tarde a tutoría.

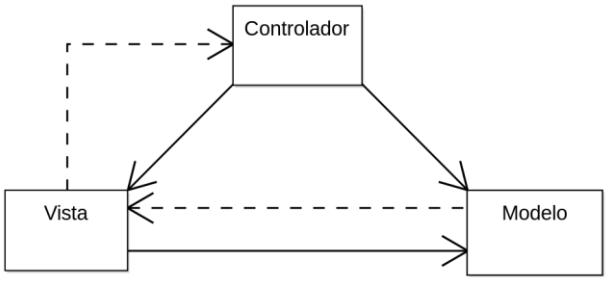

Figura 3.8: Patrón modelo-vista-controlador [27][28]

Ambas aplicaciones seguirán el patrón de diseño modelo-vista-controlador observable en la figura 3.8, el cual nos permitirá tener mejor estructuradas cada una de las clases que constituirán sendas aplicaciones que usuario. Podemos ver el esquema general del patrón MVC de ambas a continuación en la figura 3.9. En el caso de este sistema, el modelo estará situado en el servidor, mientras que la vista y el controlador estarán la aplicación cliente.

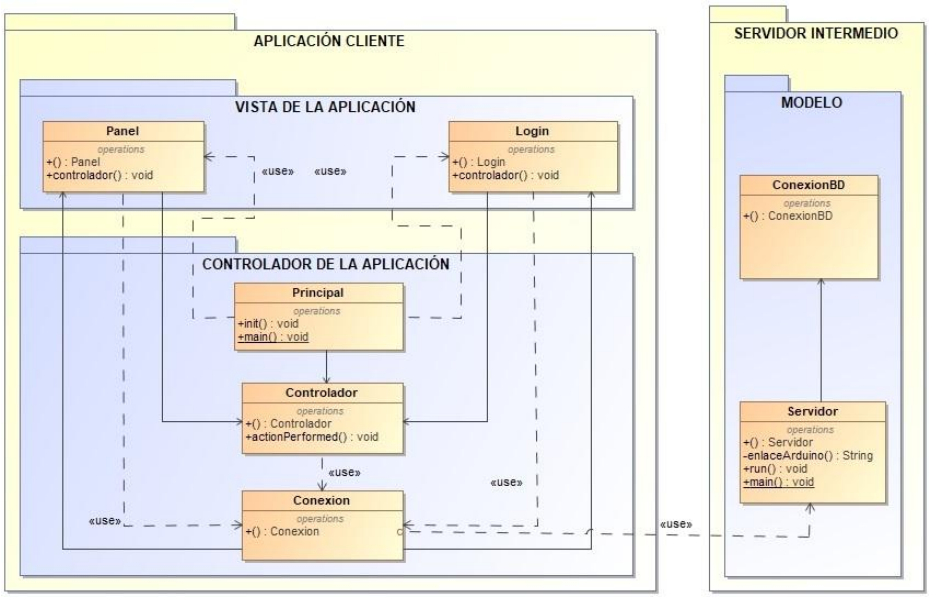

Figura 3.9: Modelo vista controlador de ambas aplicaciones cliente

3.5.1 Profesor

La aplicación del profesor ha de ser capaz de gestionar los mensajes favoritos que éste tenga guardados en la base datos, tanto para añadir nuevos mensajes como borrar mensajes ya existentes. Por otro lado, ha de recuperar de la base de datos todos sus despachos, ya que desde aquí se gestionará el envío de mensajes a las placas.

En base a los casos de uso que se mencionaron en el parrado anterior y en el apartado 3.1, se diseñó la interfaz de la aplicación, en NetBeans gracias a la funcionalidad de construcción de interfaces graficas que la aplicación tiene disponible. En la figura 3.10 podemos observar la interfaz gráfica de esta:

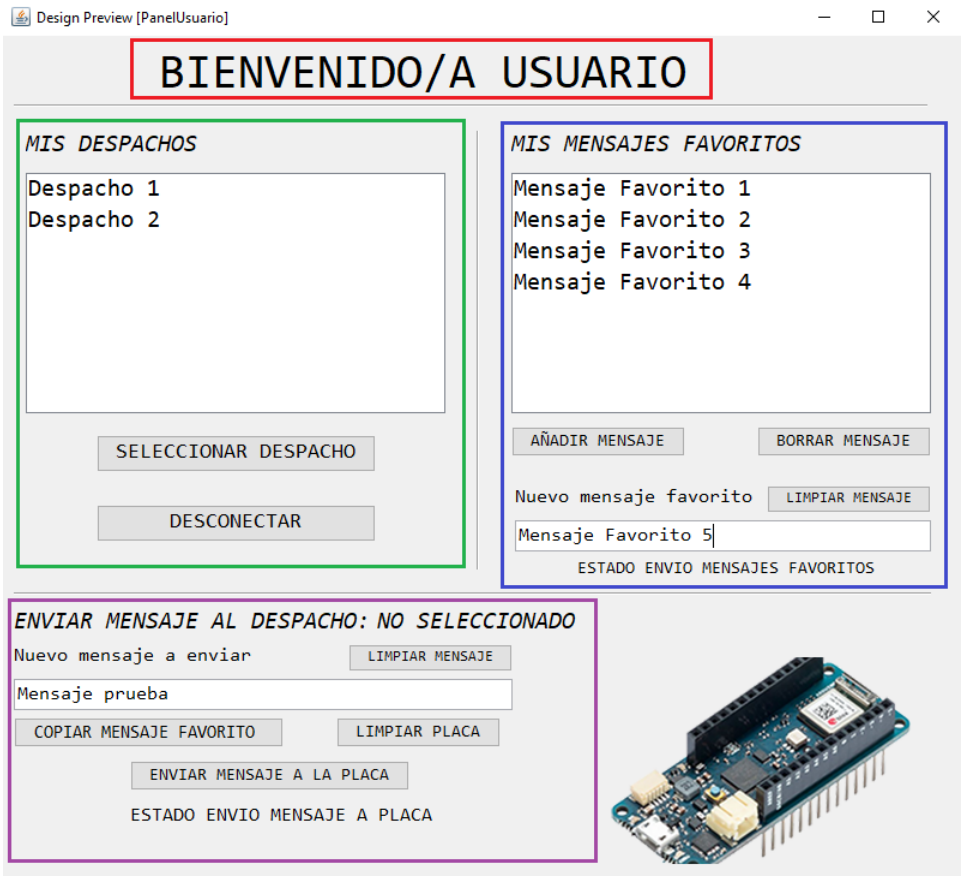

Figura 3.10: Interfaz de la aplicación del usuario

Como bien se puede apreciar en la figura anterior, este interfaz gráfica tiene la disposición de cuadro de mando, teniendo todas las funcionalidades en una única ventana pero cada una de ellas bien separadas y distinguibles las unas de las otras.

En el rectángulo rojo podemos ver el mensaje de bienvenida, el cual cambiará en función del usuario que se conecta la aplicación. Dentro del rectángulo verde de la aplicación tenemos la selección de despachos, está nos mostrará los despachos que tengamos asignados y dispondrá de dos botones para seleccionar el despacho o desconectar nuestra sección del aplicación. El rectángulo azul contiene la sección de mensajes favoritos, desde esta podremos añadir o borrar mensajes favoritos. Por último, en la zona marcada en morado tenemos el corazón del aplicación, el envío de mensajes. En esta podremos gestionar el envío de mensajes nuevos o ya existentes a la placa del despacho que hayamos seleccionado, además de poder borrar el contenido de esta con un solo clic en el botón de limpiar placa.

3.5.2 Administrador

Por otra parte, la aplicación de administración debe poder gestionar los profesores, las placas, los despachos y las distintas asignaciones entre ellos, es decir ha de ser capaz de asociar a los profesores con los despachos y a los despachos con las placas. De esta forma obtendremos una conexión entre profesores y placas.

Siguiendo una línea común al apartado anterior, la interfaz gráfica de la aplicación de administración también se presenta como un cuadro de mando, siendo este mucho mayor tamaño debido a la gran cantidad de funcionalidades que tiene, estando cada una de las zonas de administración separadas de forma que sea fácil de localizar cada función. A continuación en la figura 3.11 podemos ver la interfaz gráfica:

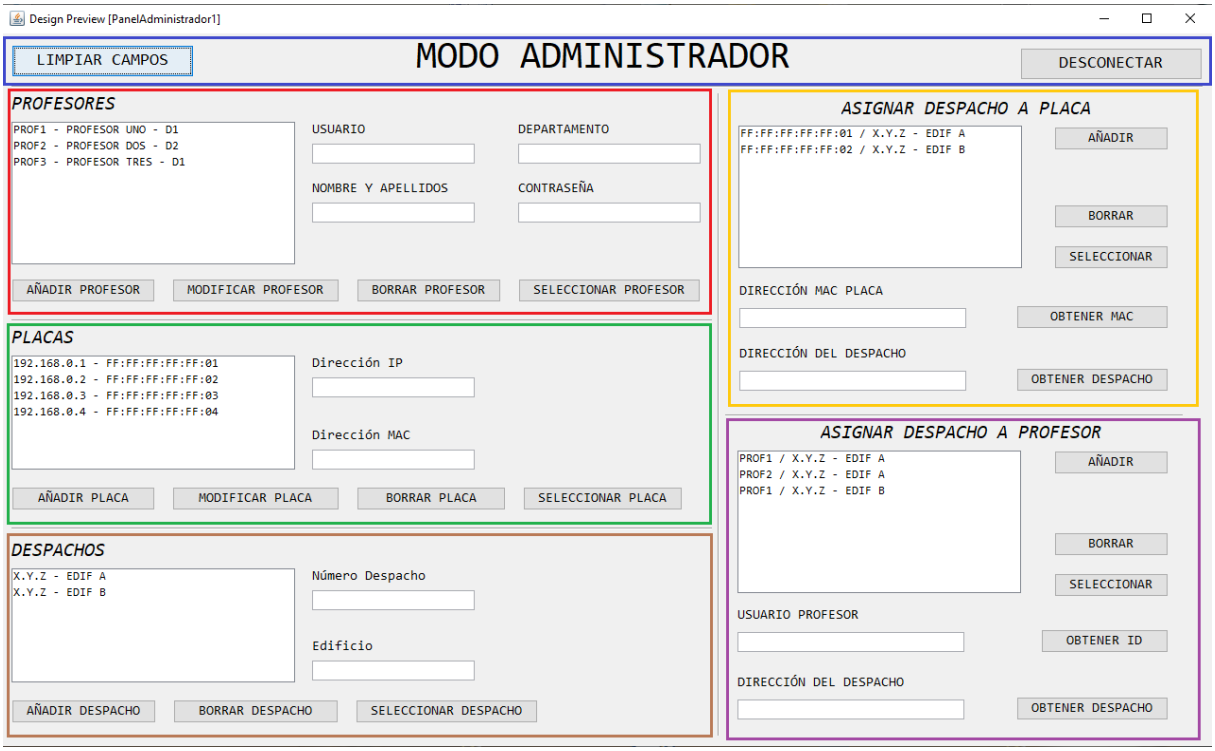

Figura 3.11: Interfaz de la aplicación de administración

Podemos observar cinco zonas claramente definidas por separadores: la zona de administración de profesores arriba marcada por un rectángulo rojo; la zona de gestión de placas señalada en verde; la sección de despachos en color marrón; la zona de asignaciones de despachos con placas marcada en amarillo; y por último la sección designaciones de despachos con profesores resaltada por un rectángulo morado. Cada una de estas secciones dispone de sus botones propios para realizar las gestiones pertinentes. También se puede observar que en la parte superior izquierda y derecha, marcadas en azul, tenemos los botones de limpiar campos y de desconexión del sistema, respectivamente.

4

Desarrollo del sistema

Este capítulo se centrará en el desarrollo de cada uno de los componentes del sistema siguiendo el mismo orden que en el capítulo anterior, es decir, en primer lugar se explicará el protocolo de comunicaciones del sistema y posteriormente el desarrollo del software implantado en el servidor intermedio, la placa Arduino, y las aplicaciones del profesor y el administrador para poder manejar la información de dicho protocolo.

4.1 Protocolo de comunicación

Tras la visión general del esquema del sistema en la figura 4.1 y de los casos de uso descritos en el capítulo anterior, es necesario implementar un protocolo de comunicación propio para la gestión de las peticiones del cliente y las respuestas del servidor. Tanto el usuario como el administrador han de poder comunicarse con el servidor Arduino de forma transparente.

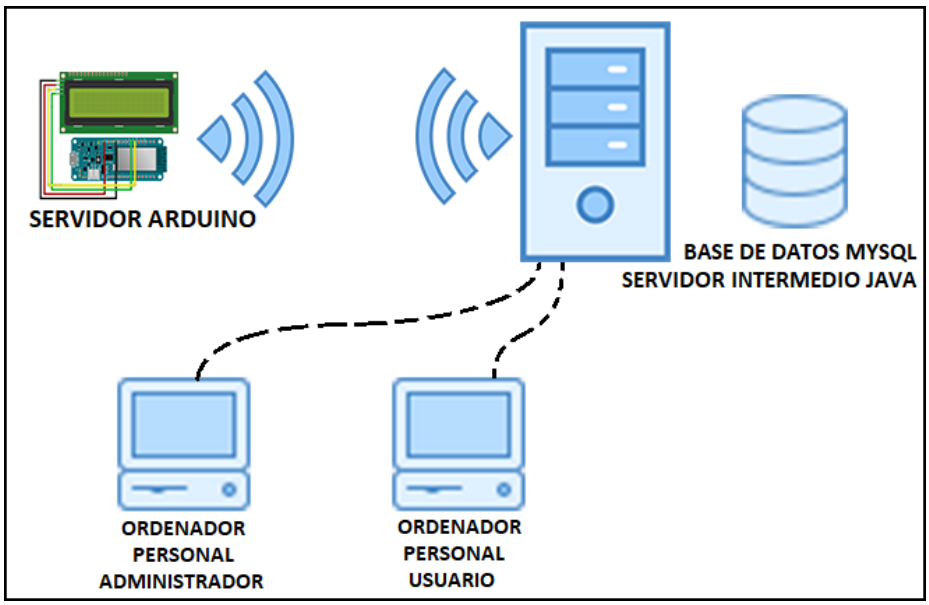

Figura 4.1: Vista general del modelo del sistema

Debido a la existencia de dos aplicaciones cliente, una para el usuario básico y otra para el usuario administrador, tenemos comandos comunes y comandos propios de cada uno. Hay que remarcar que los comandos del protocolo de comunicación del sistema se componen de una cadena de caracteres con separadores especiales para así identificar las partes de este. Esta cadena, una vez llegue al servidor intermedio será troceada en varias partes que luego serán procesadas por las distintos métodos que este tendrá implementados. Para ello se dispone del separador **<-*->** para dividir la cabecera y el cuerpo; y del separador **<*>** para dividir los datos del cuerpo

En total, el sistema dispondrá de veinte comandos distintos: 2 comandos comunes para el usuario y el administrador, 4 comandos específicos para el usuario, y 14 comandos solo para el administrador, según se muestra en las tablas 4.1, 4.2 y 4.3, respectivamente.

CABECERA	***CUERPO***	***FUNCIÓN***
LOGIN<-*->	usuario<*>contraseña<*>servidor:puerto	Entrar al sistema
LOGOUT<-*->	usuario	Salir del sistema

Tabla 4.1: Comandos comunes de las aplicaciones cliente

CABECERA	***CUERPO***	***FUNCIÓN***
NEWMSG<-*->	usuario<*>mensaje	Nuevo favorito
DELMSG<-*->	usuario<*>mensaje	Eliminar favorito
GETDESP<-*->	cp_despacho	Obtener la IP del despacho
SEND<-*->	usuario<*>ip_placa<*>puerto_placa<*>usuario:mensaje	Envío mensaje a la placa

Tabla 4.2: Comandos propios de la aplicación cliente del profesor

CABECERA	***CUERPO***	***FUNCIÓN***
NEWUSER<-*->	usuario<*>contraseña<*>nombre_apellidos<*>departamento	Nuevo usuario en BD
MODUSER<-*->	usuario<*>contraseña<*>nombre_apellidos<*>departamento	Modificar usuario BD
DELUSER<-*->	usuario	Borrar usuario de BD
NEWARD<-*->	direccion_mac<*>direccion_ip	Nueva placa en BD
MODARD<-*->	direccion_mac<*>direccion_ip	Modificar placa BD
DELARD<-*->	direccion_mac	Borrar placa de BD
NEWDESP<-*->	numero_despacho<*>edificio_despacho	Nuevo despacho BD
DELDESP<-*->	numero_despacho<*>edificio_despacho	Borrar despacho BD
ASGNDESPARD<-*->	direccion_mac<*>cp_despacho	Unir Despacho-Placa
MODDESPARD<-*->	direccion_mac<*>cp_despacho	Modificar Desp-Placa
DELDESPARD<-*->	direccion_mac<*>cp_despacho	Borrar Desp-Placa
ASGNDESPUSER<-*->	usuario<*>cp_despacho	Unir Despacho-Prof
MODDESPUSER<-*->	usuario<*>cp_despacho	Modificar Desp-Prof
DELDESPUSER<-*->	usuario<*>cp_despacho	Borrar Desp-Prof

Tabla 4.3: Comandos propios de la aplicación cliente del administrador

Cada uno de estos comandos será generado por el respectivo Controlador de las aplicaciones cliente en función de los botones que se hayan seleccionado y de los parámetros introducidos en cada uno de los campos de texto, un ejemplo visual de esto se muestra en la figura 4.2:

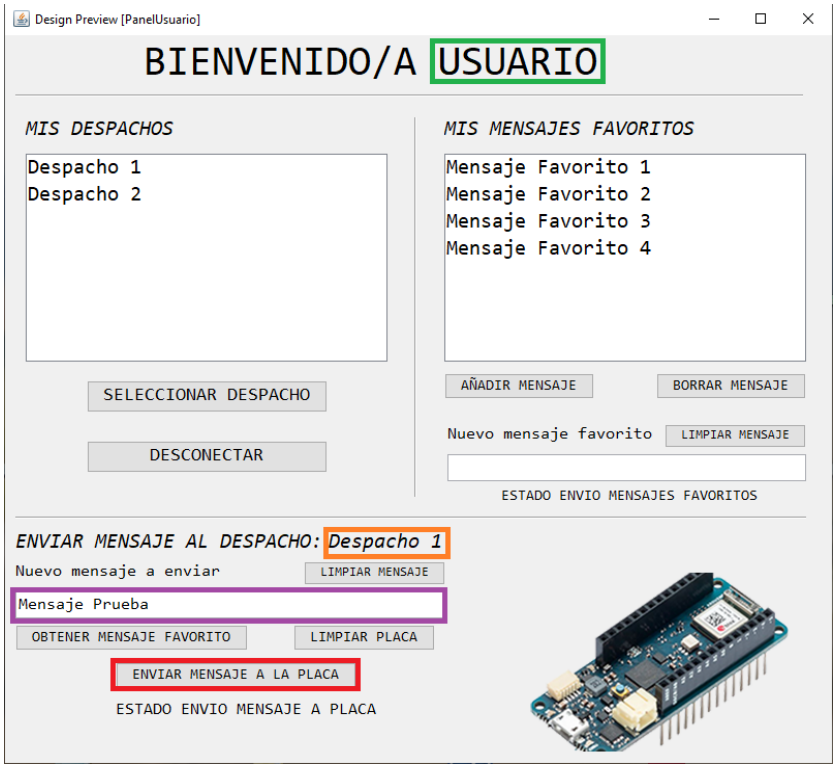

Figura 4.2: Selección de parámetros para generar un comando

Dicha selección de parámetros y botón generará el siguiente comando interno marcados según el color del área seleccionada:

SEND<-*->USUARIO<*>IP_DESPACHO_1<*>PUERTO<*>USUARIO:MENSAJE PRUEBA

4.2 Base de datos

Para implementar la base de datos se usó MySQL Workbench, un entorno de trabajo que forma parte de la instalación de MySQL. Dentro de este, se procedió a crear una conexión y un esquema en el cual se guardarían las tablas y sus datos como se aprecia en la figura 4.3. Este procedimiento puede verse con mayor claridad en el apéndice A.3 junto con el proceso de instalación de la base de datos.

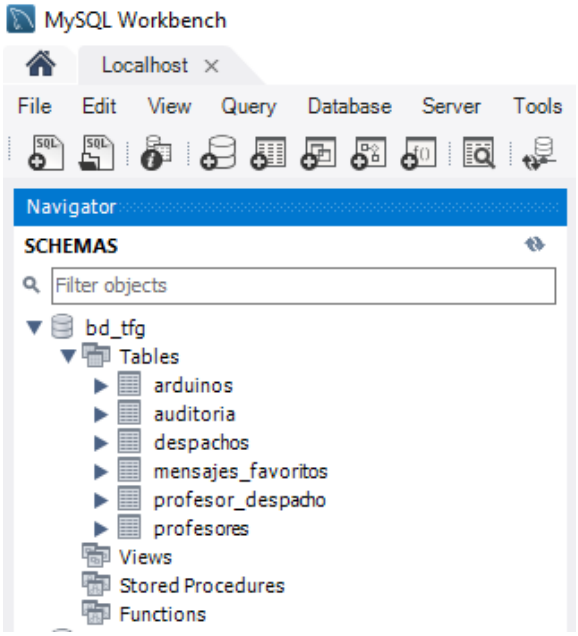

Figura 4.3: Esquema de la base de datos y sus tablas

Una vez ejecutado el código fuente para crear la base de datos, se procedió a rellenar esta con nuevas entradas en cada una de sus tablas. Un ejemplo, en la figura 4.4 podemos ver en color azul las filas de la tabla *Profesores* y en rojo la asignación entre profesores y despachos de la tabla *Profesores-Despachos*:

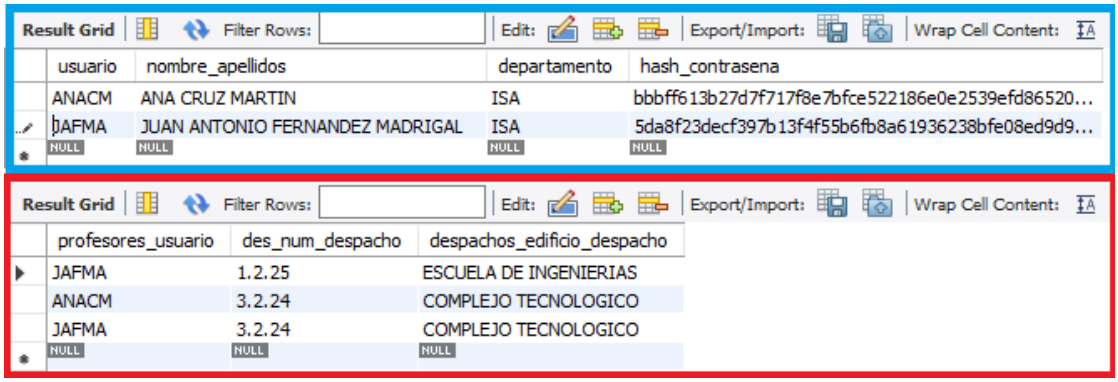

Figura 4.4: Tablas Profesor y Profesor-Despacho con sus respectivos valores

4.3 Servidor Intermedio Java

Una vez diseñado el diagrama de clases de nuestro servidor intermedio en el apartado 3.3, hemos de implementar cada uno de los métodos necesarios para el correcto funcionamiento de este. Debido a que dicho servidor se divide en dos clases, su código se describirá en dos subapartados, comentando las funciones más importantes.

- **Clase Servidor**

  Dentro de esta clase tenemos el programa principal que será responsable de la ejecución del servidor. Para realizar la implementación se han utilizados sockets TCP [29] de forma concurrente usando la interfaz Executor. Esta crea un hilo de ejecución por cada conexión que se realiza al servidor mediante el método *execute*, de forma que no se quede bloqueado y pueda seguir atendiendo peticiones. Para ello, el socket se creará con la dirección IP actual que tenga asignada la máquina que ejecute dicho programa y escuchará en el puerto que el administrador del sistema decida. Como se puede observar en la figura 4.5, el programa principal del servidor intermedio dispondrá de sentencias para control de errores y de un bucle para atender peticiones de forma indefinida, creando hilos gracias al método *execute* descrito anteriormente.

```
//Método principal para la ejecución del servidor
//Utiliza la interfaz EXECUTOR para crear conexiones concurrentes al servidor
public static void main(String[] args) throws IOException{
	ServerSocket socketServidor = null;
	Executor servicio = Executors.newCachedThreadPool();
	Connection conexion = ConexionBD.conectarConBD();
	try{
		InetAddress addr = InetAddress.getByName(IP);
		socketServidor = new ServerSocket(PUERTO, 50, addr);
		System.out.println("Escuchando en " + socketServidor);
		try{
		   while(true){
			System.out.println("Esperando conexiones...");
			servicio.execute(new Servidor(socketServidor.accept(), conexion));
		   }
		}catch(IOException e){
			System.err.println("Aceptar fallido.");
		}
	  }catch(IOException e){
		System.err.println("No se puede escuchar en el puerto: "+PUERTO);
	  }finally{
		try{
			socketServidor.close();
		}catch(IOException e){
			System.err.println("No se puede cerrar el puerto: "+PUERTO);
		}
	}
}
```

Figura 4.5: Programa principal del servidor

Una de las funciones más importantes del servidor intermedio es establecer la conexión con el servidor Arduino. Esta será llamada explícitamente desde el método *run* responsable del hilo de ejecución de cada conexión en caso de recibir un comando *SEND*, pasándose como argumentos la IP y puerto de la placa y mensaje a mostrar en pantalla. Para ello recurriremos de nuevo a los sockets TCP, creando una conexión cliente con el servidor Arduino con los parámetros mencionados anteriormente, controlando los errores y dando como salida de función el valor que nos devuelva la placa. La figura 4.6 contiene el código responsable de establecer el enlace con la placa:

```
private String enlaceArduino(String ip, int puerto, String mensaje) throws IOException {
	//Declaramos el socket y las líneas de entrada y salida
	Socket socketPlaca = null;
	BufferedReader entrada = null;
	PrintWriter salida = null;

	//Inicializamos el String del mensaje que irá a la placa
	String msgPlaca = "";

	//Creamos la conexión con la placa según los parámetros de entrada
	try {
		socketPlaca = new Socket(ip, puerto);
		entrada = new BufferedReader(new
		InputStreamReader(socketPlaca.getInputStream()));
		salida = new PrintWriter(new BufferedWriter(new
		OutputStreamWriter(socketPlaca.getOutputStream())),true);

		//Enviamos el mensaje a la placa y esperamos su respuesta
		try {
			System.out.println("Mensaje a la placa: "+mensaje);
			salida.println(mensaje);
			msgPlaca = entrada.readLine();
		}catch(IOException e) {
			System.out.println("IOException: " + e.getMessage());
		}catch(NullPointerException e) {
			msgPlaca = "-2";
		}
	}catch(IOException | NullPointerException e) {
		System.err.println("No puede establer conexión con la placa");
		msgPlaca = "-2";
	}finally {
		//Cerramos la conexión con la placa y las líneas de entrada y salida
		try {
			salida.close();
			entrada.close();
			socketPlaca.close();
			System.out.println("Desconexión con la placa realizada");
		}catch(NullPointerException e) {
		}
	}
	//La placa nos ha de devolver siempre alguno de estos valores {-1, 0, 1}
	return msgPlaca;
}
```

Figura 4.6. Implementación del método conexión entre servidor y placa

- **Clase ConexionBD**

Esta clase es la responsable de realizar la conexión entre el servidor Java y la base de datos MySQL, utilizando para ello la interfaz JDBC [30] [31]. En el programa principal del servidor, mostrado en la figura 4.5, se puede apreciar la creación de un objeto llamado *Connection*, este será el responsable de conectar con nuestra base de datos a partir de la clase ConexionBD, en la cual almacenaremos los parámetros necesarios además de un método para conectarnos a esta. En la figura 4.7 podemos observar los valores que se han introducido para crear la conexión al esquema *bd_tfg*, esquema que se mostró en el apartado 4.2:

```
public class ConexionBD {
    private static String BD = "jdbc:mysql://127.0.0.1:3306/bd_tfg";
    private static String driver = "com.mysql.jdbc.Driver";
    private static String usuario = "root";
    private static String password = "proot";
    private static Connection conexion;

    public static Connection conectarConBD() {
        try {
            Class.forName(driver);
            try {
                conexion = DriverManager.getConnection(BD, usuario, password);
            } catch (SQLException ex) {
                System.out.println("Error al crear una conexión con la base de datos");
            }
        } catch (ClassNotFoundException ex) {
            System.out.println("Driver no encontrado");
        }
        return conexion;
    }
}
```

Figura 4.7: Clase ConexionBD

Esta conexión con la base de datos será llamada de forma explícita para ejecutar diferentes instrucciones SQL según la cabecera del protocolo que las aplicaciones cliente envíen al servidor intermedio.

4.4 Servidor Arduino

A continuación se expone el código fuente que irá cargado en la placa Arduino MKR 1010 WiFi. Dicho código puede estructurarse en cuatro zonas: la inicialización de variables y librerías, el método para limpiar mensajes de profesores, la configuración inicial de la placa, y por último el bucle principal en ejecutivo cíclico. Dicho código aparecerá con comentarios explicativos de las líneas más importantes, así como un resumen de la funcionalidad que desempeña dicho código.

```
#include <SPI.h>
#include <WiFiNINA.h>           //Librería para el uso del módulo WiFi
#include <LiquidCrystal_I2C.h>  //Librería para el uso de la pantalla LCD 16x02
#include <Wire.h>               //Librería para el uso de los conectores SDA y
SCL de la placa
const int TAM = 5;              //Constante para el tamaño de los Arrays

//Inicializamos los parámetros de configuración del WiFi, IP estática y puerto
del servidor
char ssid[]   = "MIREDWIFI";             //Introducir el SSID de la red Wi-Fi
char pass[]   = "MICONTRASEÑA";         //Contraseña de dicha red
IPAddress IP_Estatica(192, 168, 0, 175);    //IP estática: 192.168.0.175
WiFiServer servidor(23);                    //Puerto: 23 (TELNET)

//Inicializamos los parámetros de configuración de la pantalla
LiquidCrystal_I2C lcd(0x27, 16, 2);   //Protocolo de pantalla I2C 0x27, 16
columnas, 2 filas

//Inicializamos los arrays de mensajes
String usuarios[TAM];
String mensajes[TAM];

//Variables globales para el control de los arrays
int ciclo;
int posicion;
int cantidad;

//Variables para el control del desplazamiento del mensaje
int longMsgAct = 0;
int contDesp = 0;
String mensajeActual = "";

//Método auxiliar para inicializar los arrays a su estado inicial y las variables
de control
void iniciarMensajes(){
  for(int i = 0; i < TAM; i++){
    usuarios[i] = "";
    mensajes[i] = "";
  }
  ciclo = 0;
  posicion = 0;
  cantidad = 0;
}
```

Figura 4.8: Inicialización de variables y librerías

En la sección de iniciación de variables y librerías que se muestra en la figura 4.8, declaramos cada una de las librerías que se van a utilizar y el tamaño de los vectores que contendrán los usuarios y sus mensajes. En un principio, este tamaño se fijó por defecto en 5 mensajes debido a que no suele haber más de cuatro profesores por despacho, dejando así un hueco extra en el hipotético caso de que cuatro profesores de un mismo despacho dejen un mensaje cada uno en circulación en la placa. Este tamaño puede modificado por el administrador si el despacho necesitase un número mayor de huecos para mensajes.

Además, en esta zona se declaran la red WiFi y la dirección IP estática que vaya a tener la placa, dos elementos muy importantes del código ya que necesitamos conectarnos a una red para poder establecer comunicación con el servidor intermedio y disponer de una dirección IP que esté almacenada en la base de datos y por lo tanto no cambia.

Respecto al puerto, todas las placas del sistema escucharán en el puerto 23, puerto que pertenece al protocolo TELNET debido a que no necesitaremos ningún tipo de seguridad a la hora de enviar el mensaje del servidor intermedio a la placa, ya que lo que se envía es lo que se muestra por pantalla.

Por último pero no menos importante, disponemos de un método para inicializar los usuarios y mensajes que éstos tengan en circulación en la placa, así como para establecer los valores iniciales del ciclo actual, la posición actual y la cantidad de mensajes que en circulación la placa. Cada una de estas variables desempeña un papel fundamental en el control del modelo ejecutivo cíclico implementado en la placa.

```
//Realizamos la configuración inicial de la placa
void setup() {
  //Inicializamos los variables y los arrays
  iniciarMensajes();

  //Configuramos el monitor serie
  Serial.begin(115200);

  //Configuramos la pantalla LCD
  lcd.init();
  lcd.backlight();
  lcd.clear();

  //Configuramos el WiFi e iniciamos el servidor en la IP asignada
  WiFi.config(IP_Estatica);
  WiFi.begin(ssid, pass);

  //Imprimimos un . cada 500ms por el monitor serie hasta que conecte a la red
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  //Mostramos por el monitor serie la IP para asegurarnos que se ha configurado
bien
  Serial.print("Conectado a ");
  Serial.print(ssid);
  Serial.print(" con la IP ");
  Serial.print(WiFi.localIP());
  Serial.println("");

  //También mostramos por pantalla que nos hemos conectado
  lcd.setCursor(0, 0);
  lcd.print("Conectado a:");
  lcd.setCursor(0, 1);
  lcd.print(ssid);
  delay(5000);

  //Iniciamos el servidor y borramos la pantalla LCD
  servidor.begin();
  lcd.clear();
}
```

Figura 4.9: Configuración inicial de la placa

En la configuración inicial de la placa de la figura 4.9 inicializaremos los vectores de usuarios y mensajes, configuraremos la pantalla LCD y conectaremos el dispositivo a la red WiFi que hayamos introducido previamente en la sección de la inicialización de variables y librerías. Una vez conectada la placa a la red WiFi se nos mostrará por la pantalla dicha red, corroborando la conexión a esta. De forma complementaria y en caso de tener la placa conectada al entorno de desarrollo de Arduino, dispondremos de la misma información a través del monitor serie que provee el IDE.

A continuación, en la figura 4.10 mostraremos la primera mitad del bucle principal de la placa, en donde podemos observar cómo se implantan la escucha de nuevas peticiones por parte de clientes y el desplazamiento de los mensajes que tienen un tamaño superior a dieciséis caracteres.

```
//Bucle principal en ejecutivo cíclico
void loop() {
  //Escuchamos si hay algún cliente
  WiFiClient cliente = servidor.available();
  if (cliente) {
    String lineaEntrada = cliente.readString();
//Leemos la línea que nos llega del cliente
    lineaEntrada = lineaEntrada.substring(0, lineaEntrada.length() - 2);
//Eliminamos los dos últimos caracteres basura

    //Troceamos la línea de entrada en usuario y mensaje separando en funcion del
caracter ':'
    int pos = lineaEntrada.indexOf(":");
    String usuario = lineaEntrada.substring(0, pos + 1);
    String mensaje = lineaEntrada.substring(pos + 1, lineaEntrada.length());

    if(mensaje.equals("--LMPZBRR--")){          //Observamos si el mensaje
recibido es el comando para limpiar la placa
      limpiarProfesor(usuario);
      servidor.println("1");                    //Enviamos un 1 para indicar que
se han borrado los mensajes de dicho usuario
    }else{
      //Leemos la entrada del cliente
      if(cantidad == TAM){                      //Comprobamos que no se haya
llegado al máximo de mensajes
      servidor.println("-1");                   //En caso de que se haya
alcanzado, enviamos un -1 que será interpretado como un error
        }else{
          //Añadimos el usuario y el mensaje a su array correspondiente
          usuarios[cantidad] = usuario;
          mensajes[cantidad] = mensaje;
          cantidad++;                           //Actualizamos la cantidad de
mensajes
          servidor.println("0");                //Si hemos podido añadir el
mensaje correctamente, enviamos un 0
        }
      cliente.stop();                           //Cerramos la conexión con el
cliente
    }
  }

  //Solo se accede si la longitud del mensaje actual es mayor a 16 caracteres
cada 0.75 segundos
  if(ciclo%30 == 0 && (longMsgAct - 16) >= 1 ){
    if(contDesp > longMsgAct - 16){
      contDesp = 0;
    }
    lcd.setCursor(0, 1);
    lcd.print(mensajeActual.substring(contDesp, longMsgAct));
    contDesp++;
  }
```

Figura 4.10: Bucle principal en ejecutivo cíclico (1/2)

Comencemos con la escucha de nuevas peticiones, en la cual observamos si hay o no un nuevo cliente. Si el caso es afirmativo, procederemos a leer la línea de entrada, que será dividida en dos trozos a partir del carácter ':', siendo el primero el usuario y el segundo el mensaje. Hemos de tener muy en cuenta que dentro de esta sección de código se tramita el añadir o el borrar mensajes de la placa. Si el mensaje es la palabra reservada *-LMPZBRR-* se llamará al método *limpiarProfesor* y se le pasará como argumento el usuario que ha mandado ese mensaje, enviando al cliente el mensaje de confirmación '1', corroborando el borrado de sus mensajes. En caso contrario, almacenaremos el usuario y el mensaje en sus vectores correspondientes, incrementando la cantidad de mensajes en el sistema y enviando al cliente el mensaje de confirmación '0'. Si por algún casual el número de mensajes ha llegado al máximo y no se puede añadir el nuevo mensaje, se le hará saber al cliente con el mensaje de confirmación '-1'.

Respecto a la sección de código responsable del desplazamiento de los mensajes, sólo se ejecutará en el caso de que la longitud del mensaje a mostrar sea superior a la de los dieciséis caracteres que permite mostrar la pantalla. Para realizar esto, calculamos cuantos caracteres de sobra tiene el mensaje, y este número de caracteres será la cantidad de veces que se desplace hacia la izquierda volviendo posteriormente a su forma original. Esto se ve con mayor detalle en la tabla 4.4:

NÚMERO ITERACIÓN	***MENSAJE MOSTRADO POR PANTALLA***
Mensaje original	Un mensaje de prueba
1ra iteración	"Un mensaje de pr"
2da iteración	"n mensaje de pru"
3ra iteración	" mensaje de prue"
4ta iteración	"mensaje de prueb"
5ta iteración	"ensaje de prueba"
6ta iteración	"Un mensaje de pr"

Tabla 4.4: Fases del desplazamiento de un mensaje en pantalla

En la figura 4.11 mostraremos la segunda mitad del bucle principal de la placa, donde podemos ver en sí la sección de actualización de mensajes, ciclo, y posición actuales. Como podemos observar al final del código, disponemos de una espera activa de 20ms, si multiplicamos dicha cantidad por los 750 ciclos que tarda en ejecutarse esta sección, obtenemos 15000ms. La variable ciclo aumenta su valor cada 20ms, por tanto cada quince segundos se reinicia y actualiza el mensaje que se muestra por pantalla. Este reinicio del ciclo es el corazón del ejecutivo cíclico que tiene el control de la placa Arduino.

```
    //Cambia de mensaje cada 15 segundos
  if(ciclo%750 == 0){
    //Limpiamos pantalla para evitar solapamientos
    lcd.clear();
    //Si la cantidad es 0, mostramos este mensaje informativo
    if(cantidad == 0){
      lcd.setCursor(0, 0);
      lcd.print("Sin mensajes que");
      lcd.setCursor(0, 1);
      lcd.print("mostrar");
    }else{
      //Resetea la posición actual si se ha llegado al maximo de
mensajes
      //o si el usuario está vacío
      if(posicion == TAM || usuarios[posicion].equals("")){
        posicion = 0;
      }

      //Muestra el usuario en la primera linea
      lcd.setCursor(0, 0);
      lcd.print(usuarios[posicion]);
      //Muestra el mensaje en la segunda línea
      lcd.setCursor(0, 1);
      lcd.print(mensajes[posicion]);

      //Cargamos la información necesaria para desplazar el mensaje si
fuera necesario
      mensajeActual = mensajes[posicion];
      longMsgAct = mensajes[posicion].length();

      posicion++; //Actualizamos la posicion actual para el siguiente
mensaje
    }
    ciclo = 0;  //Reseteamos el contador del ciclo
  }
  ciclo++;  //Actualizamos el ciclo
  delay(20);
}
//Fin del bucle principal en ejecutivo cíclico
```

Figura 4.11: Bucle principal en ejecutivo cíclico (2/2)

Por último, en la figura 4.12 tenemos una de las funciones primordiales de la placa, el poder eliminar los mensajes en circulación de los profesores que lo deseen a través del botón *Limpiar Placa* de la aplicación cliente de usuario. Para ello, este método se basa en buscar y copiar en unos vectores auxiliares aquellos usuarios y sus respectivos mensajes que no sean igual al usuario que ha sido pasado como argumento. Tras haber realizado este paso, los usuarios y mensajes almacenados en los vectores auxiliares serán copiados de nuevo en los vectores originales de tal forma que no habrá usuarios y mensajes vacíos entre ellos. También se realiza una actualización del ciclo, posición y cantidad de mensajes actuales en circulación.

```
//Método para borrar los mensajes de un determinado profesor
void limpiarProfesor(String usuario){
  String usrAux[TAM];
  String msgAux[TAM];
  int nuevaPos = 0;

  //Vamos a recorrer hasta la cantidad de mensajes que tengamos almacenados
  for(int i = 0; i < cantidad; i++){
    //Si son distintos, los copiamos en los arrays auxiliares
    if(!usuarios[i].equals(usuario)){
      usrAux[nuevaPos] = usuarios[i];
      msgAux[nuevaPos] = mensajes[i];
      nuevaPos++;
    }
  }

  //Copiamos los arrays auxiliares en los originales
  for(int j = 0; j < TAM; j++){
    if(j < nuevaPos){
      usuarios[j] = usrAux[j];
      mensajes[j] = msgAux[j];
    }else{
      usuarios[j] = "";
      mensajes[j] = "";
    }
  }
  //Reseteamos ciclos y posiciones
  //Actualizamos la cantidad de mensajes en circulación
  ciclo = 0;
  posicion = 0;
  cantidad = nuevaPos;
}
```

Figura 4.12: Método para limpiar mensajes de un determinado profesor

4.5 Aplicación cliente

Al igual que ocurrió con el servidor intermedio, las aplicaciones cliente se desarrollaron según su diagramas de clases, ya que estas tienen partes en común como pueden ser el programa principal, la ventana de identificación o el método de conexión al servidor intermedio. Por tanto, en este apartado se describirán dichas clases comunes, siendo vistas las partes específicas de cada aplicación en su respectivo apartado. Hay que mencionar que la funcionalidad de cada una de las aplicaciones será explicada en sus respectivos manuales de uso (apéndices B y C)

- **Clase Login**

Esta es la primera interfaz que se cargará al ejecutar nuestras respectivas aplicaciones, en ella se nos pedirán nuestras credenciales tal y como se aprecia en la figura 4.13:

Figura 4.13: Vista de la ventana de autenticación

Esta se conectará al servidor intermedio y desde allí con la base de datos, comprobando que los valores sean correctos. Para dar seguridad a la contraseña, se envía el hash resultante de usar una función *SHA-256*, número que será comparado con el almacenado en la base de datos. Una vez nos hayamos identificado, la ventana de ingreso se hará invisible y la interfaz principal se volverá visible.

- **Clase Conexión**

Encargada de establecer un socket con el servidor que ha sido pasado como argumento. Esta enviará peticiones al servidor intermedio, siendo el contenido de la línea de salida del cliente el correspondiente comando del protocolo interno. En la figura 4.14 podemos ver el método *conectar* de esta clase:

```
public static String conectar() throws IOException {
	Socket socketCliente = null;
	BufferedReader entrada = null;
	PrintWriter salida = null;
	String lineaServidor = "";

	try {
		socketCliente = new Socket(IP, puerto);
		//Creamos el canal de entrada
		entrada = new BufferedReader(new
			InputStreamReader(socketCliente.getInputStream()));
		//Creamos el canal de salida
		salida = new PrintWriter(new BufferedWriter(new
			OutputStreamWriter(socketCliente.getOutputStream())),true);
		} catch (IOException e) {
			System.err.println("No puede establer canales de E/S para la
				conexión");
			lineaServidor = "-3";
		}
	BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));

	try {
		salida.println(msg);
		lineaServidor = entrada.readLine();
		System.out.println("Valor "+lineaServidor);
	} catch (IOException | NullPointerException e) {
		System.out.println("IOException: " + e.getMessage());
		lineaServidor = "-3";
	}
	// Libera recursos
	salida.close();
	entrada.close();
	stdIn.close();
	socketCliente.close();

	return lineaServidor;
}
```

Figura 4.14: Método conectar de la clase Conexión

- **Clase Controlador**

Esta clase es la encarga de capturar los eventos realizados en la interfaz de la aplicación, como pueden ser pulsar un botón o seleccionar una fila determinada de una lista. A continuación se expone en la figura 4.15 el esquema general que ha sido utilizado para controlar cada uno de esos eventos.

```
public void actionPerformed(ActionEvent e) {
            if(e.getActionCommand().equals("COMANDO_1")) {
                  //ALGORITMO COMANDO 1
            }else if(e.getActionCommand().equals("COMANDO_2")) {
                  //ALGORITMO COMANDO 2
            . . . . . .
            }else if(e.getActionCommand().equals("COMANDO_N")) {
                  //ALGORITMO COMANDO N
            }else {
            }
```

Figura 4.15:Esquema del controlador de ambas aplicaciones

4.5.1 Profesor

Lo único destacable de esta aplicación es la interfaz gráfica en ejecución, la cual podemos apreciar en la figura 4.16, puesto que los componentes relevantes de ella han sido explicados en el apartado anterior:

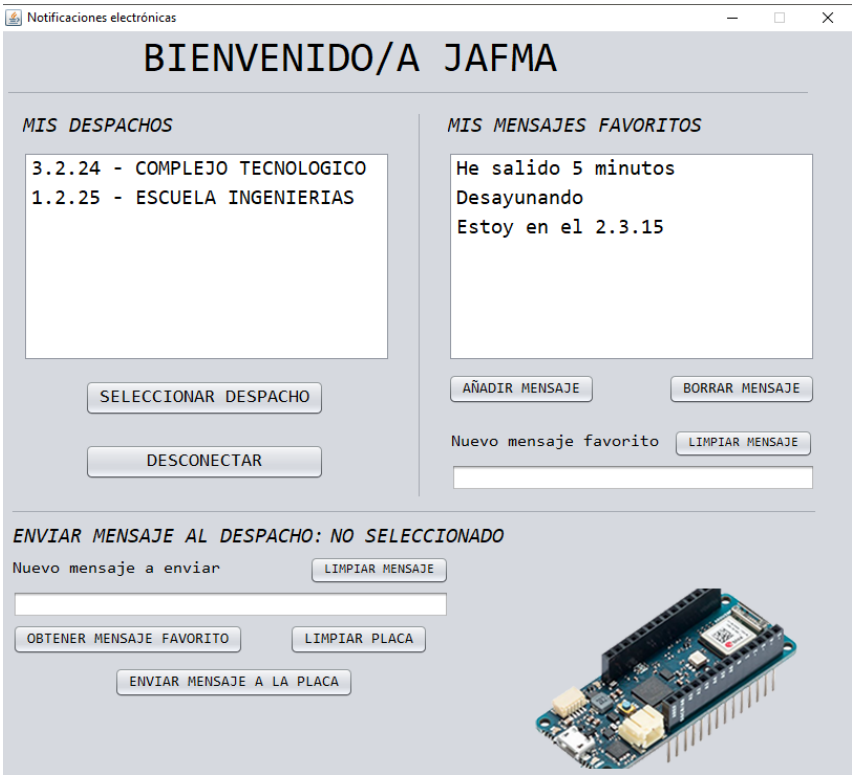

Figura 4.16 Panel del programa principal del usuario en ejecución

4.5.2 Administrador

De la misma forma que ocurre con la aplicación de usuario, lo único destacable de esta es su interfaz gráfica, visible en la figura 4.17, puesto que los componentes importantes de ella han sido explicados con anterioridad. Hay que destacar que esta dispone de una función hash *SHA-256* para evitar que la contraseña usuario sea enviada por la red en texto plano:

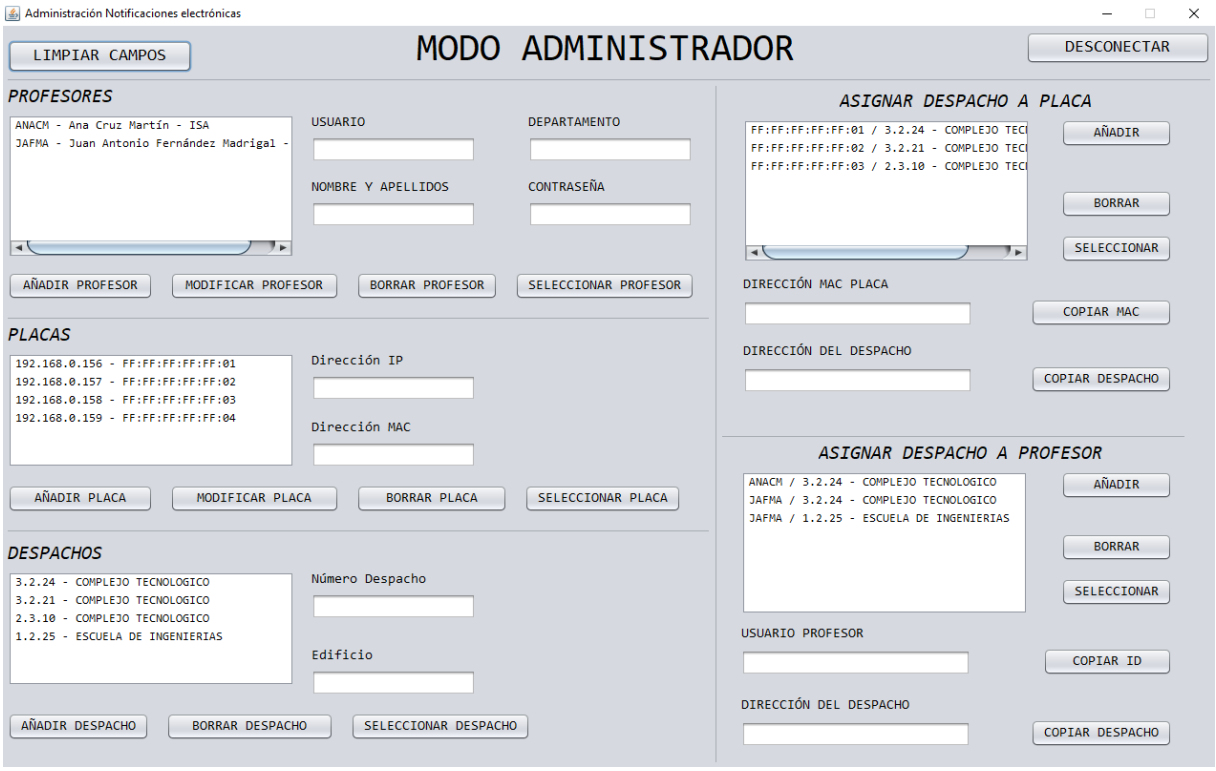

Figura 4.17: Panel del programa principal del administrador en ejecución

5

Despliegue y pruebas de funcionamiento del sistema

Durante el desarrollo de este capítulo se describirá el hardware sobre el cual fue realizado el despliegue de los distintos componentes del sistema, así como las pruebas que se llevaron a cabo para corroborar su funcionamiento.

5.1 Despliegue

A la hora realizar el despliegue del sistema se utilizaron dos equipos personales: un ordenador de sobremesa que tendría la función de ejecutar el servidor intermedio y la base de datos, y un ordenador portátil con mucha menor capacidad de cómputo que el ordenador de sobremesa cuya función sería la de ejecutar tanto la aplicación cliente de usuario como la de administrador. Hay que remarcar que la placa Arduino no se menciona debido a que no necesita de ningún ordenador personal para ejecutar el servidor que esta tiene programada, simplemente ha de estar conectada a la red eléctrica o a una batería, sin embargo hemos de tenerla muy en cuenta puesto que es una pieza fundamental para el sistema.

Las características técnicas de cada uno de estos equipos se muestran en la tabla 5.1, pudiendo verse cada uno de ellos en las figuras 5.1 y 5.2:

TIPO PC	***CPU***	***RAM***	***GPU***	***DISCO***
Sobremesa	i3-2100@3.2GHz	8GB@1333MHz	NVidia GTX 950	SSD 480GB
Portátil	Celeron B815@1.6GHz	4GB@1333MHz	Intel Graphics	HDD 700GB

Tabla 5.1: Características de los ordenadores personales usados para el despliegue

Para la ejecución del servidor intermedio y de la aplicación cliente, se realizó un volcado del código fuente de estos en ejecutables Java, los cuales tienen extensión .JAR y se pueden lanzar en cualquier sistema que tenga Java instalado. Este proceso será descrito con mayor detalle en el manual de instalación del sistema (apéndice A).

Figura 5.1: Ordenador de sobremesa

Figura 5.2: Ordenador portátil

5.2 Pruebas

Durante toda la fase de desarrollo software de los distintos componentes del sistema, se fueron realizando diferentes ensayos para corroborar el funcionamiento de cada uno de los componentes por separado. Una vez se tenía constancia de que este funcionaba correctamente, se pasaba a la implementación de otra de sus funcionalidades, repitiendo el mismo paso que se ha explicado antes hasta completar el desarrollo de este.

Una vez realizadas las pruebas de cada uno de los componentes por separado, se han de realizar las pruebas de funcionamiento conjunto. A continuación se detallarán las pruebas que se realizaron para asegurar el trabajo de todos los componentes al unísono, todos ellos interconectados dentro de la red local privada del domicilio personal.

En primer lugar se verificó que ambas aplicaciones clientes (la de usuario y la de administrador) eran capaces de realizar la conexión al servidor intermedio cuando introducían sus credenciales en la ventana de autenticación, ya que éste es el primer paso que se realiza cuando queremos acceder al sistema. Luego, se comprobó que tanto la aplicación de usuario como la aplicación de administrador enviaban información al servidor intermedio siguiendo el protocolo que se había establecido en la fase de implementación y recibían respuesta de este, generando los distintos mensajes de notificación de éxito o error.

Tras este paso, se verificaron todas las funcionalidades de la aplicación de administrador, ya que esta es la que opera sobre la base de datos introduciendo, borrando o modificando las distintas entradas que existen en las tablas, tablas a las que posteriormente acceder a el usuario desde su respectiva aplicación. Por citar varios ejemplos, se comprobó que a la hora de dar de alta un nuevo profesor se generaba una nueva entrada en la tabla de profesores y además se creaba un usuario para dicho profesor con los permisos adecuados para que este pueda acceder a sus mensajes favoritos o a los despachos que tenga asignado.

A continuación, se comprobó, como se puede ver en las figuras 5.3, 5.4 y 5.5, que tanto las instrucciones de envío de mensajes por parte del cliente se mostraban por la pantalla LCD conectada la placa, estando conectada a la red eléctrica a través de un transformador, así como las instrucciones de añadido o borrado de mensajes favoritos generaban o borraban filas en su correspondiente tabla de la base de datos.

Figura 5.3. Mensaje de conexión a la red WiFi

Figura 5.4: Mensaje inicial tras conexión a la red WiFI

Figura 5.5: Usuario con su respectiva notificación

Por último, se comprobó que las distintas instrucciones que pasaban por el servidor intermedio generaban una nueva entrada en la tabla de auditoría de la base de datos y que este era concurrente, atendiendo varias peticiones de diferentes clientes a la vez creando varias instancias de la aplicación de usuario.

Además, se verificó la concurrencia del servidor intermedio creando dos instancias de la aplicación cliente de usuario, cada una con unas credenciales de ingreso diferentes, desde las que se le enviaban peticiones de envío de mensajes a una misma Arduino.

Como bien se dijo en el apartado de despliegue de este capítulo, el sistema se probó de forma distribuida, es decir, cada uno de los componentes se ejecutaron en ordenadores distintos para asegurar la correcta comunicación de los componentes pero conectados en la misma red local, debido a la incapacidad de poder realizar un “despliegue auténtico” del servidor en una máquina dedicada a ello, ya que esta sección del trabajo ha sido desarrollada durante los meses de agosto y septiembre, y no ha sido posible el uso de un laboratorio.

6

Conclusiones y trabajos futuros

El fin de este capítulo es exponer las conclusiones a las que se ha llegado tras la realización de este trabajo de fin de grado, estando dividido en dos subcapítulos: las conclusiones alcanzadas luego de su realización, y las nuevas funcionalidades que pueden añadírsele en el futuro.

6.1 Conclusiones

El principal objetivo de este trabajo de fin de grado era diseñar e implementar un sistema distribuido que sustituyese las notificaciones escritas dejadas en los despachos de los profesores por unas electronicas a través de una pantalla, objetivo que ha sido alcanzado tras haber seguido cada una de las fases del proyecto, las cuales se comentarán durante este apartado.

Uno de los primeros pasos en el desarrollo de este sistema fue la elección de los componentes hardware y posterior montaje de estos que conforman la parte física de este, una parte fundamental debido a que es necesaria para mostrar las notificaciones electrónicas que los profesores se pueden dejar en caso de que tengan que abandonar su despacho por un breve periodo de tiempo. Se realizó un análisis de casos de uso del sistema y de los actores que tendrían presencia en este, y en base a estos se diseñaron el resto de componentes que conforman dicho sistema. Estos componentes son el servidor de la placa, la base de datos, el servidor intermedio y las aplicaciones de usuario y administrador.

Se realizó la implementación de los componentes mencionados anteriormente en base a dichos casos de uso con un protocolo de comunicación propio del sistema que permite la interacción entre ellos. Asimismo, y siendo una parte clave en el desarrollo del sistema, se llevaron a cabo las pruebas pertinentes para asegurar el correcto funcionamiento de todos los componentes de este al unísono

Además, y como aspecto puramente personal, durante la elaboración de este trabajo de fin de grado he aprendido a utilizar varias herramientas de diseño y desarrollo, dentro de las cuales se encuentra un grupo con las que nunca antes había trabajado o el tiempo de trabajo fue muy escaso, como pueden ser Fritzing, MySQL, MagicDraw o el entorno de desarrollo de Arduino. Por lo tanto, para mí otro objetivo alcanzado es haber aprendido a utilizar estas herramientas de forma correcta, ya que ha supuesto un tiempo extra para poder llegar a manejarlas con una soltura aceptable.

A continuación se comentarán algunos problemas que tuvieron lugar durante las fases de diseño y desarrollo de los componentes del sistema:

- **Problemas en el diseño de la base de datos**

  Durante la fase de diseño del modelo entidad-relación en ORACLE Datamodeler ocurrieron dos problemas que resultaron ser bastante tediosos de solucionar.

  El primero de ellos, y el más engorroso, fue que la herramienta no guardaba correctamente el modelo, por lo que el diseño de esta tuvo que realizarse de una sola pasada.

  El segundo error encontrado tiene que ver con la generación automática del código. Datamodeler es una herramienta que simplifica bastante el hecho de diseñar una base de datos, sin embargo, y debido a una falla interna de este, a la hora de crear el código se produce una duplicación de las claves ajenas como puede verse en la figura 6.1, por lo que hay que depurar el código.

```
ALTER TABLE arduinos
    ADD CONSTRAINT ard_desp_fk FOREIGN KEY
( despachos_numero_despacho,

despachos_edificio_despacho )
        REFERENCES despachos ( numero_despacho,
                               edificio_despacho );

ALTER TABLE despachos
    ADD CONSTRAINT desp_ard_fk FOREIGN KEY
( arduinos_direccion_mac )
        REFERENCES arduinos ( direccion_mac );

ALTER TABLE mensajes_favoritos
    ADD CONSTRAINT mens_fav_prof_fk FOREIGN KEY
( profesores_usuario )
        REFERENCES profesores ( usuario );

ALTER TABLE profesor_despacho
    ADD CONSTRAINT prof_desp_despacho_fk FOREIGN KEY
( des_num_despacho,

despachos_edificio_despacho )
        REFERENCES despachos ( numero_despacho,
                               edificio_despacho );

ALTER TABLE profesor_despacho
    ADD CONSTRAINT prof_desp_profesores_fk FOREIGN KEY
( profesores_usuario )
        REFERENCES profesores ( usuario );

ALTER TABLE arduinos
    ADD CONSTRAINT ard_desp_fk FOREIGN KEY
( despachos_numero_despacho,

despachos_edificio_despacho )
        REFERENCES despachos ( numero_despacho,
                               edificio_despacho );
```

Figura 6.1: Duplicación de claves ajenas en el código generado por Datamodeler

Además, hay que remarcar que aunque ORACLE SQL y MySQL son parecidos y de la misma empresa, no son el mismo lenguaje a pesar de compartir exactamente la misma filosofía, por lo que hubo que realizar cambios en este a la hora de ejecutarlo en el servidor MySQL del sistema.

- **Problemas en el desarrollo del servidor Arduino**

Una de las dificultades encontradas fue hallar el modo de conseguir que la Arduino MKR 1010 WiFi tuviera una dirección IP estática. Por defecto, la librería WiFiNINA asigna una dirección IP a la placa a través de un protocolo de configuración dinámica de host (DHCP) si no se llama a un método de dicha librería (*WiFi.config*), el cual tiene como argumento una dirección IP.

El problema residía en que el firmware del módulo WiFi de la placa no estaba actualizado a su última versión, por lo cual, la placa ignoraba el método de configuración de la dirección IP para que esta fuera estática y seguía asignando la IP de forma dinámica, algo que no es compatible con este sistema, ya que las direcciones de cada Arduino se guardan en la base de datos y una asignación estática generaría conexiones incorrectas. Una vez actualizado el firmware, siguiendo las directrices de la web oficial de Arduino, se solventó el problema de la dirección estática. Esta solución está detallada en el manual de instalación de Arduino (apéndice A.1)

6.2 Trabajos futuros

El sistema, tal y como se presenta en este trabajo de fin de grado, no es más que un mero prototipo. Todas las mejoras que se van a comentar a continuación pueden formar parte de futuros desarrollos que mejorarían su funcionalidad.

La parte hardware del sistema podría ser actualizada añadiendo una batería externa de polímero de lítio de 3.7 voltios y 700 miliamperios-hora mínimo (batería recomendada por la propia Arduino) para sustituir la actual alimentación por cable. Asimismo, la conexión entre la placa y la pantalla podría ser sustituida por un circuito impreso, ahorrando mucho espacio y simplificando de forma drástica el ensamblaje. Incluso se podría añadir una carcasa al montaje final con elementos electrónicos como ledes de encendido, estado…

En el lado software podríamos incluir una gran variedad de mejoras, como el desarrollo de una aplicación móvil para los clientes con el mismo tipo de funcionalidades, añadiendo nuevas u optimizando las ya presentes. También se podría desarrollar un servidor intermedio que respondiese peticiones *HTTP* en lugar de comandos internos. Por otro lado, se podría realizar una mejora general en la administración de la base de datos creando nuevas funciones como pueden ser un sistema gestor de contraseñas, procedimientos para realizar estadísticas, disparadores, trabajos periódicos… elementos que pueden mantener con mucha mayor facilidad la consistencia de los datos de una base de datos de producción y que no se han llegado a implementar dada la naturaleza de prototipo de este proyecto.

Apéndice A
Manual de instalación

A continuación se explicará cómo realizar la instalación de cada uno de los componentes del sistema. Para ello, será necesario disponer de los componentes electrónicos necesarios para el hardware del sistema, tener un ordenador que tenga instalado MySQL y Java para actuar como servidor, y tener otro ordenador personal con Java para ejecutar las aplicaciones clientes.

A.1 Servidor Arduino

Para cargar el código en nuestra placa necesitaremos el entorno de desarrollo de Arduino. En caso de no tenerlo instalado en nuestro ordenador personal, iremos a su página oficial [4] y desde allí lo descargaremos.

Una vez instalado procederemos a ejecutarlo. Cuando la aplicación esté abierta, conectaremos nuestra placa al ordenador con un cable micro-USB y seleccionaremos en *Herramientas* el tipo de placa y el puerto en la que se encuentra conectada, tal y como se muestra en la figura A.1:

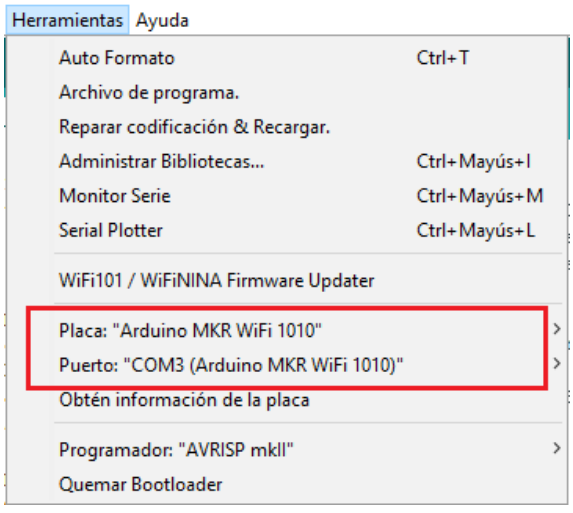

Figura A.1: Selección de la placa y del puerto

Realizado este paso, hemos de comprobar que tengamos instaladas las librerías necesarias para el correcto funcionamiento de la placa usando el *Gestor de Bibliotecas* incluido en la opción *Administrar Bibliotecas* del menú *Herramientas*. Estas son *WiFiNINA* y *LiquidCrystal_I2C*; en la figura A.2 podemos ver el proceso:

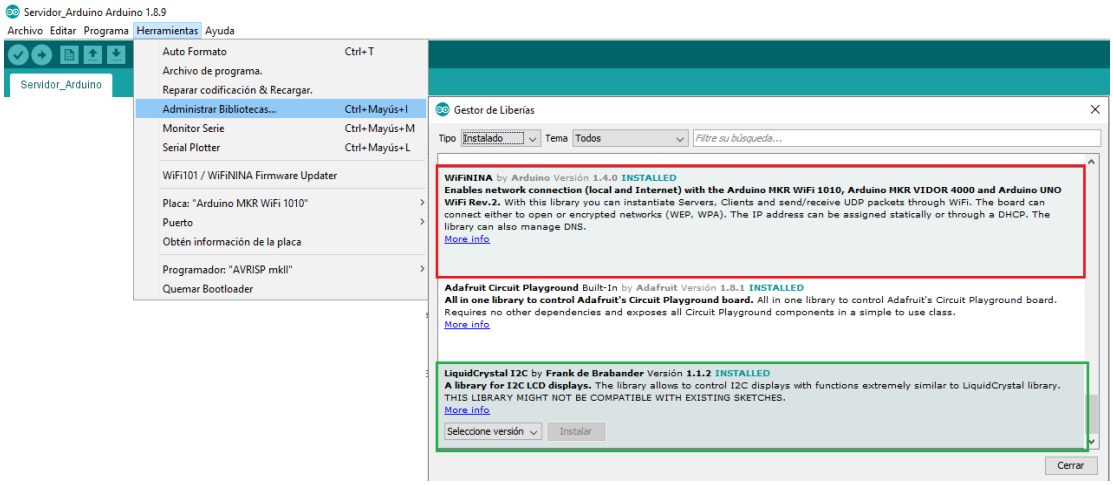

Figura A.2: Gestor de librerías de Arduino

Realizada la instalación de las bibliotecas, el siguiente paso será comprobar el firmware del módulo WiFi de la placa y si corresponde, actualizarlo. Para realizar esta acción clicaremos sobre *Archivo*, seleccionamos *Ejemplos*, escogemos *WiFiNINA* y el subapartado *Tools*, eligiendo por último *CheckFirmwareVersion*. Estos pasos se ven a continuación en la figura A.3:

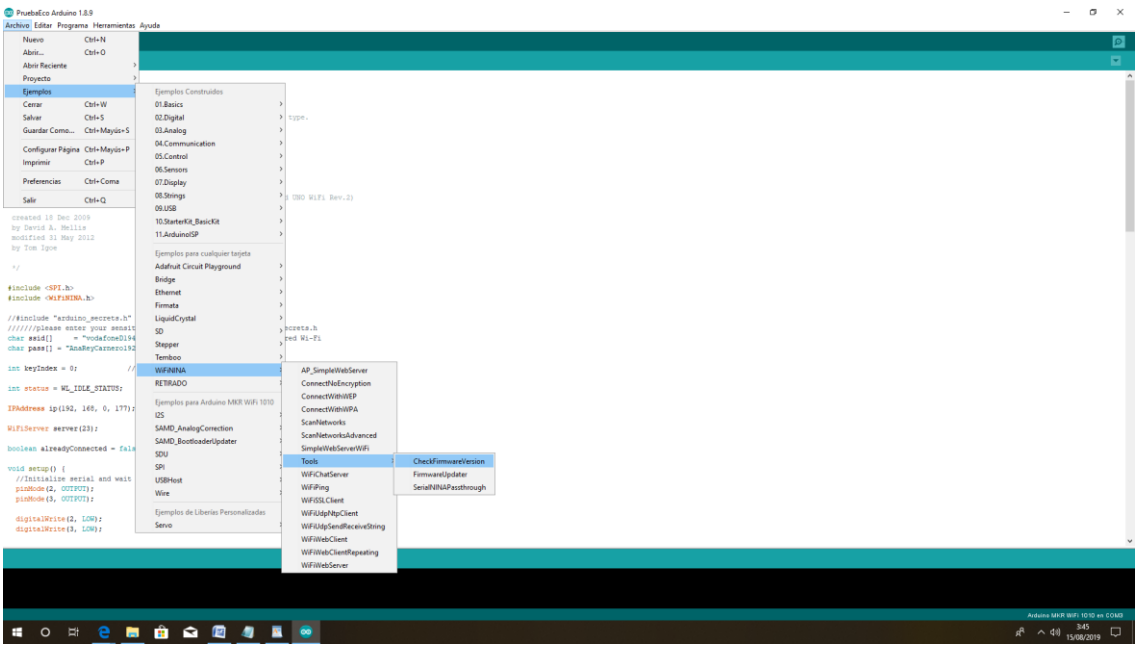

Figura A.3: Pasos para realizar la actualización del firmware WiFi

Tras realizar estos pasos, subiremos el código de actualización del firmware seleccionado a nuestra Arduino y dejaremos que se ejecute. Una vez realizada la ejecución de este, obtendremos un mensaje en el monitor serie de nuestro entorno de desarrollo tal y como se muestra la figura A.4:

```
WiFiNINA firmware check.

Firmware version installed: 1.2.1
Latest firmware version available : 1.2.1

Check result: PASSED
```

Figura A.4: Confirmación de la actualización del firmware

Luego de haber comprobado que tenemos el firmare actualizado, procederemos a abrir el fichero con el código del servidor Arduino. Para ello haremos clic izquierdo en la pestaña *Archivo*, obteniendo como resultado un desplegable, dentro de este seleccionaremos *Abrir*. Ahora sólo tenemos que buscar la carpeta en la que se encuentre el código del servidor Arduino. Este paso podemos verlo más detalladamente en la figura A.5:

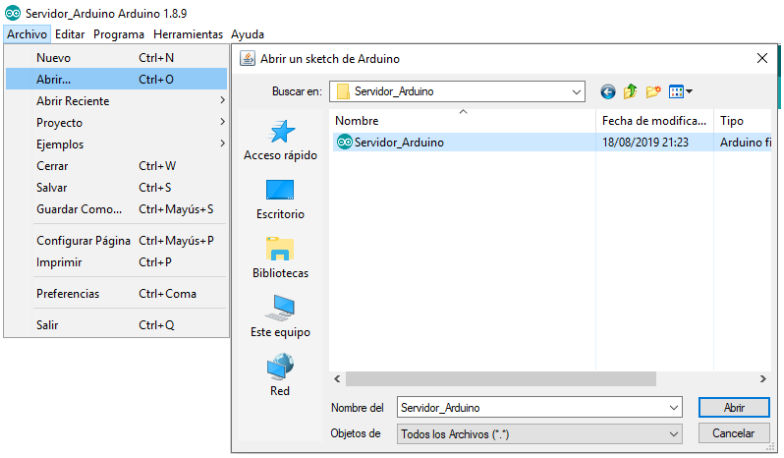

Figura A.5: Apertura del código del servidor Arduino

Realizado este paso, el entorno nos dirá que el código ha sido subido de forma correcta y podremos observar que la placa comienza a funcionar.

```
Subido

Verify 23132 bytes of flash with checksum.
Verify successful
done in 0.022 seconds
CPU reset.
```

Figura A.6: Confirmación de la carga del código en la placa.

A.2 Servidor Intermedio Java

A continuación se explicará la forma de realizar el volcado del código del servidor en un fichero ejecutable .JAR. El primer paso es iniciar Eclipse, clicar sobre la pestaña *File* y posteriormente clicar en la opción *Export* tal y como se ve a continuación en la figura A.7.

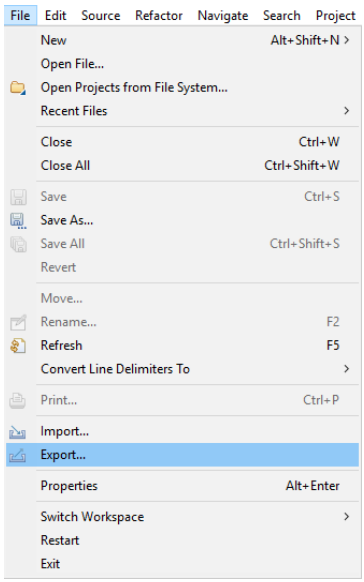

Figura A.7: Desplegable de la pestaña FILE

Tras haber realizado este paso se nos abrirá un cuadro de diálogo con múltiples elecciones. Para generar el fichero ejecutable hemos de expandir la carpeta llamada Java, seleccionar *Runnable JAR File* y clicar en el botón siguiente. Este paso puede verse con mayor claridad en la figura A.8.

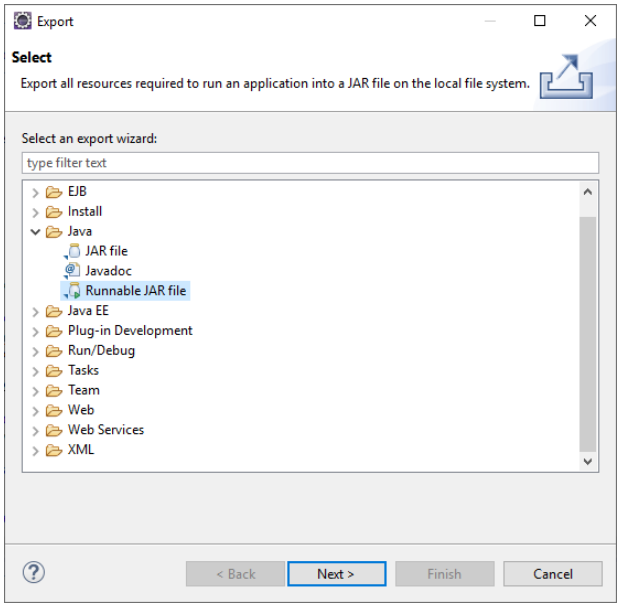

Figura A.8: Ventana de exportación con sus opciones

A continuación seleccionaremos el fichero con el método principal para la ejecución del código y seleccionaremos una carpeta de destino para el fichero ejecutable. En la figura A.9 podemos observar un volcado de ejemplo del código fuente del servidor, posteriormente la figura A.10 observamos el fichero resultado de este proceso. Estos pasos se realizan tanto con el código fuente de las aplicaciones cliente del usuario y del administrador como con el del servidor intermedio.

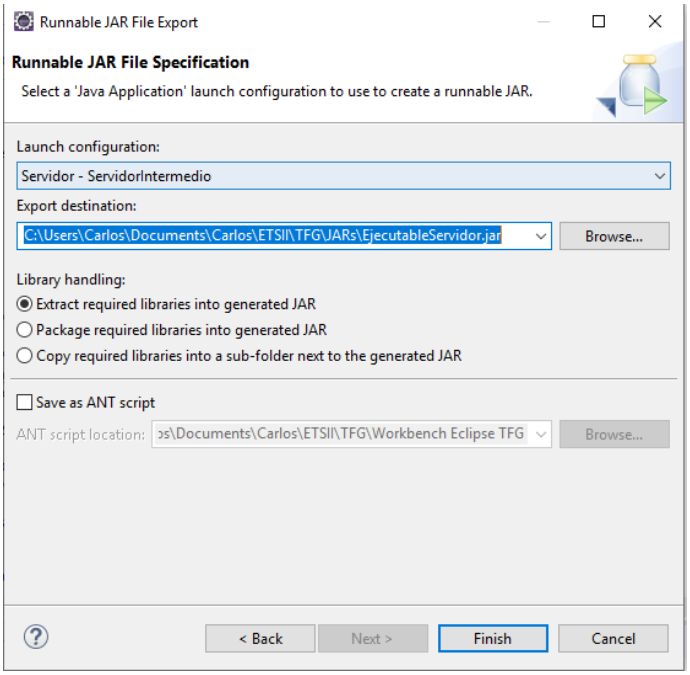

Figura A.9: Ventana de exportación final

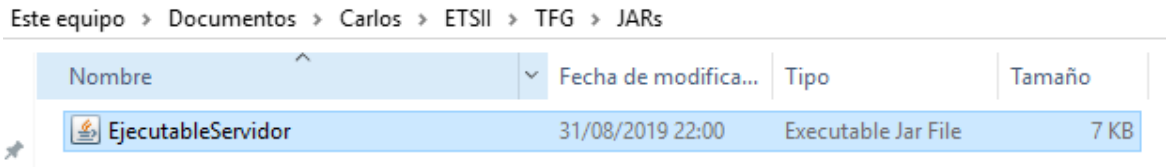

Figura A.10: Ejecutable resultante de la exportación

Hay que remarcar que el servidor intermedio no dispone de una interfaz gráfica como la que tienen las aplicaciones cliente, por tanto, la ejecución de este fichero ha de realizarse por consola. Para ello, abriremos una terminal de comandos, nos posicionaremos en la carpeta en la que esté el ejecutable del servidor e introduciremos esta instrucción:

```
java –jar nombreServidor.jar
```

Una vez introducida, nuestro servidor comenzará a escuchar peticiones de clientes.

A.3 Base de datos

Como bien se dice en la introducción de este anexo, necesitamos tener instalado MySQL en el ordenador que vaya a cumplir la función de servidor. Para ello iremos a su página oficial y desde allí lo descargaremos.

Una vez completado este paso, abriremos MySQL y crearemos una nueva conexión tal y como se ve en la figura A.11, introduciendo los parámetros que nosotros deseemos:

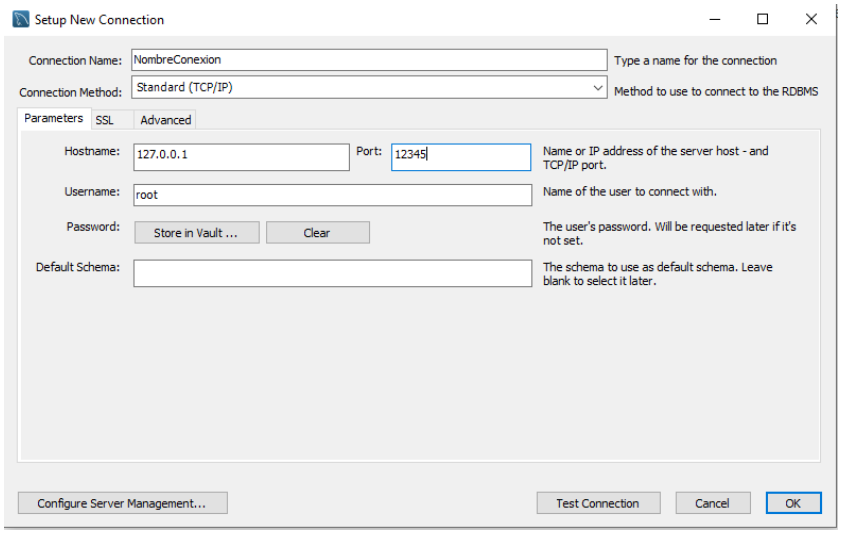

Figura A.11: Creación de una nueva conexión en MySQL

Realizado este paso, abriremos la nueva conexión que hemos creado en el editor y crearemos un nuevo esquema. Las figura A.12 y a A.13 muestran este proceso:

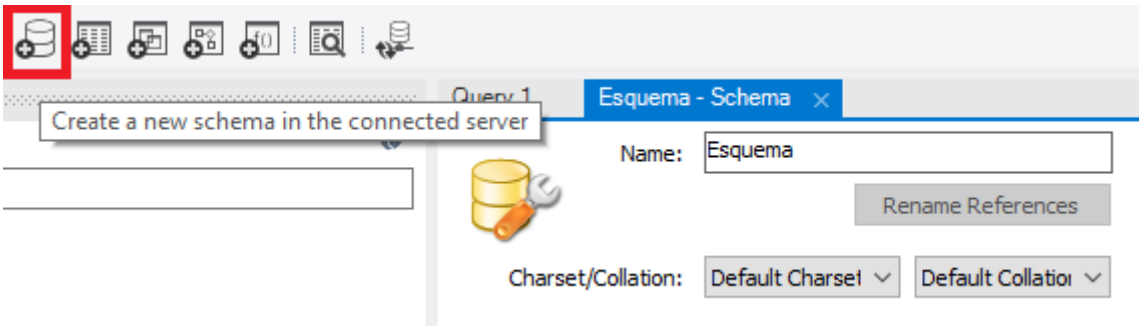

Figura A.12: Creación de un nuevo esquema (1/2)

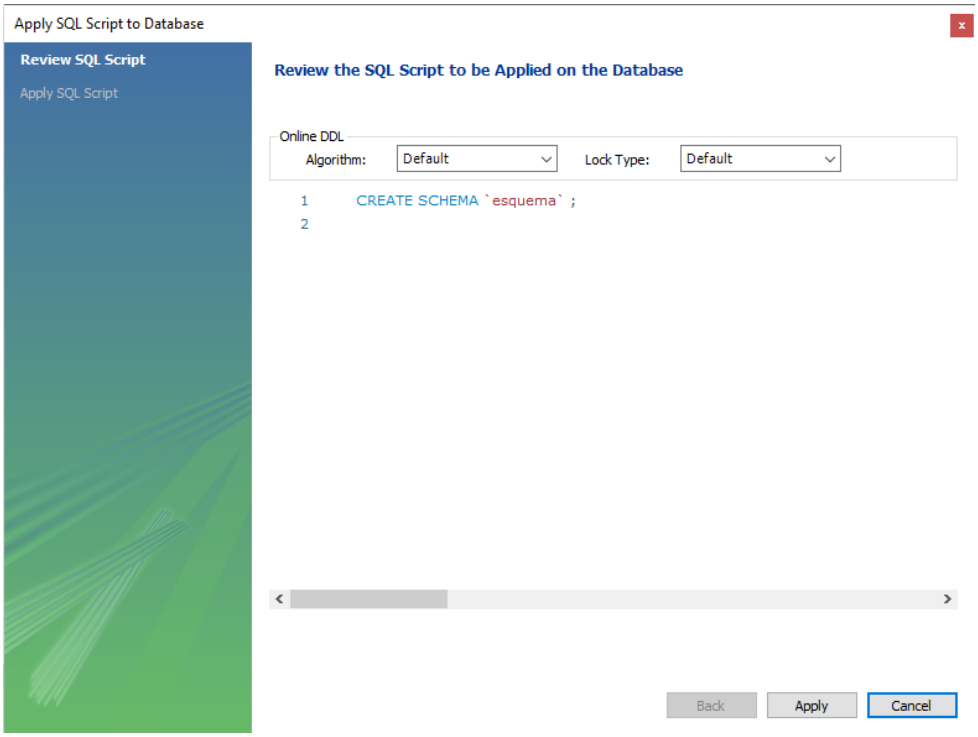

Figura A.13: Creación de un nuevo esquema (2/2)

Una vez creado nuestro esquema, este aparecerá en el navegador de esquemas, en la figura A.14 podemos apreciar esto:

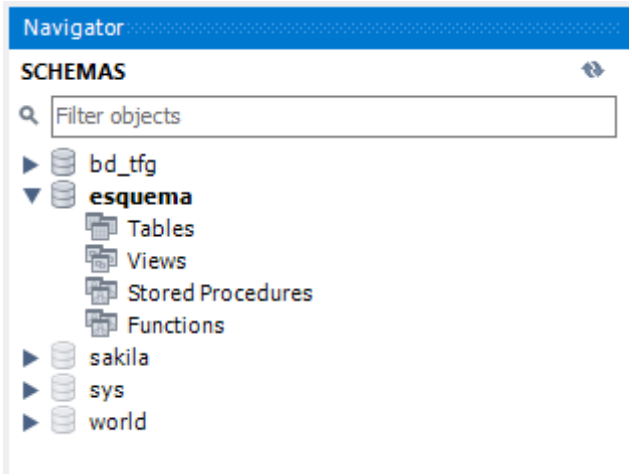

Figura A.14: Esquema creado en el navegador de MySQL

Comprobada la creación de nuestro esquema, es hora de abrir y ejecutar el fichero SQL que contiene nuestra base de datos. Para ello se seguirán los pasos que la figura A.15 muestra, seleccionado la correspondiente carpeta en la que este se encuentre:

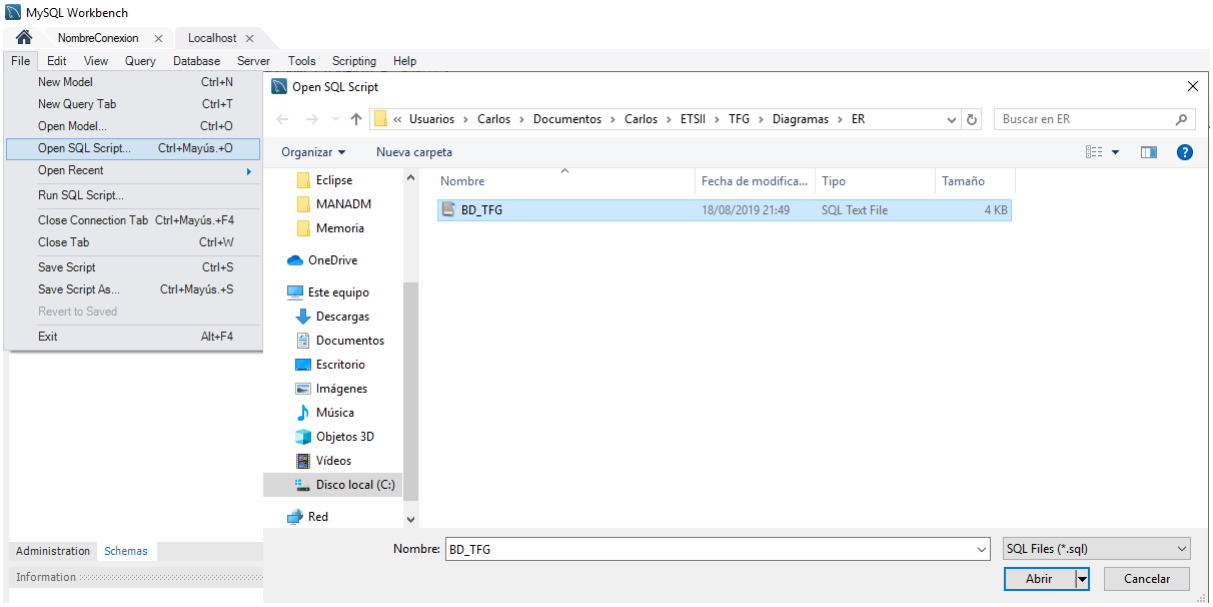

Figura A.15: Selección del fichero con el código de la base de datos

Con nuestro fichero una vez cargado, pulsaremos el botón para ejecutar todo el código como se ve en la figura A.16, creando las correspondientes tablas y relaciones en el esquema de nuestra conexión:

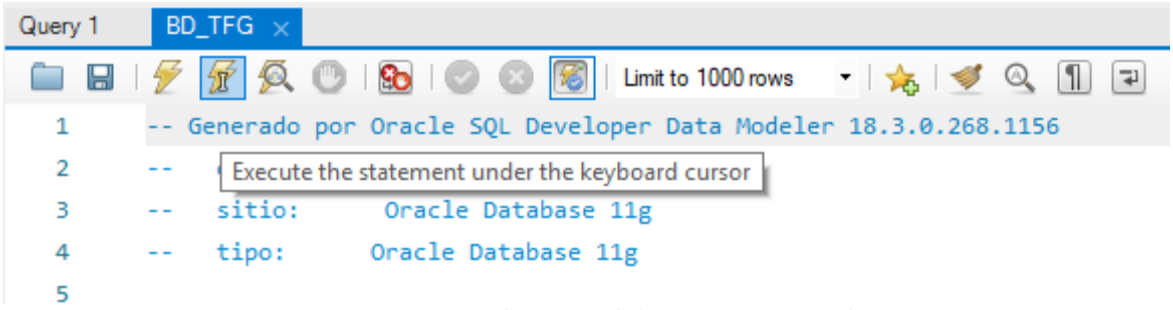

Figura A.16: Fichero cargado listo para ser ejecutado

A.4 Aplicaciones cliente de usuario y administrador

Para realizar el volcado del código de cada una de las aplicaciones cliente, se seguirán los mismos pasos que se detallaron en el apartado A.2, generando un fichero ejecutable .JAR para la aplicación del profesor y otro para la del administrador. Este archivo puede estar en cualquier parte del ordenador siendo recomendable su almacenamiento en el escritorio. Realizados estos pasos hacemos doble clic sobre el ejecutable y la aplicación comenzará a funcionar.

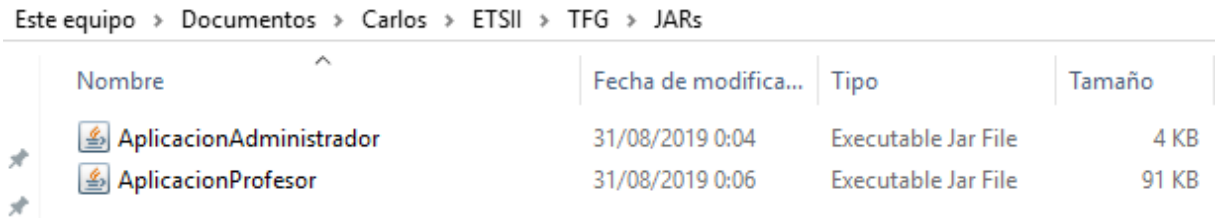

Figura A.17: Ejecutables resultantes de la exportación

Apéndice B
Manual del profesor

En el siguiente apéndice se explicará el uso de la aplicación cliente del profesor, la cual gestionará los despachos que este tenga asignado y los mensajes favoritos que haya guardado.

B.1 Inicio de sesión

Una vez iniciemos la aplicación nos aparecerá una ventana de ingreso al sistema como podemos ver en la figura B.1. Aquí se nos solicitará nuestro usuario, contraseña y la dirección IP junto con el puerto del servidor al que vamos a acceder. Por defecto, nos aparecerán los campos rellenados con ejemplos. Para dejar los campos vacíos, pulsamos el botón *Borrar* y estos pasarán a estar en blanco para podamos introducir nuestros datos correctos.

Figura B.1: Pantalla de autenticación con campos de ejemplo

Una vez tengamos introducidos los distintos campos necesarios para acceder a la aplicación, pulsamos el botón *Entrar* y realizaremos el ingreso a la pantalla principal como se puede apreciar en la figura B.2.

Figura B.2: Pantalla de autenticación con campos rellenos

En caso de no haber ingresado bien algún parámetro, este se nos comunicará mediante un pequeño mensaje de error. En la figura B.3 podemos ver unos ejemplos: señalado en rojo tenemos el mensaje si introducimos mal los credenciales y en verde tenemos el mensaje correspondiente a un error en la conexión con el servidor.

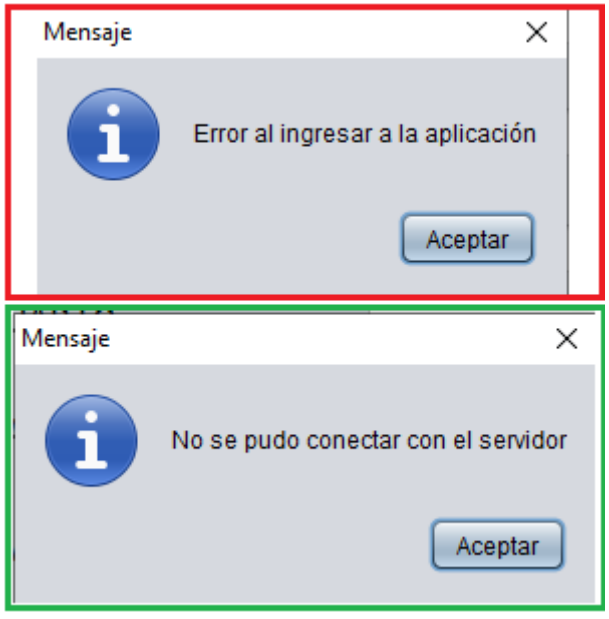

Figura B.3: Posibles mensajes de error

B.2 Selección de despacho

Una vez accedamos a la aplicación nos aparecerá el panel principal con tres secciones como se puede ver en la figura B.4, una a la izquierda, otra a la derecha y la última en la parte inferior.

Figura B.4: Panel principal de la aplicación

Antes de realizar cualquier envío, hemos de asegurarnos de que disponemos de un despacho seleccionado. Para ello, hemos de elegir uno de la lista de despachos que tengamos asignados. El siguiente paso consiste en pulsar el botón *Seleccionar Despacho*, tras haber realizado esto, podremos observar como el rótulo de la zona de envío de mensajes habrá se cambiado por el despacho seleccionado.

B.3 Envío de mensajes al despacho seleccionado

Figura B.5: Selección del despacho al que se van a enviar mensajes

Una vez seleccionado el despacho tal y como se ve en la figura B.5, tenemos dos formas de introducir el mensaje a enviar a la Arduino: escribiéndolo a mano o eligiéndolo de la lista de mensajes favoritos. Vamos a realizar los pasos para enviar un mensaje favorito a la placa.

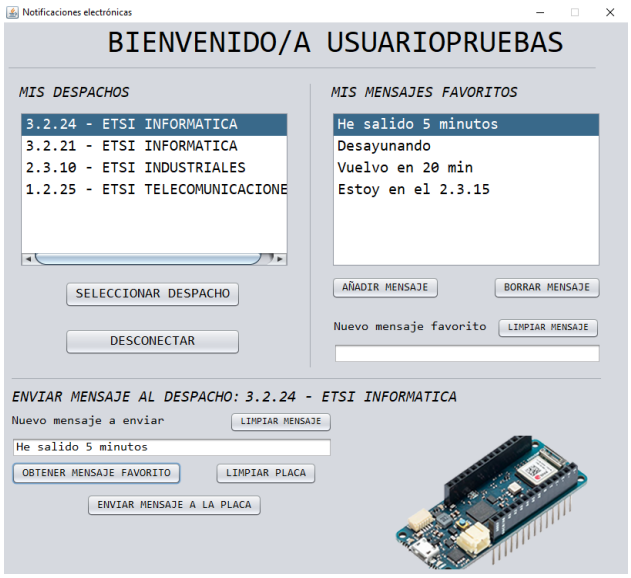

Figura B.6: Selección del mensaje favorito a enviar

Elegimos un mensaje cualquiera de la lista de mensajes favoritos, y una vez seleccionado, en la zona de envíos pulsamos el botón *Obtener Mensaje Favorito*. Realizado este paso, nos debería aparecer el mensaje seleccionado en el campo de texto de envío a la placa tal y como puede apreciarse en la figura B.6.

Una vez escrito el texto que queramos enviar a la placa, pulsamos el botón de *Enviar Mensaje a la Placa* y este se tramitará a esta como se ve en la figura B.7:

Figura B.7: Envío del mensaje a la placa

Cuando el mensaje se haya tramitado por parte del servidor intermedia y de la placa de forma correcta, recibiremos un aviso diciendo que la operación ha sido realizada con éxito como se puede apreciar en la figura B.8. El campo de texto de los mensajes a enviar se pondrá en blanco de forma automática. El botón *Limpiar Mensaje* también pondrá el campo de nuevo mensaje favorito en blanco.

Figura B.8: Éxito en el envío del mensaje a la placa

Por otro lado, si ocurre un error durante el envío del mensaje, obtendremos un aviso que nos dirá que se ha producido un error y que el mensaje no pudo enviarse a la placa. En la figura B.9 podemos ver esta situación:

Figura B.9: Error en el envío del mensaje a la placa

B.4 Gestión de mensajes favoritos

- **Añadir un mensaje favorito**

En la figura B.10 podemos ver el proceso para añadir un nuevo mensaje a la lista de favoritos. Primero hemos de escribirlo manualmente en el campo de texto de nuevos mensajes.

Figura B.10: Rellenado del campo de nuevo mensaje favorito

Acto seguido, pulsamos el botón *Añadir Mensaje* y este aparecerá en la lista de mensajes favoritos en el último lugar de la fila como se ve en la figura B.11.

Figura B.11: Operación de inserción de mensaje satisfactoria

En la figura B.12 podemos ver que en el caso de que se produzca un error, aparecerá una notificación alertándonos de esta situación.

Figura B.12: Error al añadir un nuevo mensaje favorito

- **Borrar un mensaje favorito**

Para borrar los mensajes favoritos procederemos de la misma forma que cuando enviamos un nuevo mensajes, como puede verse en la figura B.13, primero seleccionamos el mensaje que se va a eliminar y luego pulsamos el botón *Borrar*.

Figura B.13: Selección del mensaje favorito a borrar

Tras haber borrado el mensaje seleccionado, obtendremos un aviso confirmando que la eliminación del mensaje ha sido satisfactoria tal y como se aprecia en la figura B.14:

Figura B.14: Operación de borrado de mensaje satisfactoria

Apéndice C
Manual del administrador

En este último apéndice se explicará el uso de la aplicación de administración, la cual será la encargada de la gestión de los profesores, placas y despachos dados de alta en el sistema y de sus posteriores asignaciones.

C.1 Inicio de sesión

De la misma forma que ocurría con la aplicación del profesor, tras iniciar la ejecución nos aparecerá una ventana de ingreso al sistema. Aquí se nos solicitará nuestro usuario, contraseña y la dirección IP junto con el puerto del servidor al que vamos a acceder. Por defecto, y como podemos ver en la figura C.1, nos aparecerán los campos rellenados con ejemplos. Para dejar los campos vacíos, pulsamos el botón *Borrar* y estos pasarán a estar en blanco para podamos introducir nuestros datos correctos.

Figura C.1: Pantalla de autenticación con campos de ejemplo

Una vez tengamos introducidos los distintos campos necesarios para acceder a la aplicación, pulsamos el botón *Entrar* y realizaremos el ingreso a la pantalla principal como se puede apreciar en la figura C.2:

Figura C.2: Pantalla de autenticación con campos rellenos

En caso de no haber ingresado bien algún parámetro, este se nos comunicará mediante un pequeño mensaje de error. En la figura C.3 podemos ver unos ejemplos: señalado en rojo tenemos el mensaje si introducimos mal los credenciales y en verde tenemos el mensaje correspondiente a un error en la conexión con el servidor.

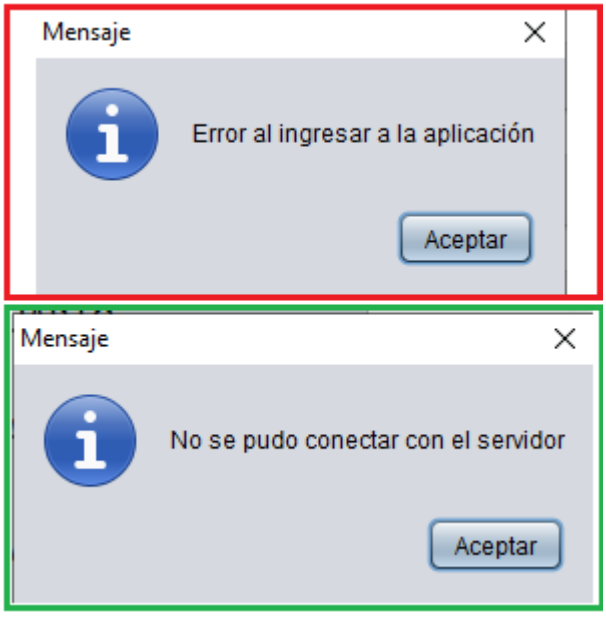

Figura C.3: Posibles mensajes de error

C.2 Administración del profesor

Mostraremos aquí las funciones asociadas a la gestión del profesorado.

- **Añadir un profesor**

Para añadir un profesor a la base de datos hemos de rellenar todos los campos de la zona de profesores como se observa en la figura C.4:

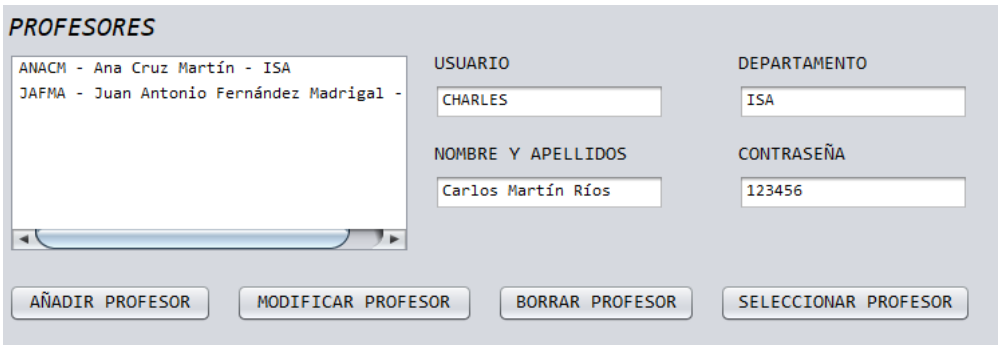

Figura C.4: Parámetros para nuevo profesor

Tras haber realizado el paso anterior, pulsaremos en el botón *Añadir Profesor* y se nos tramitará la operación con la base de datos tal y como se ve en la figura C.5:

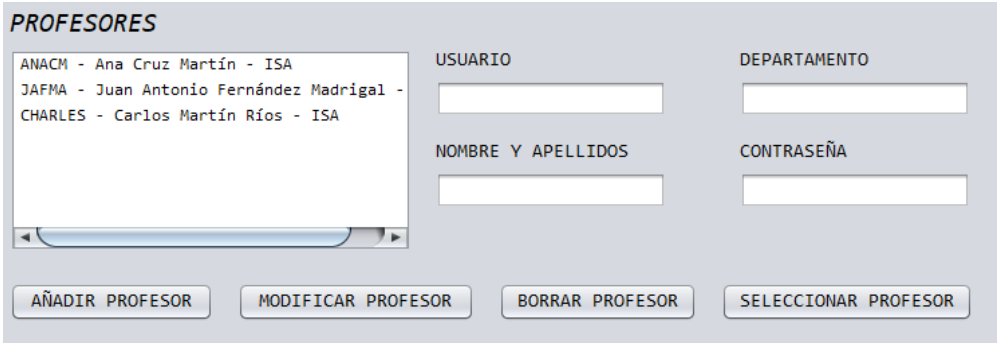

Figura C.5: Nuevo profesor añadido

- **Borrar un profesor**

Para borrar un profesor del sistema procederemos seleccionando dicho profesor de la lista y posteriormente pulsando sobre el botón *Borrar Profesor*, como puede verse en la figura C.6:

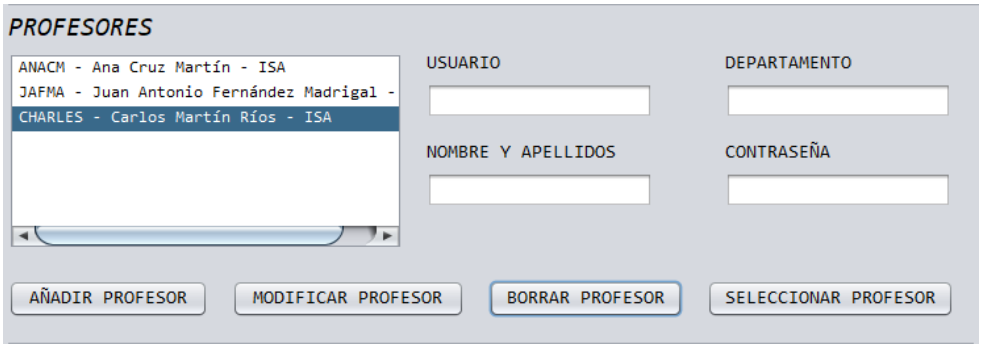

Figura C.6: Selección del profesor a borrar

A continuación en la figura C.7 podemos ver cómo se ha borrado el profesor seleccionado:

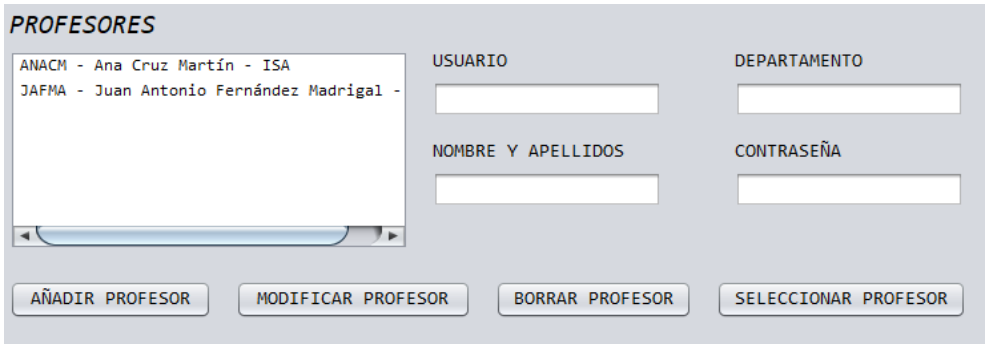

Figura C.7: Borrado de un profesor

- **Modificar un profesor**

Para modificar un profesor realizaremos los siguientes pasos: primero elegiremos un profesor de la lista y posteriormente pulsaremos el botón *Seleccionar Profesor*, una vez seleccionado, cambiaremos los parámetros que deseemos a excepción del nombre de usuario:

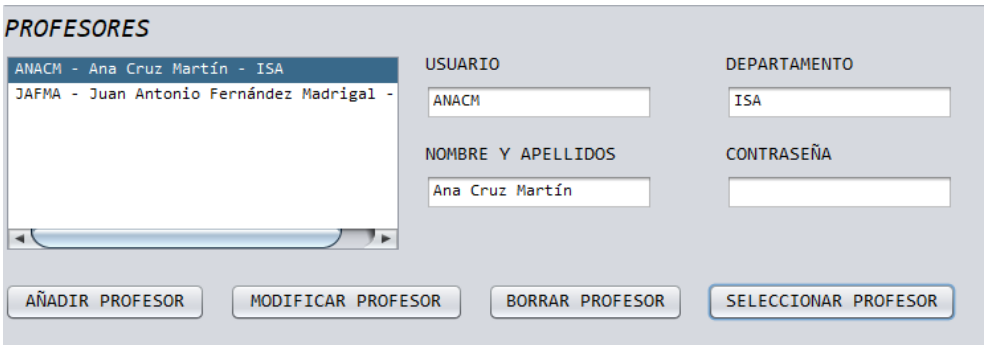

Figura C.8: Selección del profesor a modificar

En la figura C.9 podemos ver cómo se ha modificado el campo correspondiente:

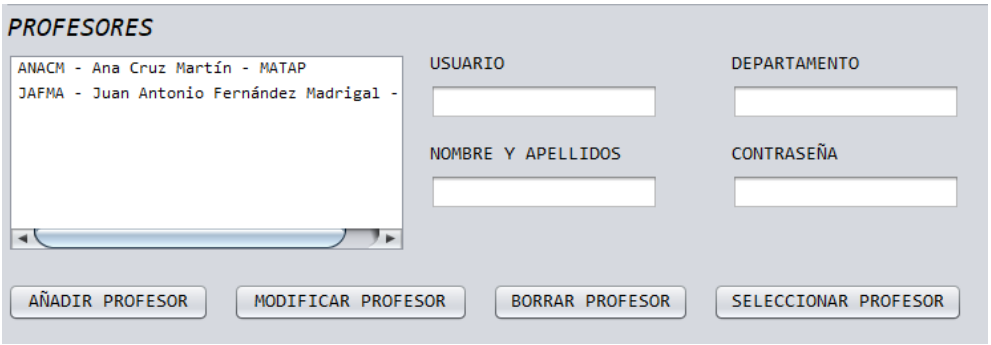

Figura C.9: Modificación del profesor seleccionado

C.3 Administración de las placas

Mostraremos aquí las funciones asociadas a la gestión de las placas del sistema.

- **Añadir una placa**

Actuaremos de la misma forma que para agregar un profesor, rellenamos los campos y pulsamos el botón de *Añadir Placa*:

Figura C.10: Parámetros para nueva placa

En la figura C.11 podemos ver la nueva entrada correspondiente a los datos introducidos en el paso anterior:

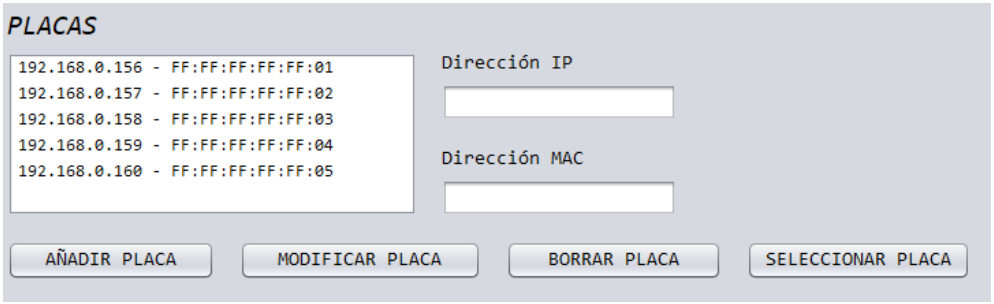

Figura C.11: Nueva placa añadida

- **Borrar una placa**

Seguiremos los mismos pasos que se dieron para borrar un profesor del sistema: primero seleccionamos una entrada de la lista, y pulsaremos el botón *Borrar placa* como se puede observar en la figura C.12:

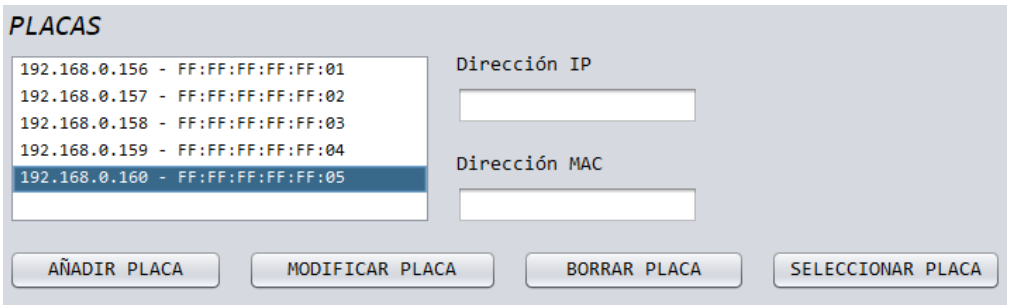

Figura C.12: Selección de la placa a borrar

Dicha instrucción borrará la entrada de la base de datos, tal se aprecia en la figura C.13:

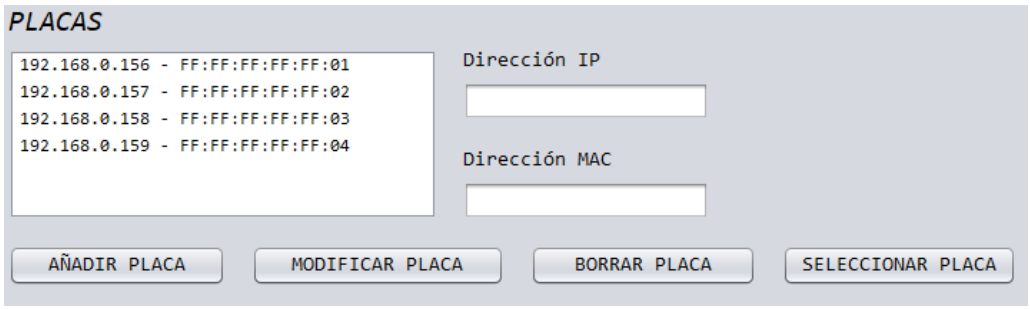

Figura C.13: Borrado de una placa

- **Modificar una placa**

Seleccionamos una placa de la lista y posteriormente pulsamos el botón *Seleccionar Placa* tal y como se ve en la figura C.14:

Figura C.14: Selección de la placa a modificar

Tras haber realizado el paso anterior, pulsamos *Modificar Placa* y la entrada de la lista será modificada con los valores que se introdujeron anteriormente como se puede apreciar en la figura C.15 . Hay que remarcar que no se puede modificar el campo MAC:

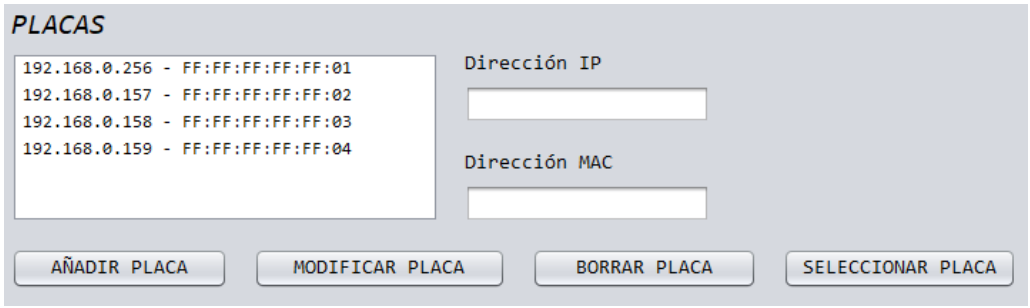

Figura C.15: Placa con la IP modificada

C.4 Administración de los despachos

Mostraremos aquí las funciones asociadas a la gestión de los despachos del profesorado.

- **Añadir un despacho**

Seguiremos el mismo procedimiento que se explicó anteriormente, rellenamos los correspondientes campos primero y pulsamos el botón *Añadir Despacho*:

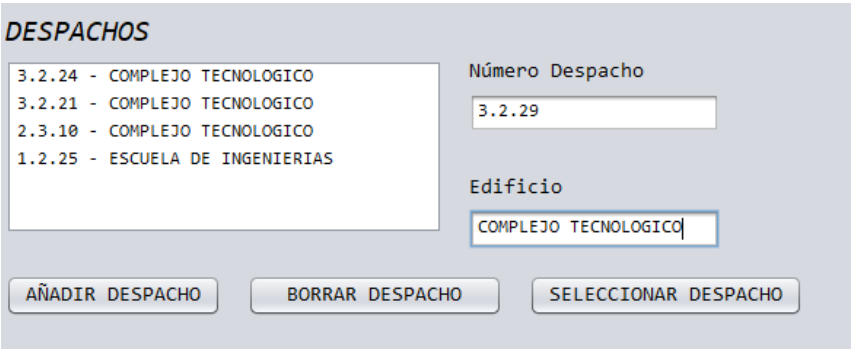

Figura C.16: Parámetros para nuevo despacho

Tras este paso, el nuevo despacho estará en la lista como puede verse en la figura C.17:

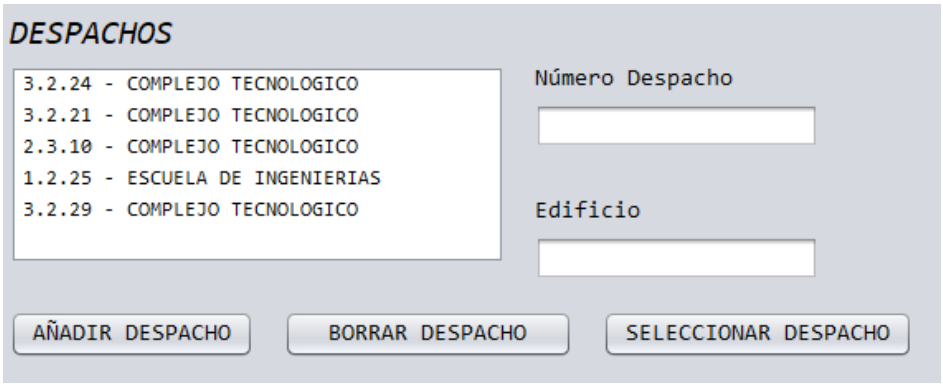

Figura C.17: Nuevo despacho añadido

- **Borrar un despacho**

Seleccionamos el despacho que deseemos eliminar y pulsamos el botón *Eliminar Despacho*.

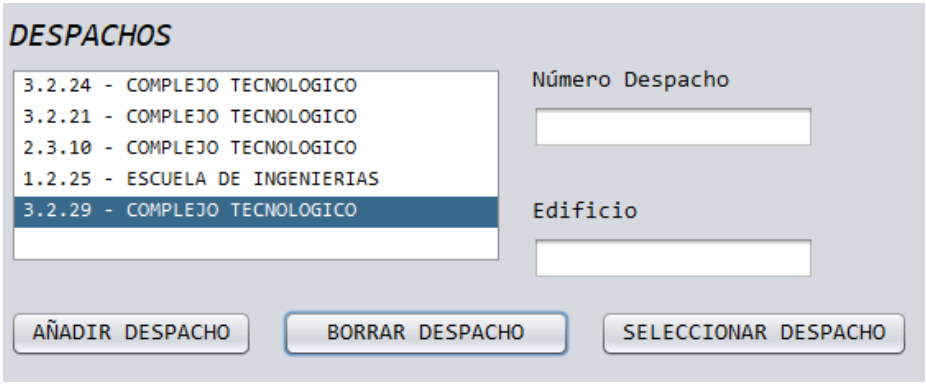

Figura C.18: Selección del despacho a borrar

Dicha entrada será eliminada de la lista como puede verse en la figura C.19:

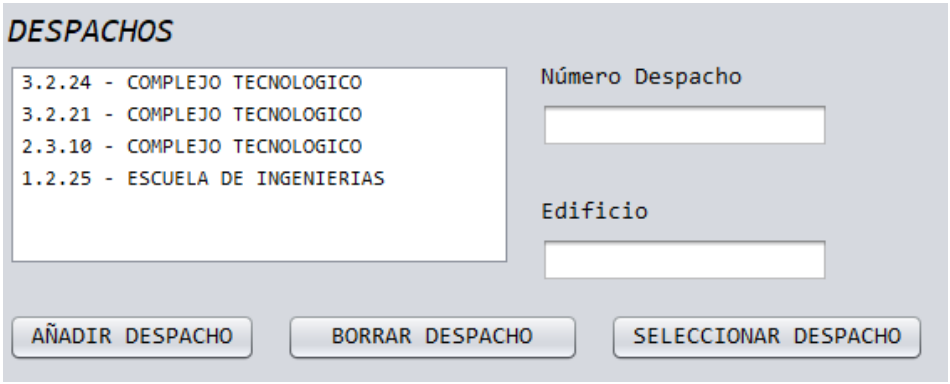

Figura C.19: Borrado de un despacho

C.5 Asignación de una placa a un despacho

Mostraremos aquí las funciones para poder realizar las correspondientes asignaciones entre las placas y los despachos dados de alta en el sistema.

- **Añadir una placar a un despacho**

  La forma más sencilla para realizar este paso es usar los botones de *Copiar MAC* y *Copiar Despacho*. Para ello deberemos seleccionar los valores que deseemos introducir en sus respectivas listas. Una vez elegidos los campos de texto como se aprecia en la figura C.20, pulsaremos el botón *Añadir*:

Figura C.20: Selección de parámetros

Con esto generaremos una nueva fila en la base de datos como se ve en la figura C.21:

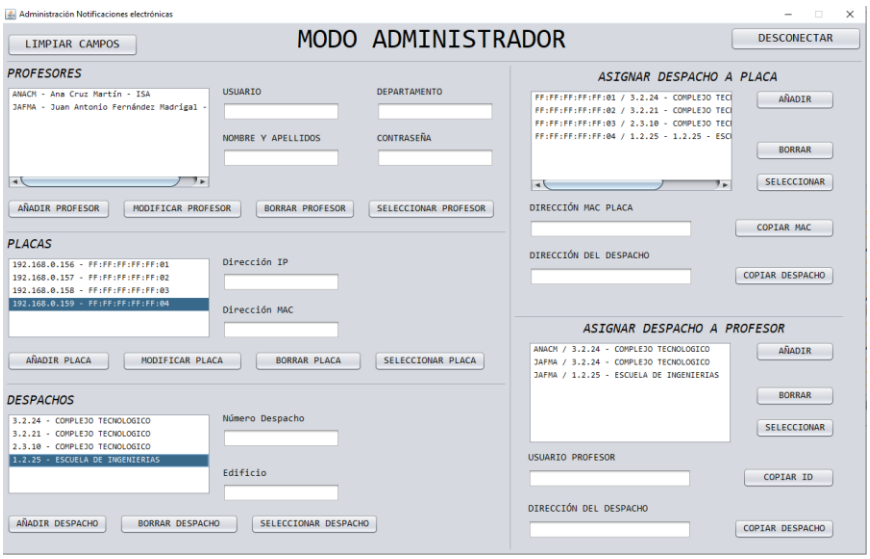

Figura C.21: Nueva asociación despacho-placa

- **Eliminar una placa de un despacho**

Para eliminar una asociación, seleccionaremos esta de la lista y posteriormente pulsaremos el botón *Borrar*:

Figura C.22: Selección asociación despacho-placa

Tras realizar este paso, la fila será borrada de la base de datos como puede verse en la figura C.23:

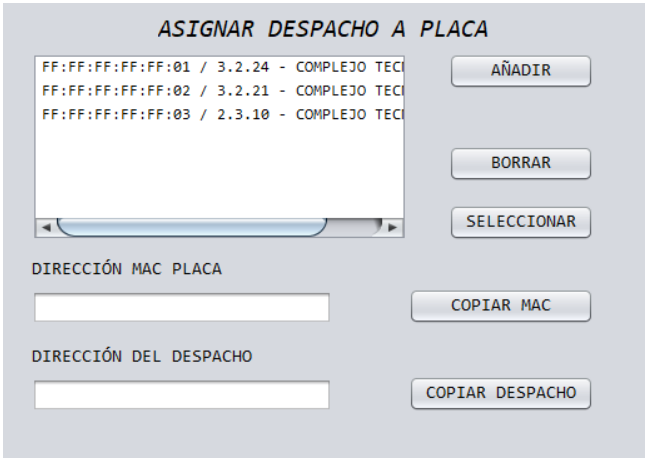

Figura C.23: Borrado de la asociación despacho-placa

C.6 Asignación de un profesor a un despacho

Mostraremos aquí las funciones para poder realizar las correspondientes asignaciones entre las placas y los despachos dados de alta en el sistema.

- **Añadir un profesor a un despacho**

  Realizaremos el mismo procedimiento que para asociar una placa a un despacho. Usaremos los botones *Copiar ID* y *Copiar Despacho* para facilitar el proceso de añadido a la base de datos tal y como se ve en la figura C.24:

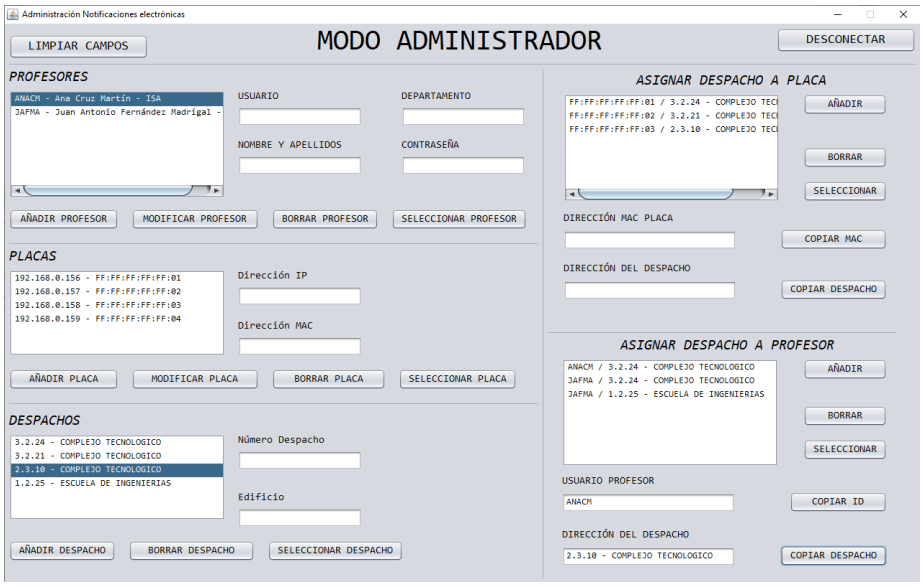

Figura C.24: Selección de parámetros

Realizado este paso, tendremos una nueva entrada en la base de datos:

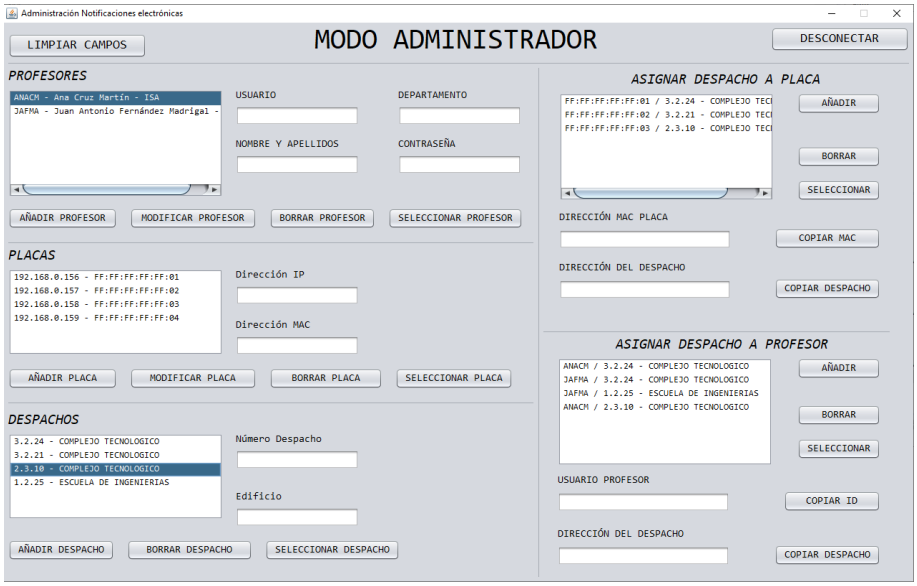

Figura C.25: Nueva asociación despacho-profesor

- **Eliminar un profesor de un despacho**

Siguiendo los mismo pasos que en el anterior punto, seleccionaremos la asociación a borrar y pulsaremos el botón *Borrar* como puede apreciarse en la figura C.26:

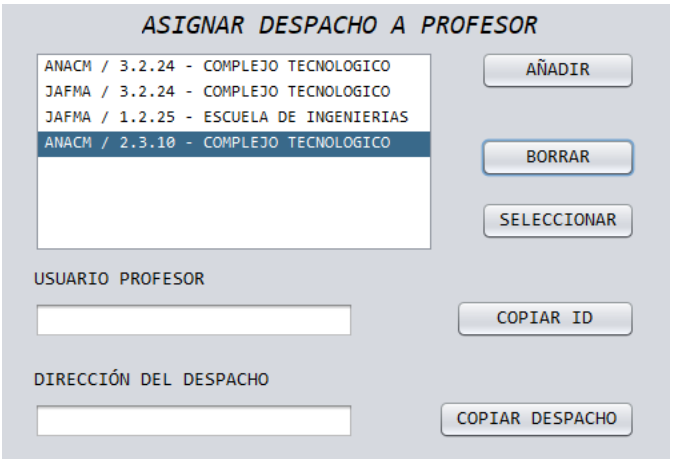

Figura C.26: Selección asociación despacho-profesor

Esta borrará la correspondiente fila en la base de datos que se le ha pasado como parámetro:

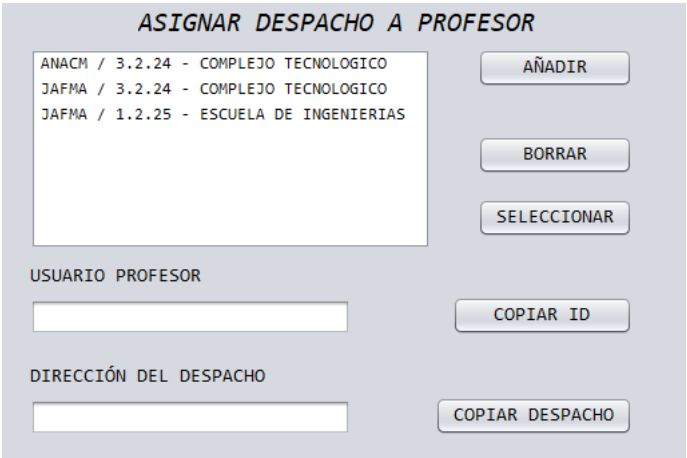

Figura C.27: Borrado de la asociación despacho-profesor

7

Bibliografía

[1] Arduino. Introduction. 2019,

Sitio web: https://www.arduino.cc/en/Guide/Introduction

[2] Wikipedia. Desarrollo en cascada. 2019,

Sitio web: https://es.wikipedia.org/wiki/Desarrollo_en_cascada

[3] Foros de Arduino. An arduino based doorbellmessaging system for your lab. 2017,

Sitio web: https://blog.arduino.cc/2017/06/29/an-arduino-based-doorbellmessaging-system-for-your-lab/

[4] Arduino. Arduino IDE. 2019,

Sitio web: https://www.arduino.cc/en/Main/Software

[5] Arduino. WiFiNINA Reference. 2019,

Sitio web: https://www.arduino.cc/en/Reference/WiFiNINA

[6] Arduino. LiquidCrystal Reference. 2019,

Sitio web: https://www.arduino.cc/en/Reference/LiquidCrystal

[7] Wikipedia. Eclipse (software). 2019,

Sitio web: https://es.wikipedia.org/wiki/Eclipse_(software)

[8] Wikipedia. Fritzing. 2018,

Sitio web: https://es.wikipedia.org/wiki/Fritzing

[9] Fritzing. Descarga de Fritzing. 2019,

Sitio web: http://fritzing.org/download/

[10] ORACLE. Java. 2019,

Sitio web: https://www.java.com/es/download/

[11] No Magic. Magic Draw UML. 2019,

Sitio web: https://www.nomagic.com/products/magicdraw

[12] ORACLE. MySQL. 2019,

Sitio web: https://www.mysql.com/

[13] APACHE. NetBeans IDE. 2019,

Sitio web: https://netbeans.apache.org/

[14] ORACLE. ORACLE Datamodeler. 2019,

Sitio web: https://www.oracle.com/database/technologies/appdev/datamodeler.html

[15] Arduino. Arduino UNO REV3 Tech Specs. 2019,

Sitio web: https://store.arduino.cc/arduino-uno-rev3

[16] Arduino. Arduino Genuino Zero Tech Specs. 2019,

Sitio web: https://store.arduino.cc/genuino-zero

[17] Arduino. Arduino MEGA 2560 R3 Tech Specs. 2019,

Sitio web: https://store.arduino.cc/mega-2560-r3

[18] Arduino. Arduino MKR 1000 WiFi Tech Specs. 2019,

Sitio web: https://store.arduino.cc/arduino-mkr1000

[19] Arduino. Arduino MKR 1010 WiFi Tech Specs. 2019,

Sitio web: https://store.arduino.cc/mkr-wifi-1010

[20] Sparkfun. Datasheet GDM1602K. 2004,

Sitio web: https://www.sparkfun.com/datasheets/LCD/GDM1602K.pdf

[21] Sparkfun. Datasheet Serial LCD. 2004,

Sitio web: https://www.sparkfun.com/datasheets/LCD/SerLCD_V2_5.PDF

[22] Wikipedia. I2C. 2019,

Sitio web: https://es.wikipedia.org/wiki/I%C2%B2C

[23] Wikipedia. Casos de uso. 2019,

Sitio web: https://es.wikipedia.org/wiki/Caso_de_uso

[24] Wikipedia. Modelo entidad-relacion. 2019,

Sitio web: https://es.wikipedia.org/wiki/Modelo_entidad-relación

[25] Juan Antonio de la Puente. Sistemas de tiempo real basados en ejecutivos ciclicos. 2000,

Sitio web: https://studylib.es/doc/4947242/sistemas-de-tiempo-real-basados-en-ejecutivos-cíclicos

[26] Wikipedia. Cyclic executive. 2019,

Sitio web: https://en.wikipedia.org/wiki/Cyclic_executive

[27] Wikipedia. Modelo-Vista-Controlador. 2019,

Sitio web: https://es.wikipedia.org/wiki/Modelo–vista–controlador

[28] Wikimedia. ModelViewControllerDiagram2007, Sitio web:

https://commons.wikimedia.org/wiki/File:ModelViewControllerDiagram_es.svg

[29] Wikipedia. Socket de Internet. 2019,

Sitio web: https://es.wikipedia.org/wiki/Socket_de_Internet

[30] Wikipedia. Java Database Connectivity. 2019,

Sitio web: https://es.wikipedia.org/wiki/Java_Database_Connectivity

[31] StackOverflow. Create a class to connect to any database using JDBC. 2012,

Sitio web: https://stackoverflow.com/questions/10915375/create-a-class-to-connect-to-any-database-using-jdbc