CURSO 2018/19

proyecto de final de grado superior por

Rubén Balsera de la cuesta

antonio illarramendi rubio

valentín sanchez BOTO

COORDINADO por

ANTONIO SEPÚLVEDA RUíZ

**APLICACIÓN**

**DAMNÍFICUS**

IES GASPAR MELCHOR DE JOVELLANOS | DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Contenido

[AGRADECIMIENTOS 5](#_Toc10659492)

[RESUMEN 6](#_Toc10659493)

[ANÁLISIS DE LA APLICACIÓN 7](#_Toc10659494)

[Datos técnicos 7](#_Toc10659495)

[Catálogo de requisitos 8](#_Toc10659496)

[Requisitos funcionales 8](#_Toc10659497)

[Requisitos no funcionales 10](#_Toc10659498)

[METODOLOGÍA DE DESARROLLO 11](#_Toc10659499)

[Contenido de cada prototipo 12](#_Toc10659500)

[Base de datos 12](#_Toc10659501)

[Seguridad 12](#_Toc10659502)

[DISEÑO 13](#_Toc10659503)

[Arquitectura de la aplicación 13](#_Toc10659504)

[Modelo a capas 14](#_Toc10659505)

[**Vista** 15](#_Toc10659506)

[**Modelo** 16](#_Toc10659507)

[**Persistencia** 16](#_Toc10659508)

[Tecnologías utilizadas 17](#_Toc10659509)

[Diseño de la interfaz 18](#_Toc10659510)

[Primer prototipo de pantalla 18](#_Toc10659511)

[Segundo prototipo de pantalla 18](#_Toc10659512)

[Prototipo final de pantalla 19](#_Toc10659513)

[Diseño de las pantallas 20](#_Toc10659514)

[**Pantalla principal** 20](#_Toc10659515)

[**Pantalla Registro** 21](#_Toc10659516)

[**Pantalla Área cursos** 23](#_Toc10659517)

[**Pantalla Área Personal** 28](#_Toc10659518)

[**Pantalla Subida de enlaces** 31](#_Toc10659519)

[**Pantalla ayuda** 33](#_Toc10659520)

[**Pantalla administración** 35](#_Toc10659521)

[**Mensajes de dialogo** 39](#_Toc10659522)

[Diseño de la base de datos 40](#_Toc10659523)

[Modelo Entidad-Relación 40](#_Toc10659524)

[Diseño final base de datos 41](#_Toc10659525)

[Diseño de la seguridad 42](#_Toc10659526)

[Diagrama de clases 42](#_Toc10659527)

[Conclusiones 45](#_Toc10659528)

[IMPLEMENTACIÓN 46](#_Toc10659529)

[Parte cliente 46](#_Toc10659530)

[Elementos básicos 46](#_Toc10659531)

[Capa Acceso a datos 46](#_Toc10659532)

[Capa de Negocio 47](#_Toc10659533)

[Gestión de panel principal 47](#_Toc10659534)

[Gestión de *login* de usuarios 48](#_Toc10659535)

[Gestión de registro de usuarios 48](#_Toc10659536)

[Gestión área personal del usuario 49](#_Toc10659537)

[Gestión de área cursos 51](#_Toc10659538)

[Gestión de subida de enlaces 53](#_Toc10659539)

[Gestión de área ayuda 54](#_Toc10659540)

[Gestión de área administración 55](#_Toc10659541)

[Parte servidor 58](#_Toc10659542)

[Programa servidor 58](#_Toc10659543)

[Envío de email 61](#_Toc10659544)

[Generador de token 62](#_Toc10659545)

[Base de datos 62](#_Toc10659546)

[Escritura en consola 62](#_Toc10659547)

[Archivos de configuarción 63](#_Toc10659548)

[Seguridad y encriptación 65](#_Toc10659549)

[Conclusiones 66](#_Toc10659550)

[PRUEBAS 66](#_Toc10659551)

[MANUAL DE USO 68](#_Toc10659552)

[CONCLUSIONES 72](#_Toc10659553)

[RECURSOS MATERIALES 72](#_Toc10659554)

[BIBLIOGRAFÍA 73](#_Toc10659555)

# AGRADECIMIENTOS

|  |
| --- |
| A **Antonio Sepúlveda** por ser nuestro tutor de proyecto, enseñarnos el framework .NET y su potencia y que estemos pensando siempre en que errores vamos a cometer antes de siquiera escribir una sola línea de código.  A **Miguel Ángel Muñoz** por sus tremendas *masterclasses* y su empeño en enseñarnos todos los secretos de la creación de buenas bases de datos y el correcto funcionamiento de los programas y sus hilos para que todo esto funcione.  A **Saturnina Castro** por otorgarnos sus conocimientos de JAVA que nos han ayudado a transformar nuestro conocimiento de *VisualBasic* a C#.  A **Javier Palacios** por su buen hacer, aun siendo su primer año en la asignatura, aprendimos mucho sobre *Linux*, *Python* y algún que otro buen consejo profesional.  A **Pablo Borrella** por su insistencia con inglés que nos ha ayudado a descifrar todas esas palabras de documentación que hemos revisado durante horas.  A **Carmen Illán** por sus clases no exentas de interesantes debates sobre los más variados temas de actualidad.  También mencionar a **Pilar Fernández** y **Pilar Rubio** que, aunque no nos hayan dado clase el último año también procuraron enseñarnos y formarnos como profesionales del sector de la informática y la programación.  Finalmente, no queremos olvidarnos de **Alberto**, quien todos los días nos preparaba un buen café para empezar con fuerza la mañana y nos daba ánimos antes de entrar al aula. |
|  |



# RESUMEN

Durante los dos años en los que hemos cursado el ciclo de Desarrollo de Aplicaciones Multiplataforma hemos aprendido mucho sobre el mundo de la informática y la programación.

Como ya hemos comprobado el temario del curso es muy variado y completo, son varios los lenguajes de programación que se abordan, las asignaturas orbitan unas en relación con otras, y el material necesario para abordar los ejercicios y exámenes suele ser muy abundante.

Además, habría que contar las horas invertidas en buscar información por parte de los alumnos en otros lugares ajenos al centro, como foros, tutoriales online u otros programadores.

Es por lo que tras debatir los tres autores de este proyecto entre varias ideas que teníamos, nos acabamos decantando por una aplicación en la que pudiéramos concentrar todo el conocimiento y materiales que habíamos recopilado a lo largo del curso, de forma ordenada y cómoda, para que cualquier otro alumno en un futuro pudiera beneficiarse de toda esa hemeroteca.

Por supuesto no solo se trata de una simple “enciclopedia” donde consultar información. El objetivo es que los usuarios puedan ser partícipes también dándoles la posibilidad de subir sus propios documentos a la aplicación, donde el resto de los usuarios podrán valorar ese contenido.

Además, los usuarios que se registren tendrán también la posibilidad de usar un área personal donde podrán gestionar sus enlaces subidos, sus notas en cada trimestre y asignatura del curso, medias, etc.

Todo ello en una aplicación sólida, segura y muy centrada en la manejabilidad, sencillez y una buena experiencia de usuario.

# ANÁLISIS DE LA APLICACIÓN

En este capítulo se presenta la fase de análisis, parte inicial de todo proyecto software, se definen los requisitos y se muestra una vista global de la arquitectura pensada para el sistema. En el siguiente capítulo se tendrá en cuenta este catálogo de requisitos como base para el diseño de todos los aspectos de la aplicación. Es por ello por lo que la fase de análisis es de suma importancia para el devenir de todo producto software. Es en este capítulo donde se deben asentar las bases, a modo de cimientos, del proyecto, y, a partir de las cuales se construirá todo lo demás.

El proceso que seguir se basa en primer lugar en una buena definición de requisitos y en elegir una metodología de desarrollo acorde con el proyecto.

La arquitectura de la aplicación tendrá el formato de cliente-servidor, constituyendo el servidor la base de datos, y API's de acceso, y cliente, la aplicación de escritorio.

Este capítulo es fundamental para entender el proyecto. A continuación, se desglosan los datos técnicos, la funcionalidad y las características a modo de catálogo de requisitos, teniendo en cuenta tanto requisitos funcionales como no funcionales.

## Datos técnicos

**Nombre de la aplicación:** Aplicación DAMníficus

**Plataforma:** Microsoft Windows

**Lenguaje de programación:** C# con el Framework .NET

**Requisitos mínimos del Sistema:**

* Procesador: Intel Pentium Dual Core 1,6 GHz
* Memoria: 256 MB RAM
* Tarjeta gráfica: integrada
* Espacio de almacenamiento: 5MB
* Pantalla con resolución 800x600 píxeles

**Manual de instalación:** Para hacer uso de la aplicación se debe abrir el archivo ejecutable correspondiente al cliente, es *portable*, por lo que no hace falta instalación alguna.

Además, cabe destacar que para que la aplicación funcione correctamente se ha de ejecutar el servidor, es posible tanto en la misma máquina cliente, como en otro ordenador.

El servidor es una aplicación de comandos de Windows y está programado en el mismo lenguaje que el cliente.

Por último, es necesario tener una base de datos MySQL, esta Base de Datos debe estar instalada y funcionando.

## Catálogo de requisitos

El catálogo de requisitos es la especificación del comportamiento que se espera de cualquier proyecto software. Estudiando otras aplicaciones similares, se ha predefinido una serie de requisitos que se consideran indispensables para el proyecto. A continuación, se muestra una enumeración y breve descripción de los requisitos establecidos para el diseño y desarrollo de la aplicación.

## Requisitos funcionales

Los requisitos funcionales describen todas las interacciones que tendrán los usuarios con el software.

GESTIÓN DE USUARIOS

##### RF1: Registro

* + - * 1. La aplicación debe permitir al usuario introducir sus datos en el formulario de registro.
        2. El sistema se encargará de validar los datos.
        3. El sistema mostrará un mensaje de error si alguno de los datos es incorrecto o no cumple las condiciones especificadas del formulario de registro.
        4. En el caso de que la validación sea correcta, el sistema se encargará de guardar los datos del usuario en base de datos.
        5. El sistema enviará un correo electrónico de confirmación al usuario en el caso de que el registro se haya efectuado correctamente.
        6. La aplicación mostrará un mensaje de bienvenida al usuario y le redirigirá a la pantalla principal con su sesión ya iniciada.

##### RF2: Identificación

1. Para iniciar sesión el usuario deberá identificarse con su nombre de usuario y contraseña correspondiente. En el momento de la identificación se generará un token único de sesión para validar las peticiones siguientes.
2. El sistema se encargará de validar y permitir o denegar el acceso a la aplicación.
3. El sistema mostrará un mensaje de error en el caso de que la validación no sea correcta.
4. En el caso de que la validación sea correcta, se mostrará un mensaje de bienvenida al usuario y le redirigirá a la pantalla principal

##### RF3: Cierre de sesión

1. Cualquier usuario de la aplicación debe poder finalizar sesión en la aplicación mediante un botón de que indique "Cierre de sesión".
2. En el caso de que el usuario cierre la aplicación, el sistema mostrará un mensaje, para asegurarse de que el usuario quiere cerrar sesión, permitiendo aceptar o cancelar.
3. El usuario será redirigido a la pantalla principal, pero en este caso sin estar logado.

##### RF4: Área personal de usuario

1. Debe existir una pantalla en el sistema que permita al usuario consultar o modificar los datos de su cuenta.
2. Si modifica algún dato el sistema lo validará.
3. En caso de que la validación sea correcta, se actualizarán sus datos en la base de datos.

##### RF5: Enlaces del usuario

**1.** Debe existir una pantalla en la que el usuario pueda consultar los enlaces que él mismo haya subido al programa.

APLICACIÓN

##### RF6: Pantalla principal

* + - * 1. La pantalla principal de la aplicación constará del mensaje de bienvenida y el acceso al registro si el usuario todavía no se ha logueado.

##### RF7: Menú lateral

1. El menú lateral mostrará la opción de acceder a:
   1. **Principal**: Formulario de bienvenida para registrar un nuevo usuario en la aplicación.
   2. **Cursos**: Pantalla para ver los enlaces ordenados por cursos.
   3. **Área personal**: Datos de cuenta de usuario como notas o enlaces subidos por él mismo.
   4. **Subida de enlaces**: Formulario para subir nuevos enlaces.
   5. **Ayuda**: Pantalla de información donde recuperar la contraseña si se ha perdido, cambiarla o reportar bug o sugerencia.
   6. **Administración**: Este panel se muestra en lugar de el de ayuda si el usuario tiene credenciales de administrador. Aquí se pueden gestionar tanto los usuarios como los enlaces subidos por ellos.
   7. **Salir**: Botón para salir de la aplicación.

##### RF8: Búsqueda de enlaces

**1.** En todas las pantallas de enlaces el usuario podrá usar un filtro de búsqueda en tiempo real.

## Requisitos no funcionales

Requisitos complementarios o atributos de calidad. Especifican criterios que juzgan operaciones del sistema en lugar de su comportamiento (requisitos funcionales).

##### RNF1: Documentación

1. Página con servicio de ayuda online.
2. Manual de usuario de la aplicación.
3. La codificación del sistema deberá ser clara y estar documentada de manera que algún programador pueda agregar funcionalidad posteriormente, procurando seguir los estándares de programación en C#.

##### RNF2: Seguridad

1. Para poder utilizar todas las funciones de la aplicación hay que autentificarse.
2. Solo será necesario autentificarse una vez en el dispositivo.
3. Si el usuario no cierra su sesión al salir, ésta se cerrará automáticamente.
4. Los datos personales serán cifrados.

##### RNF3: Mantenibilidad y portabilidad

1. Disponibilidad en todo momento de un contacto con los administradores.
2. La aplicación escritorio es portable y no necesita instalación.
3. Será necesario disponer de una conexión a internet, en caso de que el servidor se ejecute en una red externa.

##### RNF4: Interfaz y usabilidad

1. La aplicación debe constar de una interfaz sencilla, atractiva e intuitiva. De tal forma que su uso no suponga un impedimento o esfuerzo al usuario a la hora hacer uso de la aplicación.
2. La introducción de datos debe estar estructurada procurando evitar errores.

##### RNF5: Rendimiento

1. Se esperan tiempos de respuesta no superiores a un segundo en las peticiones al servidor y menores en las consultas a la base de datos.
2. Tanto los accesos a base de datos como algún cálculo que se realice en la aplicación no suponen demasiada carga para los equipos actuales, por lo que el rendimiento será óptimo.

# METODOLOGÍA DE DESARROLLO

Se trata de un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo del software. Existe una gran cantidad de métodos diferenciados por sus fortalezas y debilidades. Una metodología de desarrollo software se basa en:

*Herramientas, modelos y métodos para ejecutar dicho proceso de desarrollo*

Cada metodología de desarrollo tiene su propio enfoque. Entre ellos, para el proyecto DAMníficus, el enfoque de prototipado parece ser el más apropiado, debido a que se centra en la experiencia con usuario y a que su construcción debe ser en poco tiempo.

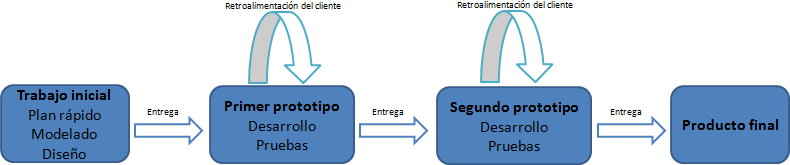
El modelo de prototipos pertenece a los modelos de desarrollo evolutivo, el prototipo debe ser construido en poco tiempo y sin utilizar muchos recursos. El diseño rápido se centra en representación de aspectos software visibles para el usuario. Este rápido diseño conduce a la creación de un primer prototipo que será evaluado y retroalimentado por el cliente, gracias al cual el equipo de desarrollo podrá entender mejor lo que se debe hacer y permitiendo ver al cliente resultados progresivos a corto plazo.

Las diferentes etapas por la que pasa un software construido con este tipo de metodología son:

* Plan rápido
* Modelado, diseño rápido
* Desarrollo
* Entrega
* Retroalimentación
* Comunicación
* Entrega final

Este modelo es muy útil cuando el cliente conoce perfectamente los objetivos generales del software, pero no detalla los requisitos.

A continuación, se ilustra la metodología de desarrollo utilizada en este proyecto mediante un gráfico sencillo.



## Contenido de cada prototipo

En el primer prototipo se crea el servidor y la interfaz de usuario principal de la aplicación. Además, la funcionalidad de iniciar sesión en la aplicación estaba disponible.

En el segundo prototipo se implementaron el registro de usuario, las áreas de ayuda/administración con su funcionalidad completa y su conexión al servidor. Se comienza el desarrollo del área personal, el área. También se implementan las peticiones necesarias en el lado del servidor.

Se crea un proyecto más para albergar una biblioteca de clases con las entidades compartidas.

En el tercer prototipo se finalizaron el área personal y el área cursos. En el área cursos se añaden las búsquedas y se mostrarán los resultados al usuario.

En el último prototipo se mejora lo anterior corrigiendo fallos y se refinan las interfaces por si no quedaron lo suficientemente intuitivas…

Antes de cada entrega de prototipo se realizan una serie de pruebas tanto de usabilidad, como de carga de datos. Además, se prueba la conexión entre el servidor y varios clientes al mismo tiempo.

## Base de datos

Es obvio que la base de datos no puede ser local en el propio equipo, por lo que se implementa una base de datos externa situada en el servidor y a la que se accede a través del API del servidor.

La base de datos almacenará toda la información de usuarios, notas, asignaturas, temas o enlaces, se explicará más adelante en el capítulo respectivo al diseño.

## Seguridad

Existirá una doble capa de seguridad para evitar cierto tipo de ataques. La aplicación se comunicará con el servidor por medio de llamadas HTTP para enviar y recibir información. Esta información puede verse en peligro ante ataques *Man in the Middle*, por ejemplo. Si el atacante consigue interceptar las cabeceras enviadas al servidor podría obtener datos muy valiosos como contraseñas y nombres de usuario.

La forma más rápida de evitar estos ataques es cifrar los datos que se intercambian entre servidor y cliente, así como cifrar las contraseñas, para que en el caso de que un atacante consiguiera los datos, no le sean de utilidad. En el caso de la aplicación los datos estarán cifrados, en el caso de las contraseñas el cifrado es doble, con el fin de evitar semejantes ataques.

# DISEÑO

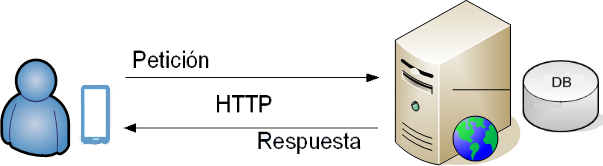
En el presente capítulo se describe todo el proceso de diseño de la aplicación DAMníficus. Se ha realizado un diseño que abarca todos los requisitos descritos en el apartado [**Catálogo de requisitos**](#_bookmark19). El diseño proporciona una idea completa del software desarrollado en el proyecto. Además, se justifican las decisiones tomadas para el posterior desarrollo. Para ello, el capítulo se divide en [Arquitectura de la](#_bookmark31) [aplicación](#_bookmark31), [Tecnologías utilizadas](#_bookmark33), [Diseño de pantallas](#_bookmark42), [Diseño de la interfaz](#_bookmark47), [Diseño de la base de datos](#_bookmark57), [Diseño de scripts y procedimientos almacenados](#_bookmark63) y [Diseño de la seguridad](#_bookmark64).

El diseño de la aplicación ha ido sufriendo algún cambio a lo largo del desarrollo. En consecuencia, se muestran en este documento varias versiones de algunos diagramas de diseño de este. A continuación, se enumeran los diferentes motivos que han implicado dichas modificaciones:

1. Pruebas no satisfactorias, errores, dificultades o no satisfacción de todos los requisitos.
2. Ampliación o adición de algún requisito.
3. Adquisición de nuevos conocimientos.

## Arquitectura de la aplicación

Como se menciona en el capítulo de Análisis, la arquitectura de la aplicación se basa en el modelo cliente servidor. En el caso de este proyecto, y como puede verse en la siguiente figura, la aplicación de escritorio constituirá la parte cliente, y la base de datos estará alojada en un servidor al que se accederá mediante llamadas http.



Para el proyecto DAMníficus era necesario una base de datos centralizada, y no local en el equipo, por lo que la separación entre cliente y servidor se convierte en más que lógica en este caso.

El cliente es quien inicia las solicitudes, teniendo un papel activo en la comunicación, y espera la respuesta del servidor.

En el caso de este proyecto, y como es común en modelos de este tipo, el cliente interactúa con el usuario a través de una interfaz gráfica.

Por otro lado, el receptor de la solicitud, el servidor, espera a que lleguen solicitudes de los clientes, desempeñando un papel pasivo en la comunicación. Tras la recepción, procesa y envía los datos al cliente.

Las ventajas que supone dicha separación son notables, ventajas de tipo organizativo debidas a la centralización de la gestión de la información y separación de responsabilidades, lo que clarifica y facilita el sistema. La escalabilidad y encapsulación son otras de las ventajas más destacables de este modelo, permitiendo aumentar funcionalidades o recursos de ambos por separado.

En el apartado de Diseño de pantallas de este capítulo se ilustran accesos al servidor mediante un diagrama de pantallas.

## Modelo a capas

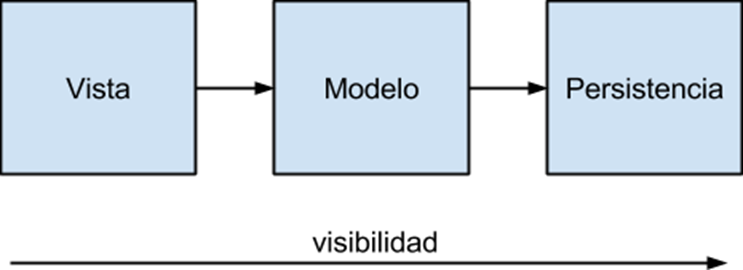
La aplicación cliente ha sido diseñada siguiendo el modelo por capas. La programación por capas es una arquitectura cuyo objetivo primordial es la separación de la lógica de negocios de la lógica de diseño. un ejemplo básico de esto consiste en separar la capa de datos de la capa de presentación al usuario.

La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que haya que hacer algún cambio, sólo se ataca al nivel requerido.

El diseño más utilizado actualmente y el que hemos usado en nuestra aplicación es el diseño en tres capas:

1. **Capa de presentación o vista:** es la que ve el usuario, presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). También es conocida como interfaz gráfica y debe tener la característica de ser entendible y fácil de usar para el usuario. Esta capa se comunica únicamente con la capa de negocio o modelo.
2. **Capa de negocio o modelo:** es donde se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio o modelo y es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de [base de datos](http://www.google.com/url?q=http%3A%2F%2Fes.wikipedia.org%2Fwiki%2FBase_de_datos&amp;sa=D&amp;sntz=1&amp;usg=AFQjCNGbpEdQpGD-XIs2gbwAUWq7CFPnRQ) almacenar o recuperar datos de él.
3. **Capa de datos o persistencia:** es donde residen los datos y es la encargada de acceder a los mismos. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa del modelo.

Una de las cosas más importantes al utilizar este modelo de capas es no olvidar que nunca puedes llamar desde una capa a otra que no se comunique directamente con ella. Es imprescindible seguir la visibilidad de capas.

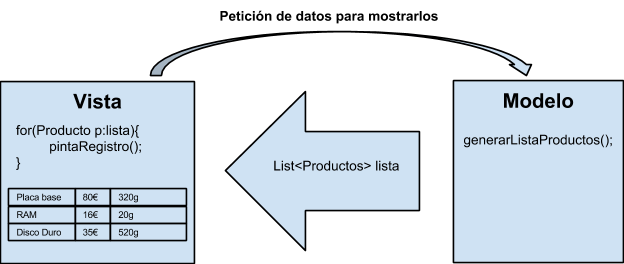


### **Vista**

La capa de vista es la capa encargada de mostrar los datos que le proporciona el

modelo al usuario. Esta capa se comunica con el usuario y con el modelo, formatea los datos para mostrárselos al usuario y utiliza lógica para mostrarlos, pero nunca lógica de aplicación.

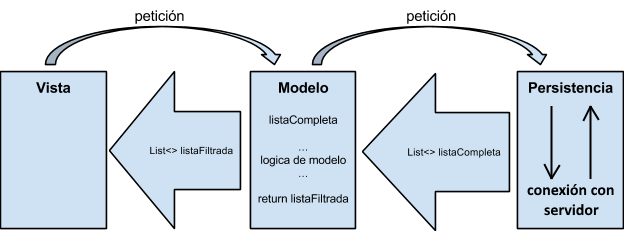
Un ejemplo de esto podría ser que la capa de modelo le envíe un array con productos y la vista lo formatee para rellenar una tabla en la que aparezcan los productos con toda su información.



### **Modelo**

La capa de modelo es la capa encargada de tratar los datos que recibe de la vista y de la persistencia. El modelo es el cerebro de la aplicación, y nunca debe comunicarse directamente con el usuario, así que evitaremos por ejemplo mostrar una pantalla de error al usuario desde esta capa. En el caso de existir el error, la forma de actuar será mandando la información del error a la vista y la vista mostrará el error.

En nuestro caso, un ejemplo muy claro es que pedimos una lista con enlaces a la base datos, la base de datos nos devuelve una lista con todos los enlaces, nosotros seleccionamos en el modelo que necesitamos y se los pasamos a la vista para que los muestre.



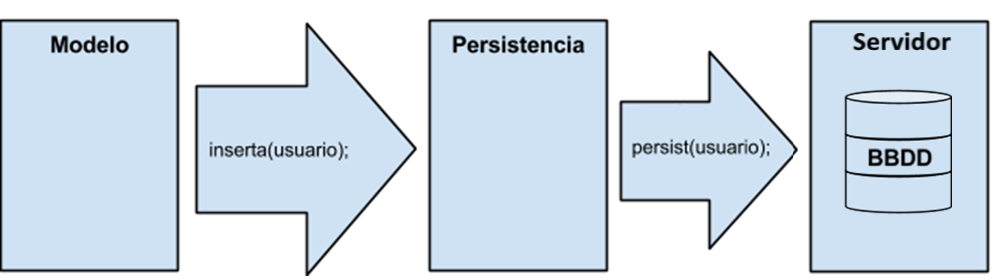
### **Persistencia**

La capa de persistencia es la encargada de comunicarse con la base de datos para realizar las operaciones de actualización, inserción, borrado y selección.

La persistencia podríamos decir que es la que hace el trabajo sin pensar, simplemente recibe las órdenes del modelo y la lleva a cabo, sin ningún tipo de lógica.

Para la capa de persistencia hemos usado un modelo relacional.

Un ejemplo de esto lo tenemos en la inserción de usuarios dentro de la base de datos. El modelo le proporciona unos datos a la persistencia y le pide al servidor que lo inserte mediante una petición, y el servidor sin ningún tipo de lógica de tratamiento de datos, simplemente coge el objeto y lo inserta en la base de datos.



## Tecnologías utilizadas

En este pequeño apartado se enumeran las diferentes herramientas y tecnologías utilizadas durante el desarrollo del proyecto.

**Microsoft Visual Studio**

Es un IDE (entorno integrado de desarrollo) que permite desarrollar aplicaciones, sitios web, aplicaciones WPF, aplicaciones web, servicios web, apps de Windows Store, etc. para poder ser ejecutadas en todas las plataformas de Microsoft Windows, Windows Mobile, Windows CE, .Net Framework, .NET Compact Framework y Microsoft Silverlight. Permite emplear múltiples lenguajes de programación.

**Newtonsoft Json.Net**

Se trata de una librería para la serialización de objetos JSON. La utilización de esta librería nos va a aportar las siguientes ventajas:

* Una forma de trabajar sencilla y unificada para todas nuestras clases.
* JSON.NET nos permite serializar de .NET a JSON, pero también de forma inversa de JSON a .NET.

**LiveCharts**

Se trata de una aplicación diseñada para facilitar la visualización de marcadores o estadísticas en programas basados en la plataforma .NET.

Con esta librería todo se actualiza y anima automáticamente, la biblioteca sólo se actualizará cuando lo considere necesario, no cada vez que los datos cambien, cuando agreguen o elimine datos, los valores de las gráficas se actualizarán por sí solas.

**ObjectListView**

En este caso estamos hablando de un wrapper de C# alrededor del ListView de .NET. Hace que el ListView sea mucho más potente, fácil de usar y con algunas características propias que no se encuentran en la librería por defecto.

**GitHub**

Es unaplataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Se utiliza principalmente para la creación de código fuente de programas de ordenador.

**XAMPP**

Se trata de un servidor independiente de plataforma, software libre, que consiste principalmente en la base de datos MySQL, el servidor Web Apache y los intérpretes para lenguajes de script: PHP y Perl.

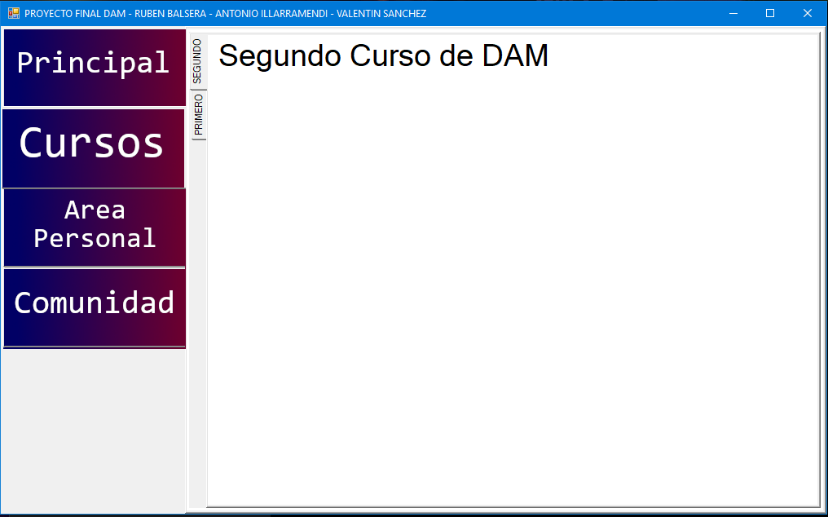
# Diseño de la interfaz

El diseño de la aplicación se basa en una interfaz principal con un menú lateral desplegable que estará en todo momento presente. En él se podrá usar los botones que navegan a los distintos formularios de la aplicación, abriéndose éstos dentro de la propia interfaz principal, como se menciona más arriba.

Este diseño de la aplicación fue evolucionando en distintas fases de prototipado que a continuación se detallan:

## Primer prototipo de pantalla

En este primer diseño puede observarse cómo la aplicación está en su fase más temprana. En este momento la interfaz luce un diseño simple y poco refinado, esto es debido a que en ese momento el equipo de desarrollo estábamos más centrados en recopilar información y probar distintas soluciones para aplicar la interfaz. En un primer momento probamos con distintas librerías externas, pero no nos convencía el resultado final.



## Segundo prototipo de pantalla

Para el segundo diseño se intentó aplicar el menú de la pantalla principal a base de pestañas. Cambiando bastantes propiedades se conseguía un diseño más acorde a lo que teníamos en mente, pero las pestañas que trae por defecto la plataforma .Net son poco potentes y pronto nos encontramos con problemas de dibujado de éstas.

Decidimos entonces usar una librería externa para las pestañas que permitían más personalización y tenían posibilidades extendidas. El problema salió a la luz cuando nos dimos cuenta de que con esta librería el rendimiento caía de forma espectacular, haciendo al programa poco manejable de cara al usuario.



## Prototipo final de pantalla

Finalmente decidimos hacer todo el diseño a mano, es decir usando paneles a los que les fuimos dando forma y color y posicionándolos adecuadamente en su sitio. De este modo conseguimos una interfaz muy versátil, a la que podíamos añadirle todo tipo de animaciones y efectos y que también podíamos personalizar en base a las necesidades que fueron surgiendo a lo largo del desarrollo.



## Diseño de las pantallas

La interfaz se ha diseñado con el propósito de que sea intuitiva y simple, pero a la vez potente y capaz de cubrir todos los requisitos enumerados en el apartado [de](#_bookmark19) catálogo [de requisitos](#_bookmark19).

Como ya se menciona en el apartado prototipos toda la interfaz está diseñada y programada desde cero, cubriéndose toda la aplicación de paneles personalizados que se adaptan perfectamente a todo tipo de resoluciones y configuraciones de pantalla.

### **Pantalla principal**

****

La pantalla principal es el núcleo de la interfaz de la aplicación cliente, está programada totalmente desde cero y compuesta de paneles, pues ofrece una mayor verdsatilidad frente a los Windows Forms clásicos.

El panel de título está programado para funcionar como una barra de título de cualquier aplicación de Windows, arrastrando el ratón sobre ella podemos mover la aplicación y también dispone de los clásicos botones de control, cómo un boton mas para poner la aplicación en pantalla dividida.

En el lado izquierdo se ha colocado un panel de navegación, en el se ha posicionado en orden descendente:

**El estado de conexión con el servidor:** Aquí se ha posicionado un label informativo, donde se muestra la conexión con el servidor, el boton de recarga volvera a comprobar si la conexión sigue activa.

**Botón ocultar menu:** Al presionar sobre dicho botón se mostrará una animación que colocará el menu en modo reducido, para priorizar el panel principal.

**Botones de navegación:** Éstos son los que usará el usuario para moverse por los distintos formularios.

**Caja de login:** Aquí el usuario podrá conectarse con su id uan vez se haya registrado, se muestra el usuario actual y las cajas de introducción de datos. Por último tenemos el boton de conexión que valida las credenciales y las envia al servidor para que se corroboren con la base de datos.

Finalmente el resto del formulario se compone del panel principal, donde sevan mostrando los distintos formularios de los que se compone la aplicación, dibujandose en el panel y destruyendose cuando se cambia de formulario. En la parte inferior derecha hay un pequeño boton al que se le a asignado la funcionalidad de redimensionar el panel de la aplicación.

Tambien se han agregado varias animaciones, además de la ya mencionada del menu lateral, como la del inicio y final de la aplicación o en los cuadros de dialogo.

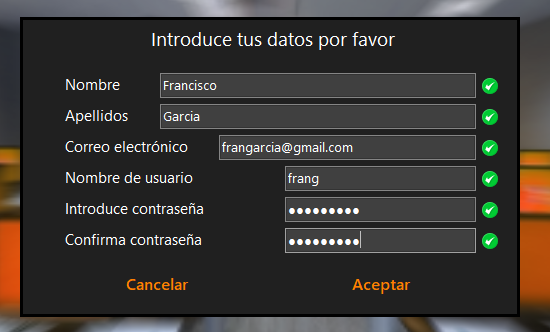
Como toque final se añadió una sutil sombra a todo el marco exterior para una mejor apariencia visual e integración con el escritorio.

### **Pantalla Registro**

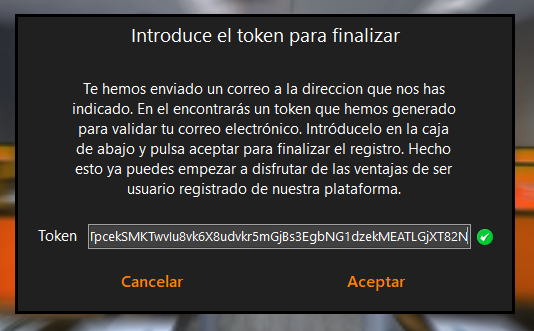
Se trata del formulario que aparece por defecto en el panel principal de la pantalla principal. Nos informa brevemente de el propósito de la aplicación y permite inciar el registro de un nuevo usuario. Si pulsamos sobre el boton “Regístrate pulsando aquí” aparecera el siguiente cuadro de dialogo:



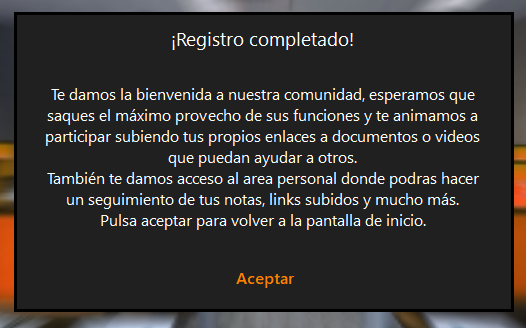
Aquí se nos pediran una serie de datos necesarios para el registro. Las cajas de texto estan validadas de tal modo que no llegen datos incorrectos al servidor.



Tras rellenar los datos que se requieren y éstos sean validados por la interfaz se activará el botón “Aceptar”, posibilitando enviar los datos en una petición al servidor.



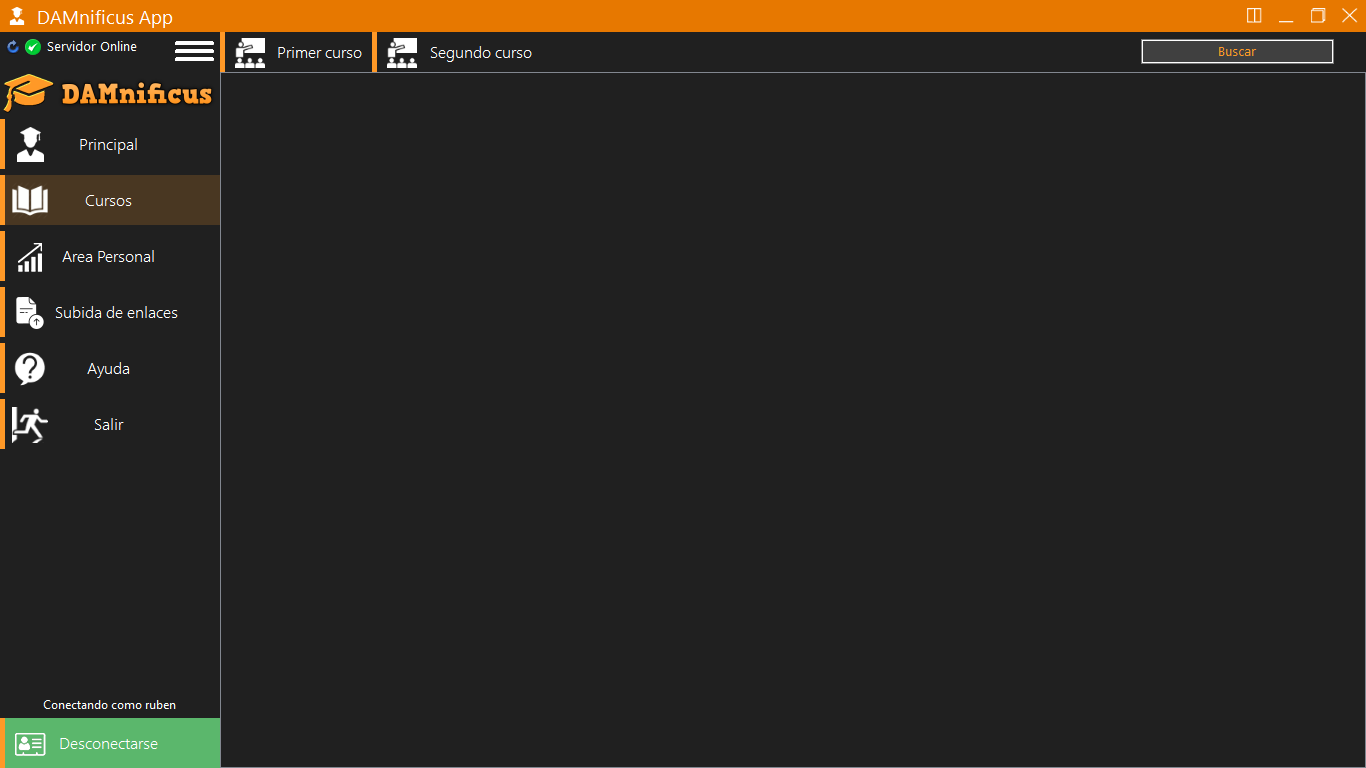
Tras esto se nos enviará un email al correo que hemos escrito donde se nos facilita un token de autentificación, este token se ha de copiar en la caja de texto que aparece en el cuadro de dialogo. Si el token es correcto se mostrará un tick verde y se habilitará de nuevo el botón “Aceptar”.



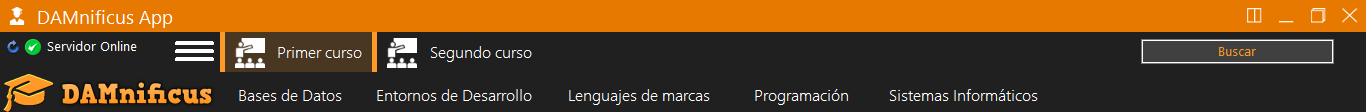
Si todo es correcto de cara al servidor se nos devolverá un mensaje de bienvenida y se nos dará acceso a la aplicación con el usuario que acabamos de registrar volviendo de nuevo a la pantalla principal.

### **Pantalla Área cursos**

En este formulario el usuario podrá visualizar, valorar y reportar fallos de los enlaces a documentos, videos o webs de los diferentes cursos y asignaturas que los componen, de una manera muy visual y sencilla.



Estando en esta pantalla tenemos que seleccionar el curso y se desplegará un menu inferior con las asignaturas que lo componen.





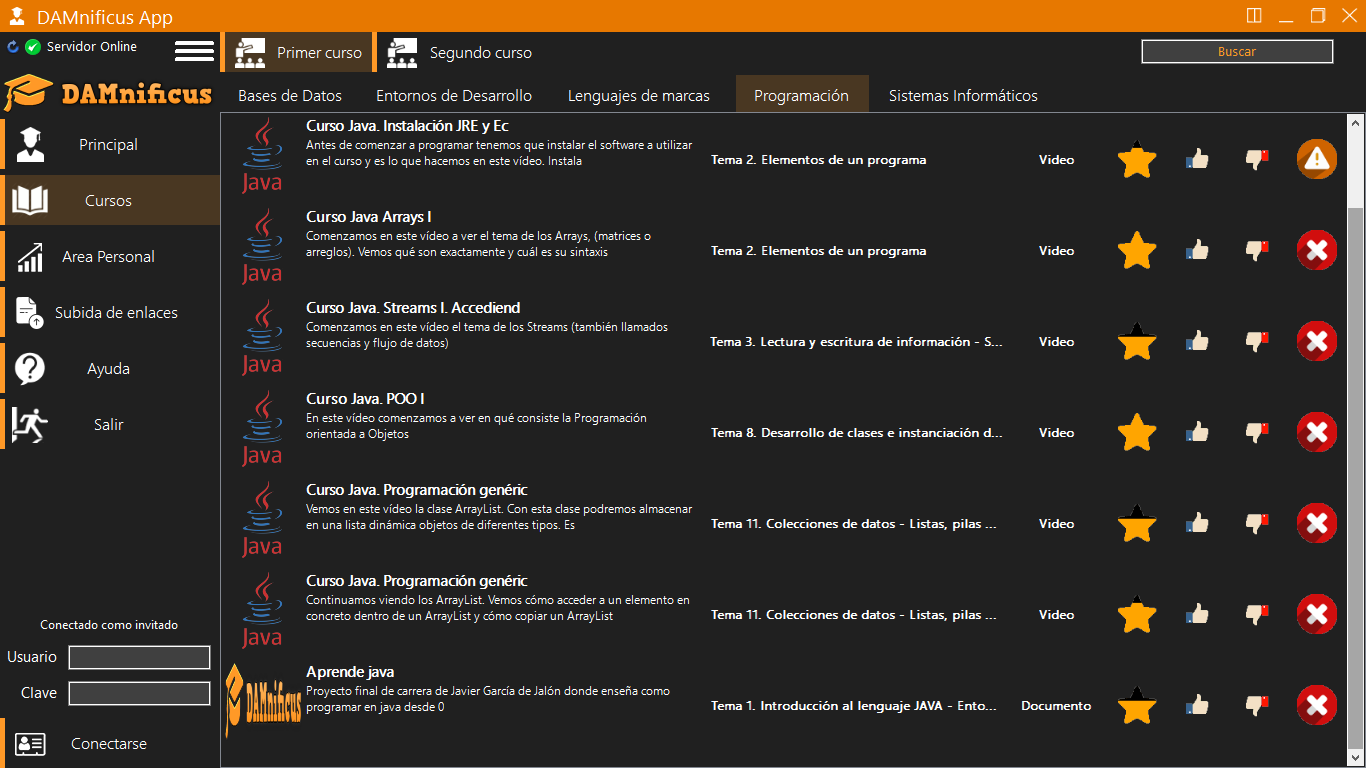
De cada enlace se muestra en diferentes columnas la siguiente información:

* Imagen.
* Titulo y descripción.
* Tema de la asignatura.
* Tipo de enlace.
* Valoración de los usuarios.
* Imagen para votar positivamente.
* Imagen para votar negativamente.
* Imagen para reportar un link caido.

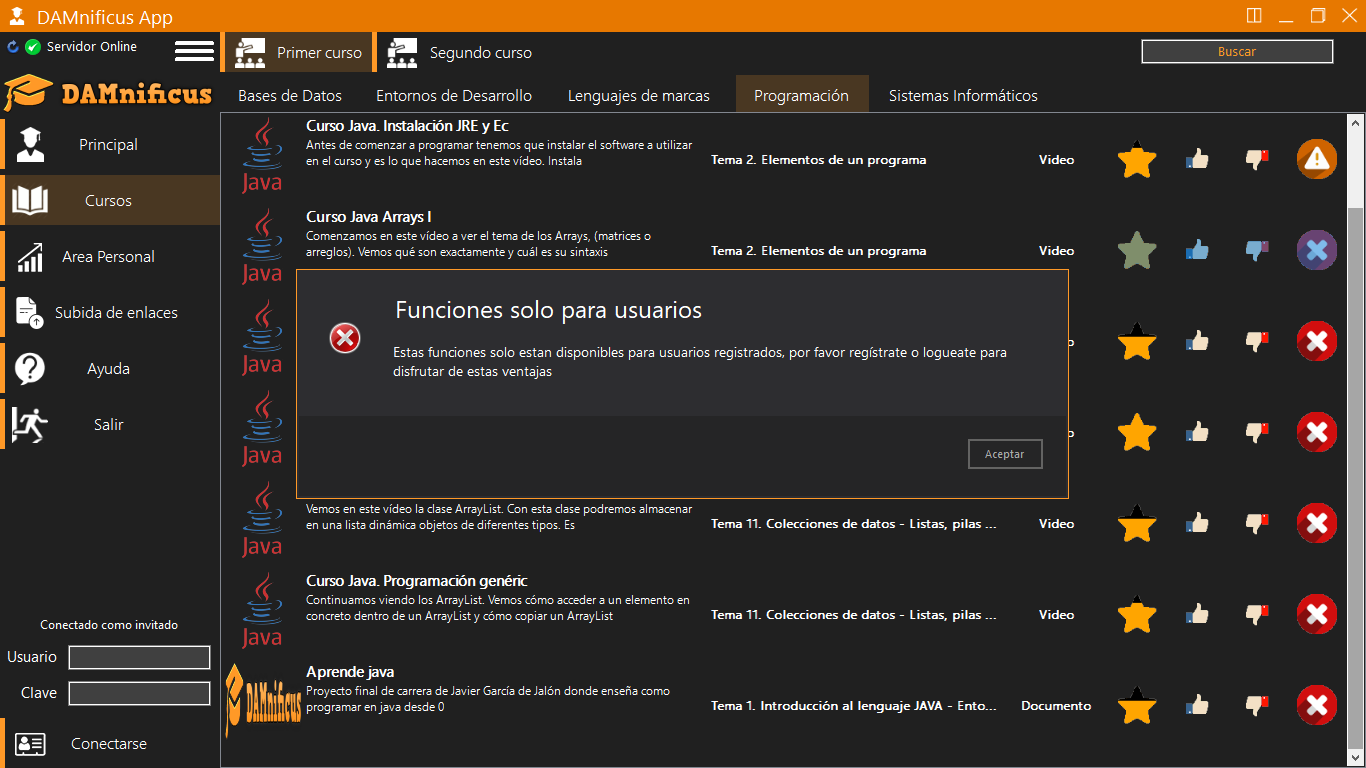
Se ha hecho una diferenciación entre usuario invitado, usuario normal y usuario admin o con privilegios de admin. En función del tipo de usuario muestra diferentes tooltip e imágenes en la columna reportar\_fallo además de tener funciones deshabilitadas.

Una vez seleccionado el curso y la asignatura se muestra la siguiente pantalla

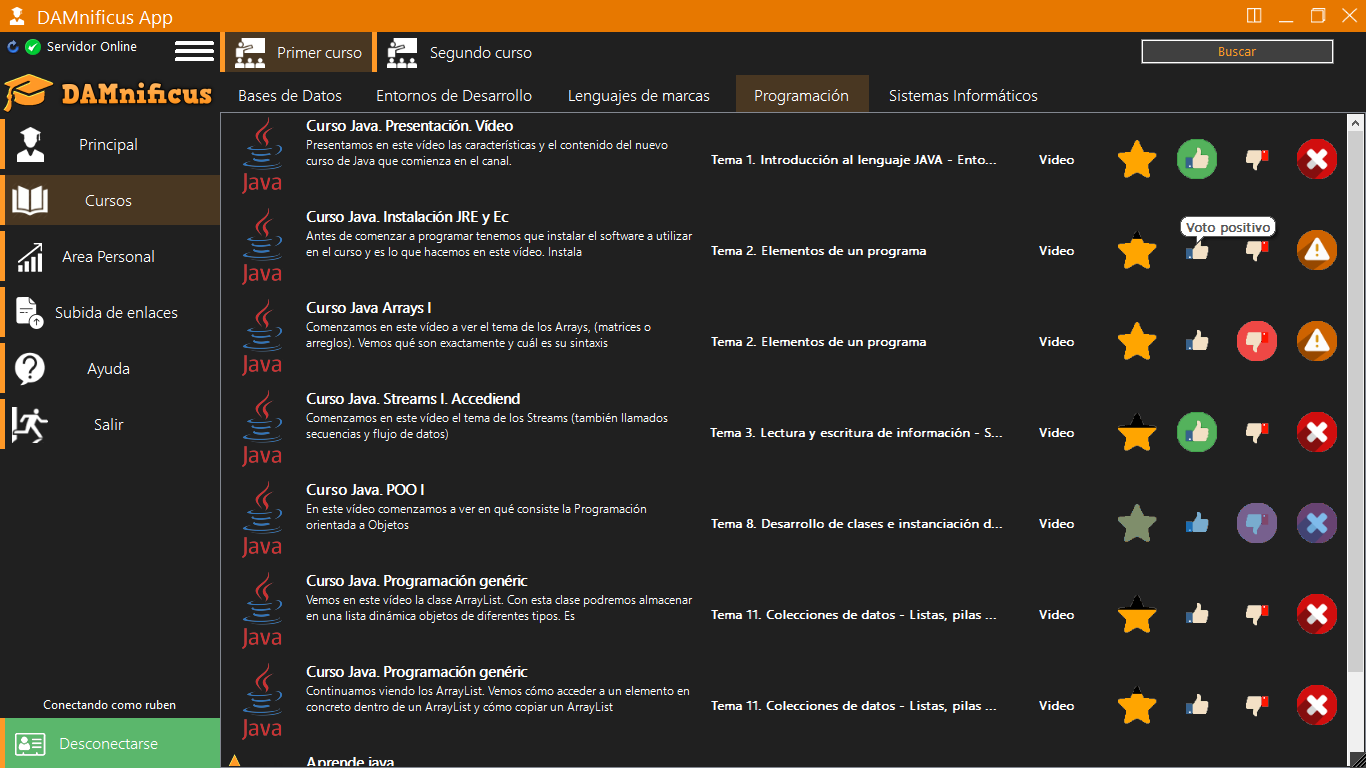
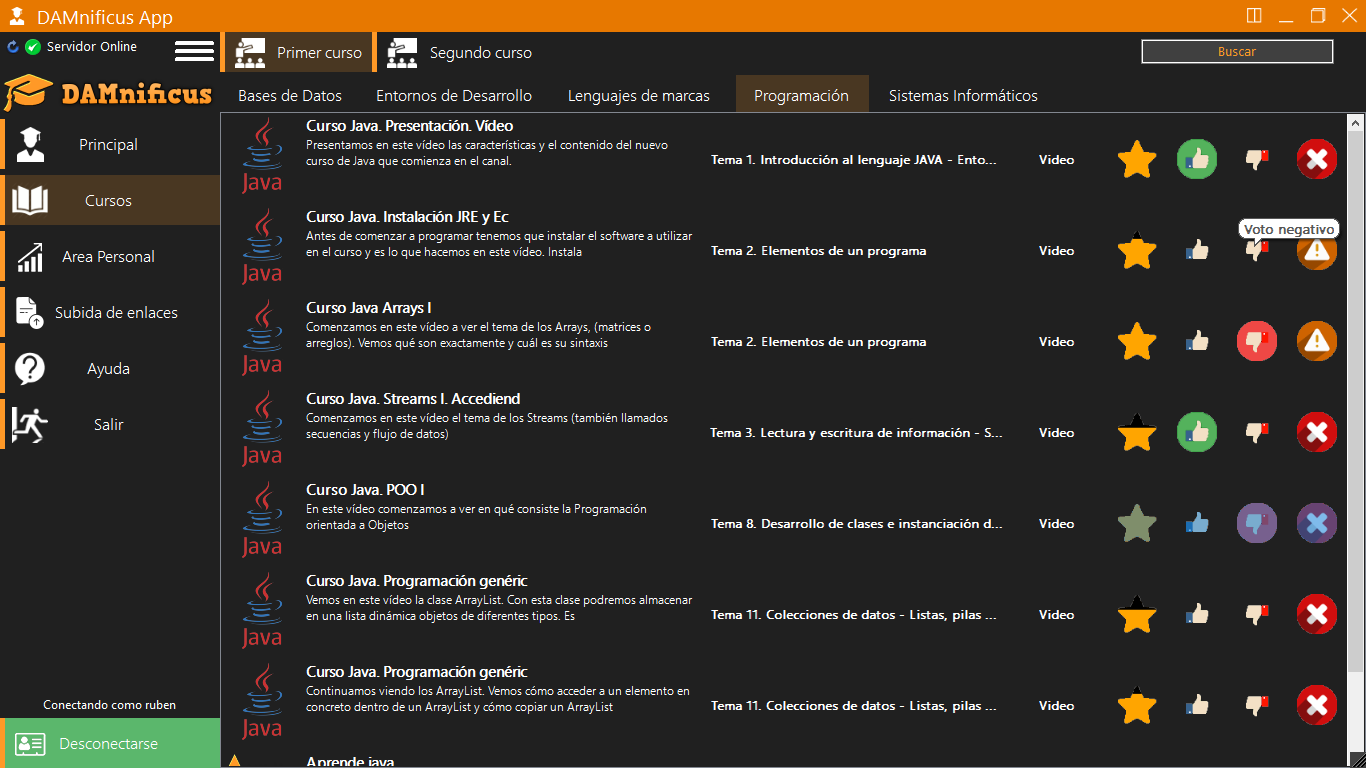
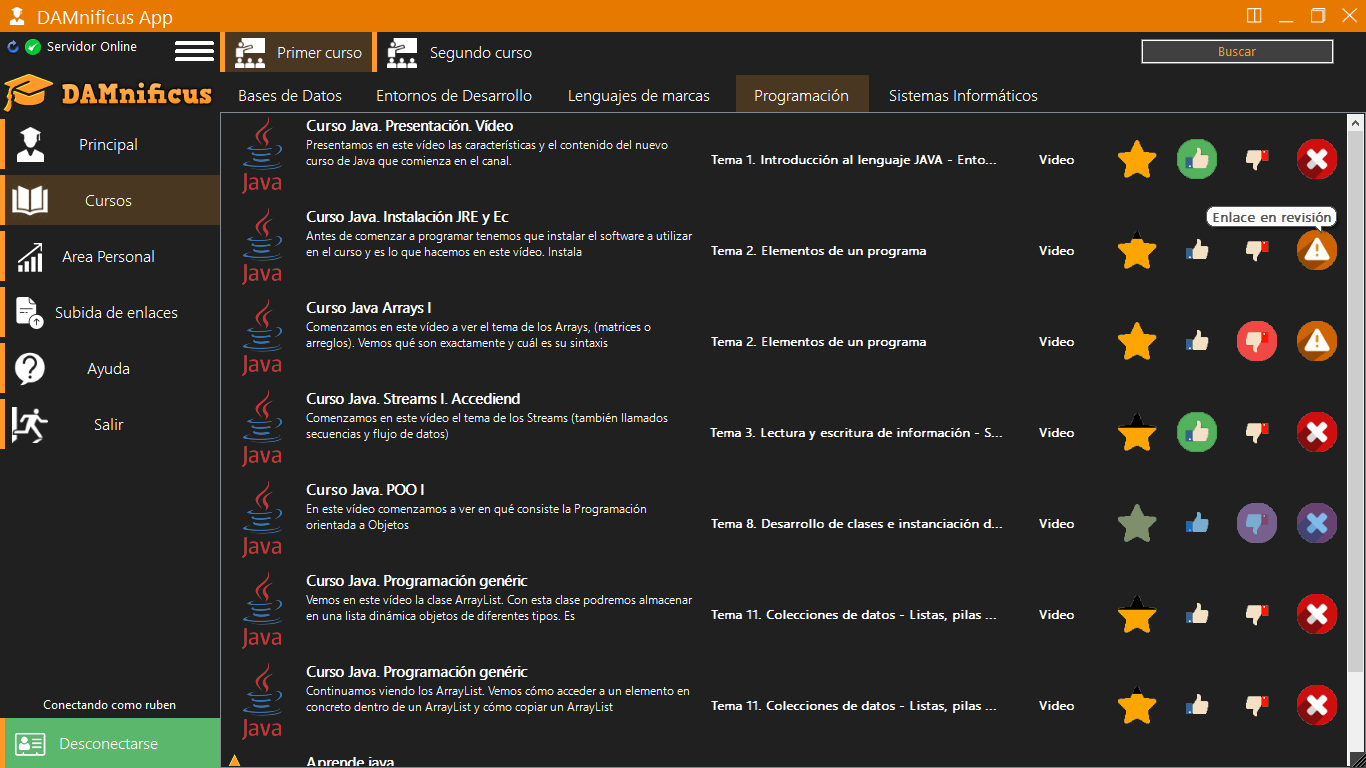
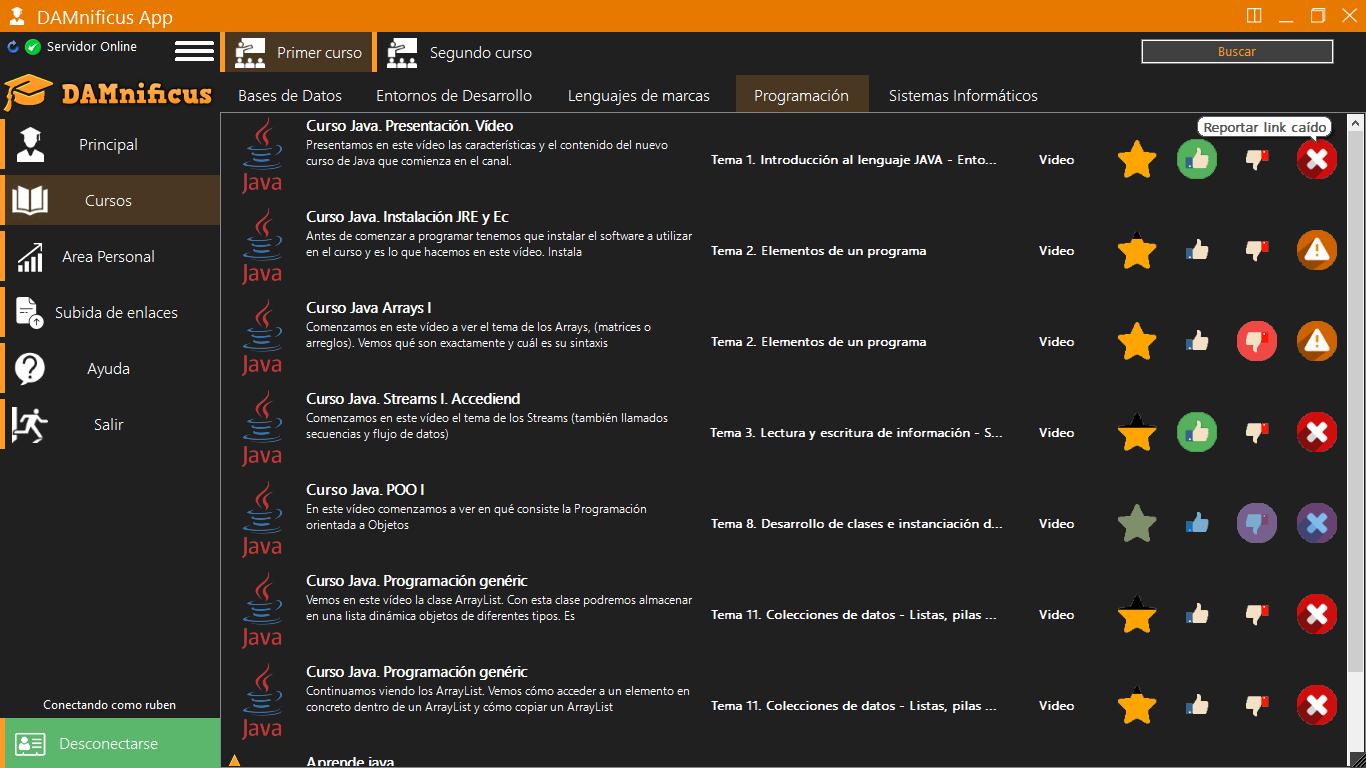
Usuario Invitado:



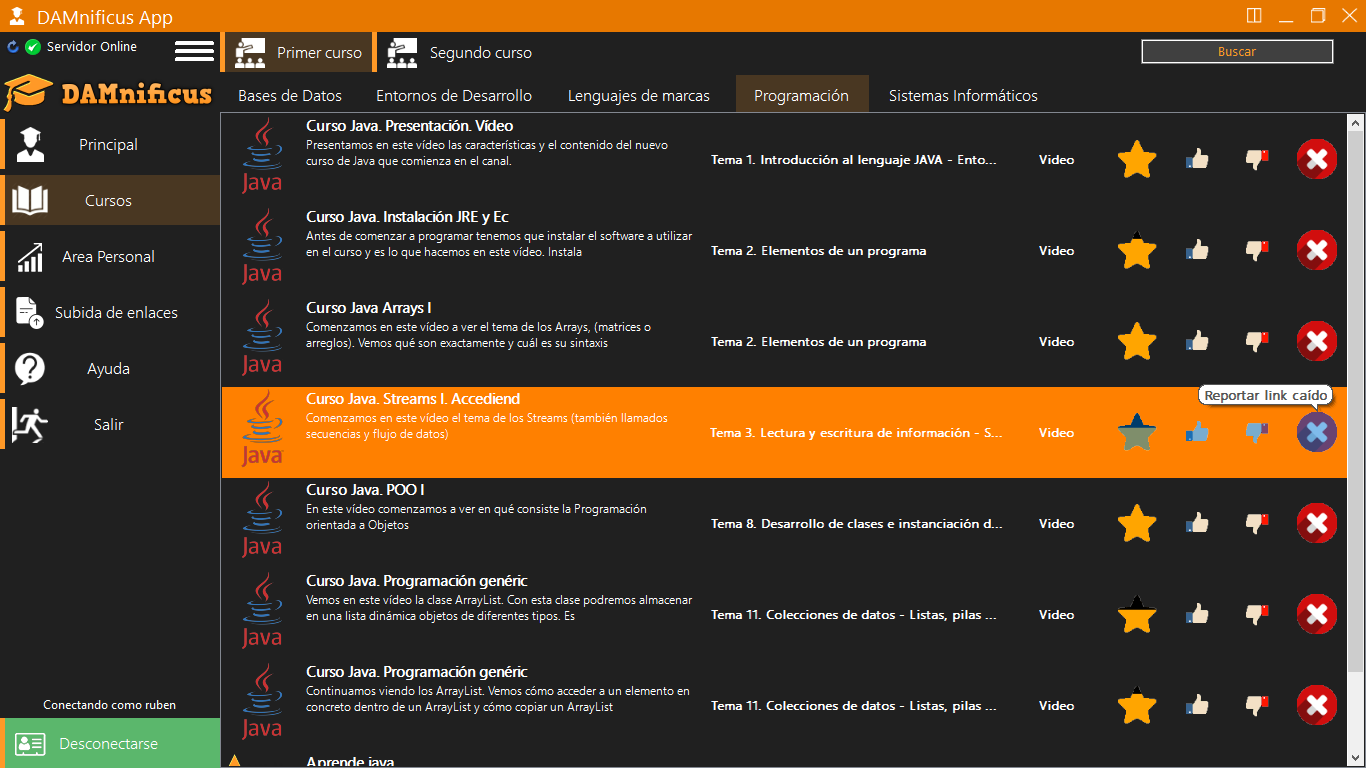
Puede visualizar todos los documentos, videos o webs que se encuentran en estado “Activo” o “Revision”, pero no los que se encuentran en estado caido. Haciendo doble click sobre la fila se abrirá el navegador llevandole al enlace seleccionado, pero no podrá valorarlos ni reportarlos como caidos, si intenta usar alguna de estas funciones se muestra un mensaje informativo indicando que esas funciones solo están disponibles para usuarios registrados.



Diferentes mensajes tooltip que puede visualizar este usuario:



Usuario Normal:

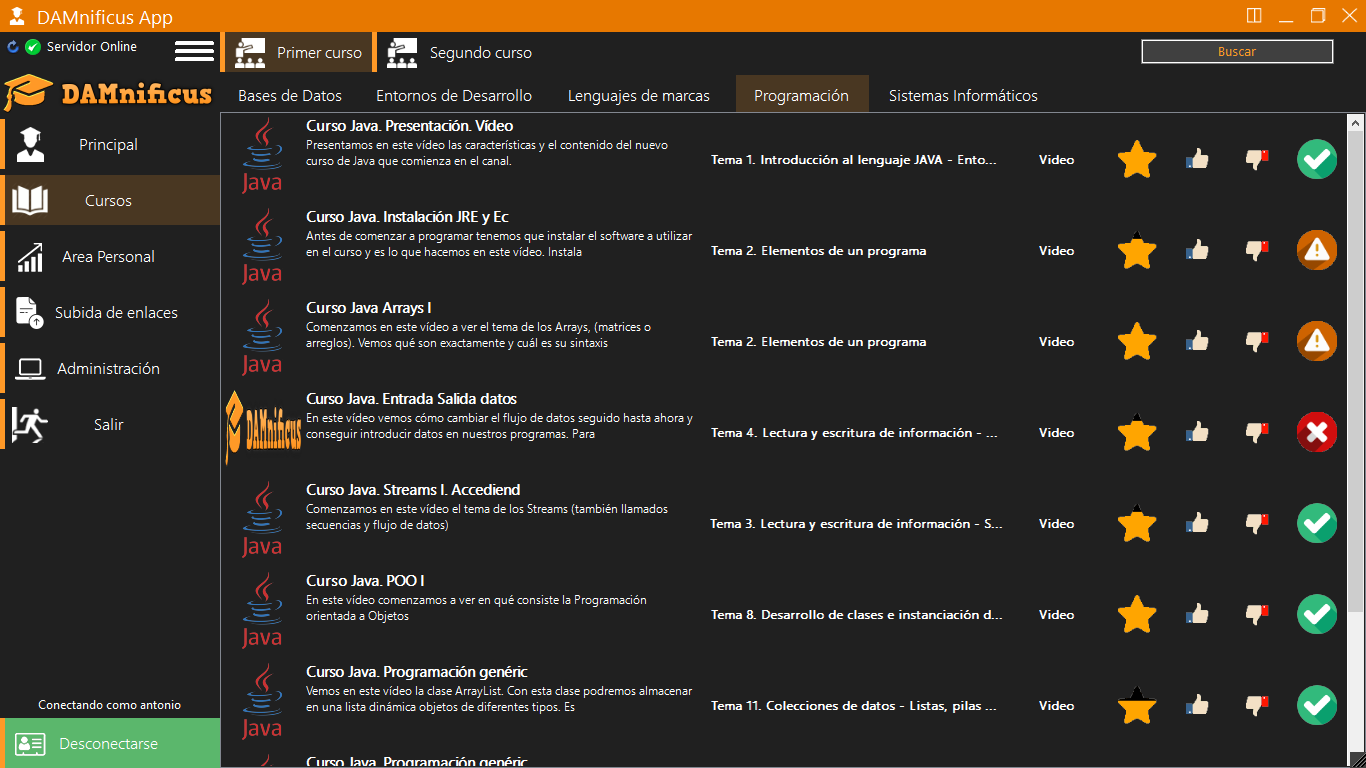


Visualiza el mismo contenido y mensajes tooltip que un usuario Invitado, con la diferencia de que este si podrá valorar el link haciendo click sobre la imagen dando un voto positivo o negativo. Al hacer esto, envía al servidor la petición de sumar o restar valoración. Si la acción se realiza correctamente la imagen cambia para hacer constar al usuario de que su voto ha sido registrado correctamente.

También puede reportar un link caído. De la misma manera, envía la petición al servidor que cambia el estado del link “Activo” a “Revisión”. Si todo es correcto se cambia la imagen y el tooltip para informar que el enlace se encuentra en estado de revisión y ya no podrá ser reportado como link caído por más usuarios.

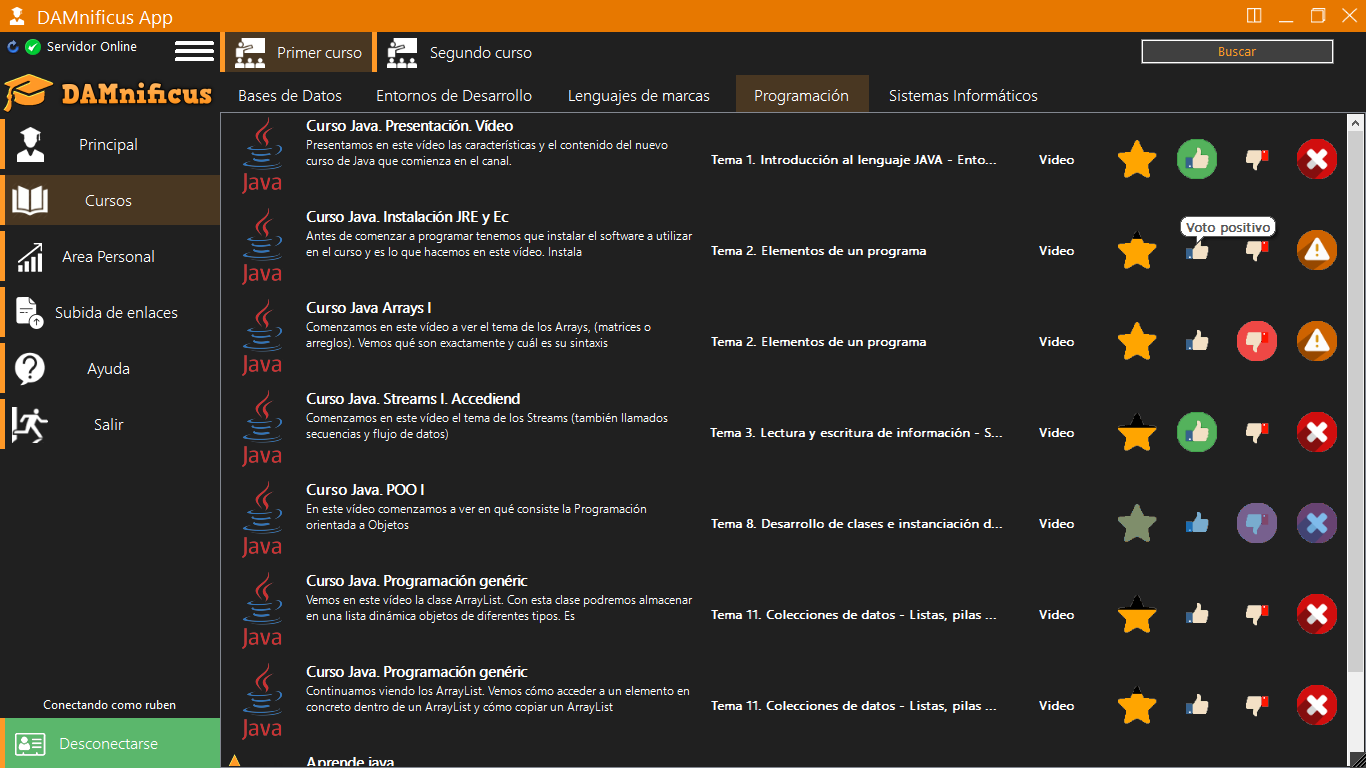
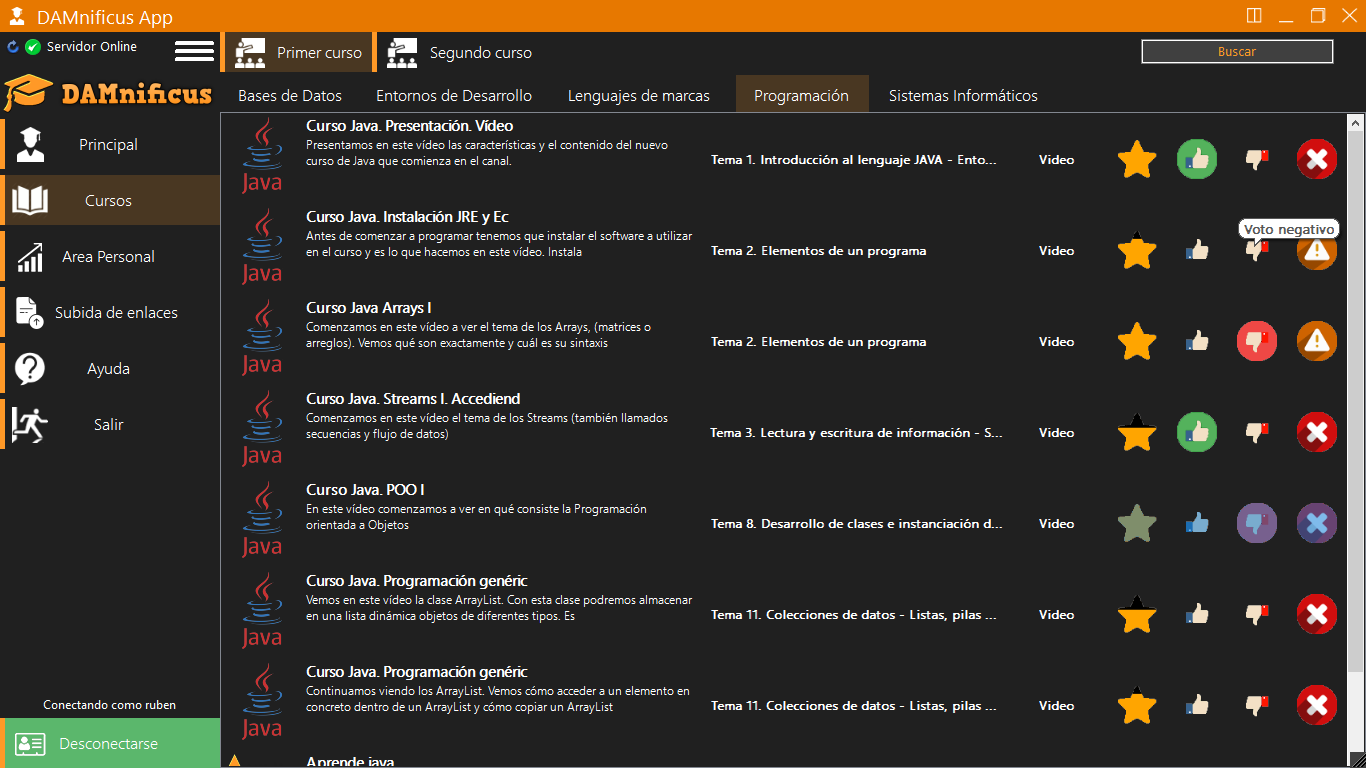
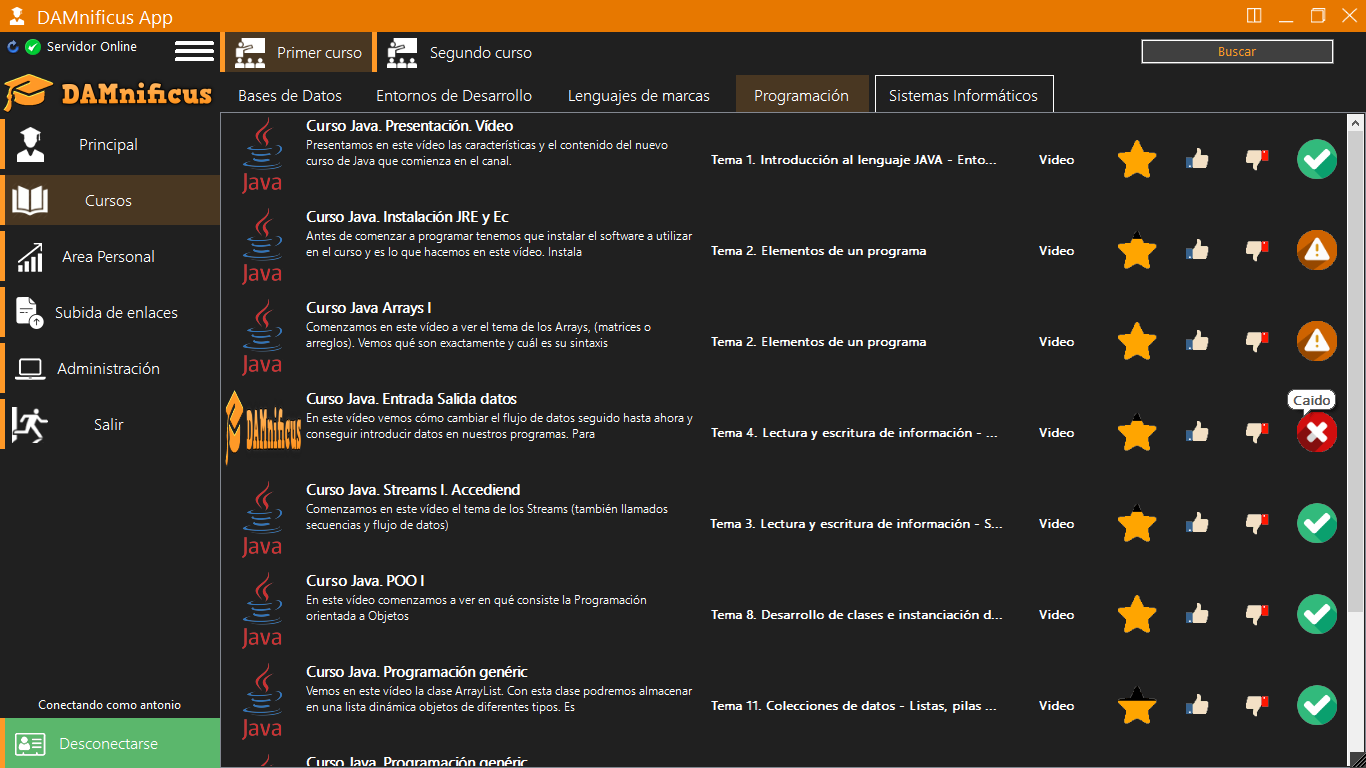
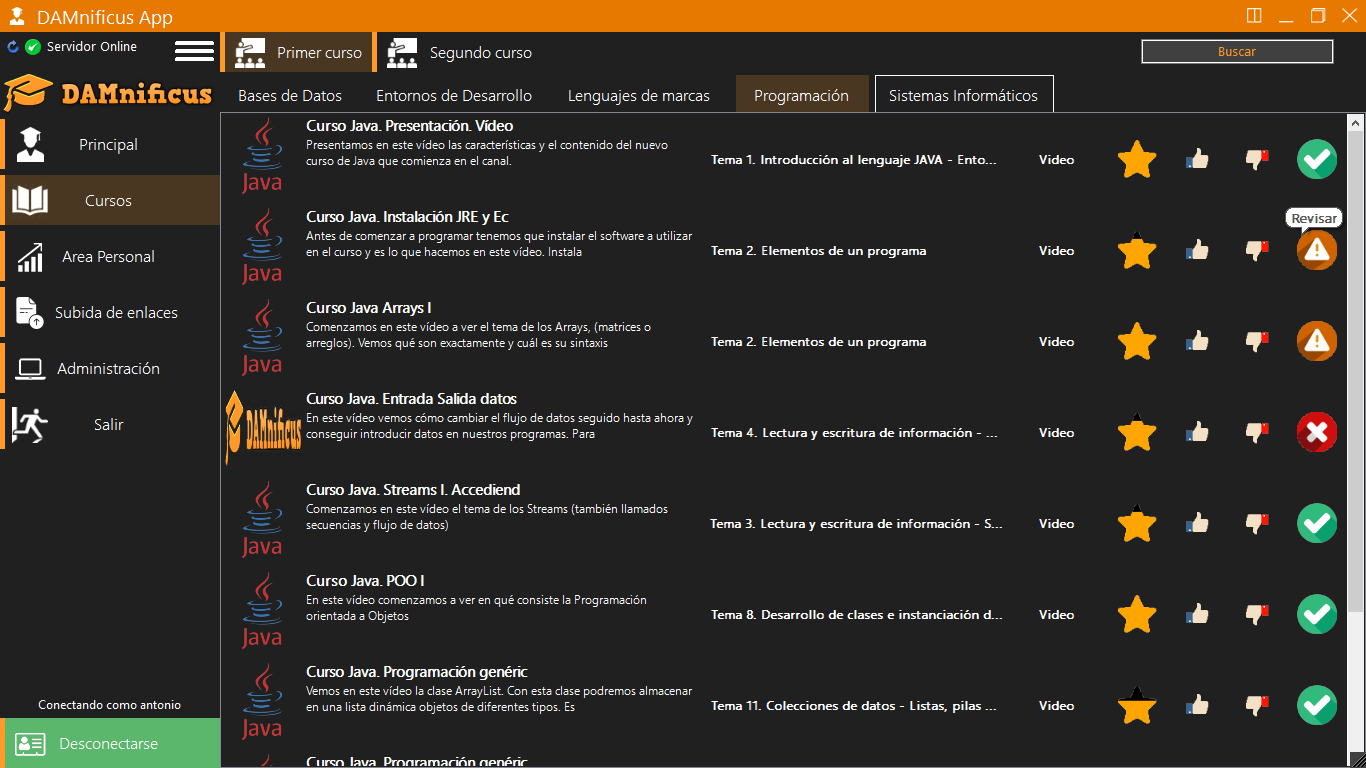
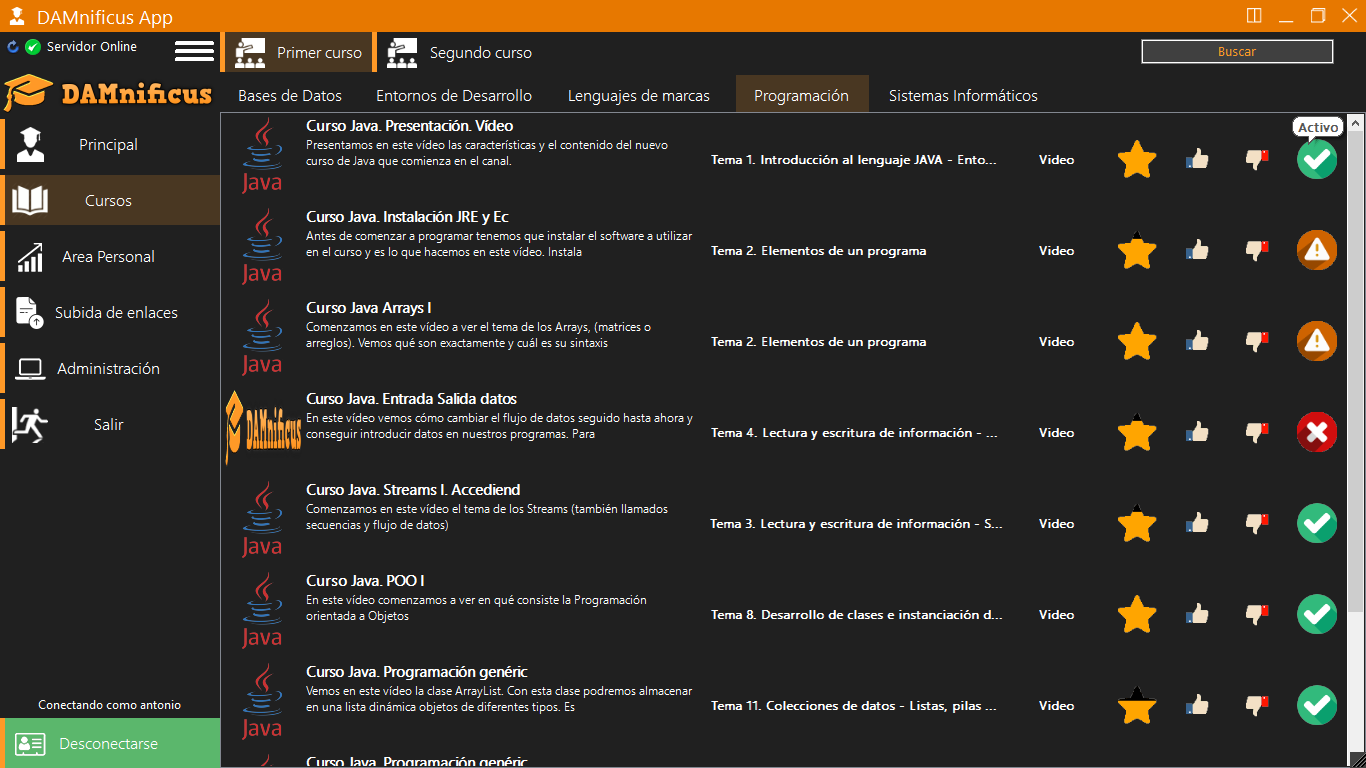


Usuario Admin o con privilegios de admin:

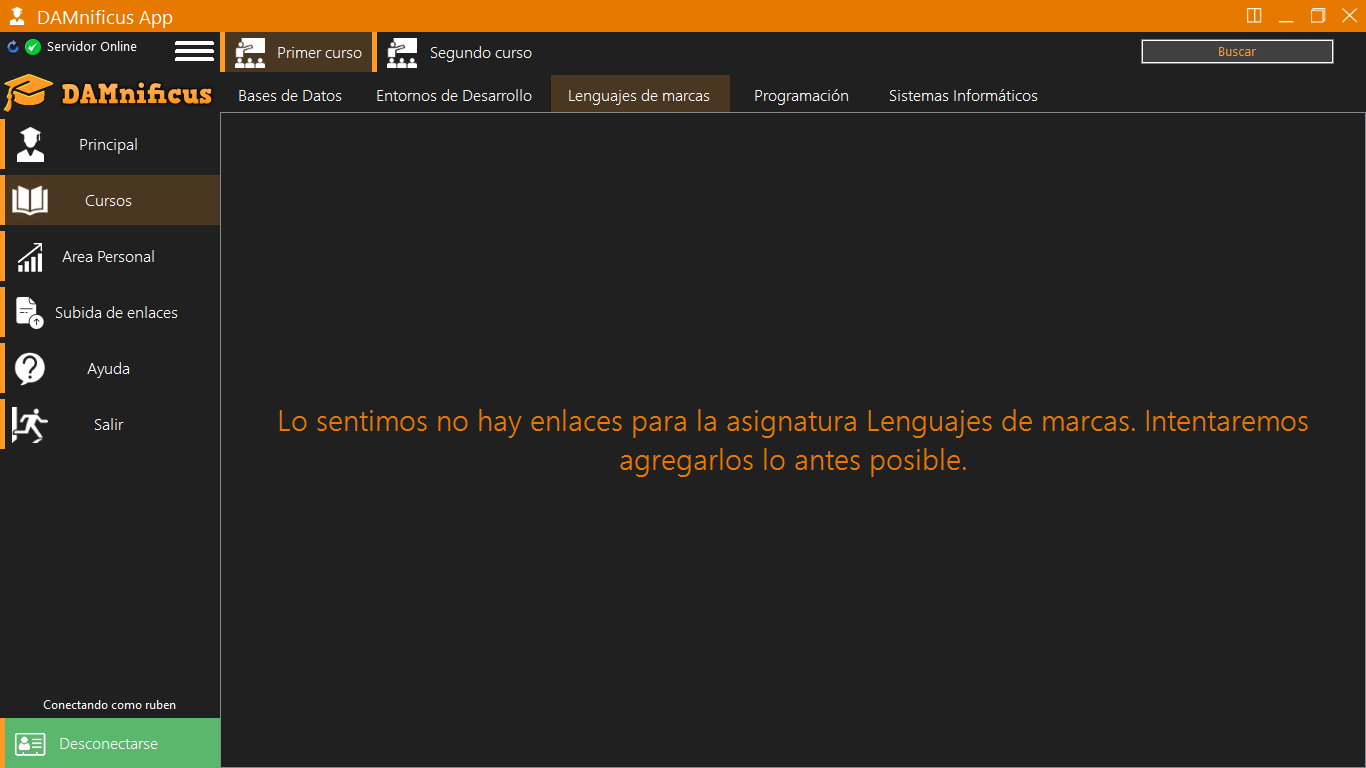


Este usuario puede visualizar todos los enlaces intependientemente del estado en el que se encuentren. En la columna reportar\_fallo tiene una tercera imagen, de la cual no disponen los usuarios Invitado y Nomal. Esto permite visualizar los diferentes estados en que se encuentran los enlaces (“Activo”, “Revisar” y “Caido”).

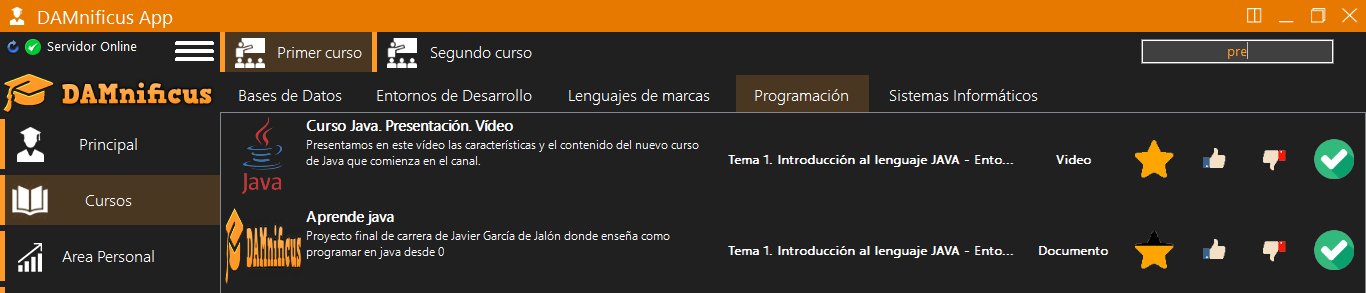
Diferentes mensajes tooltip que puede visualizar este usuario



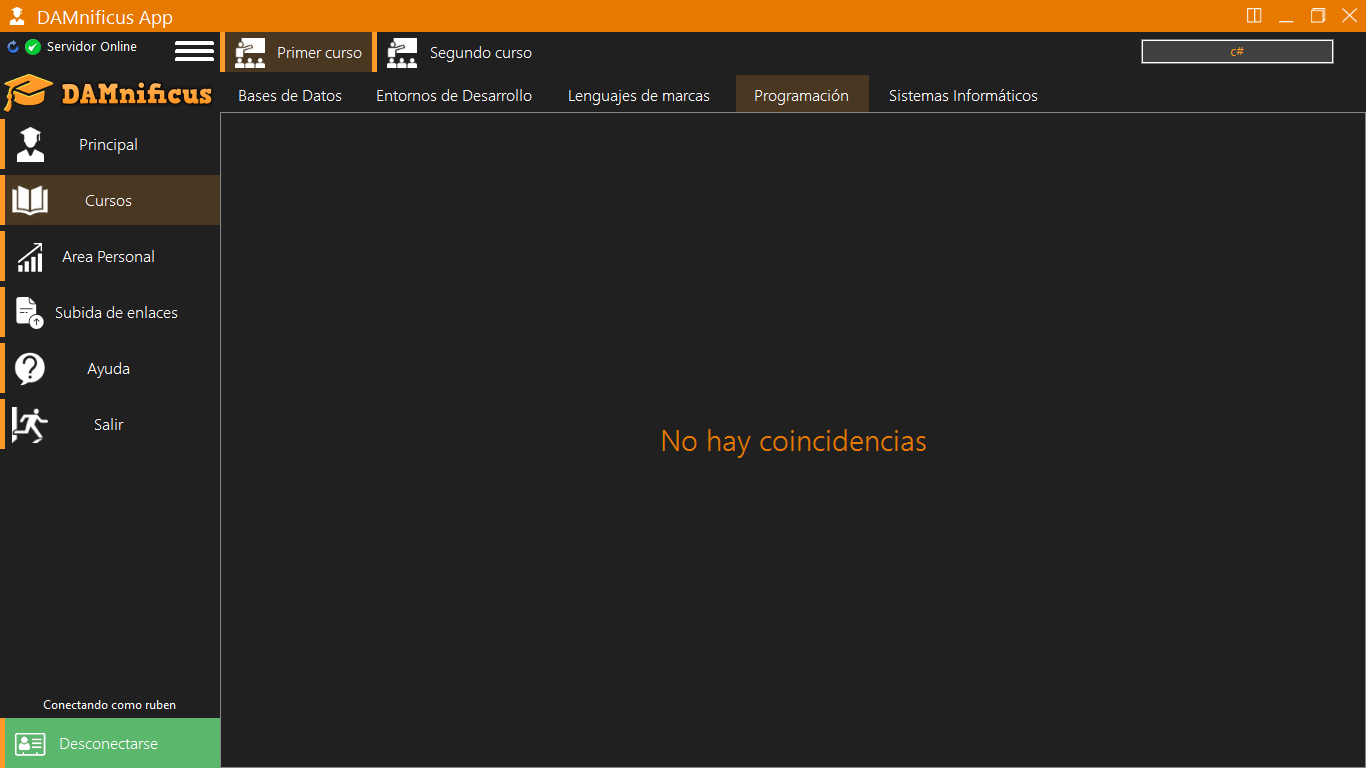
Para todos los tipos de usuarios si la asignatura seleccionada no tiene añadido ningun enlace se informará con un mensaje en la pantalla.



Otra característica del formulario Cursos es el filtro en tiempo real. Este filtro se aplica al título de cada enlace he irá eliminando de la lista los enlaces que no coincidan con la cadena introducida en el textbox “Buscar”.

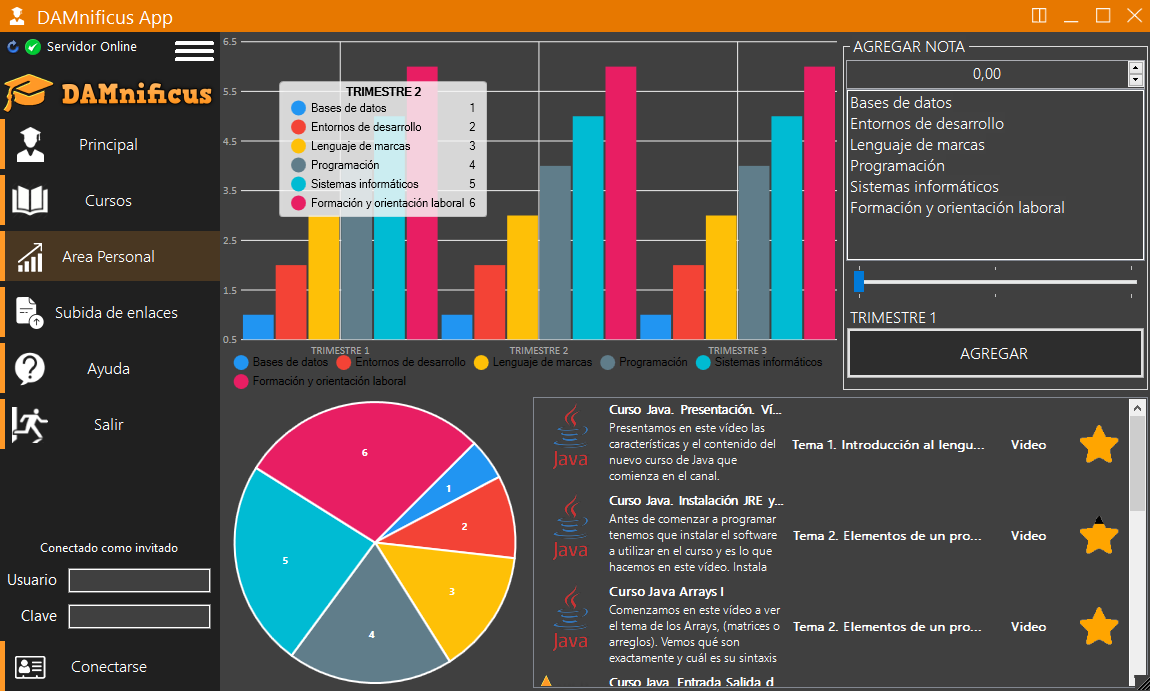


Si no existe ninguna coincidencia muestra un mensaje en la pantalla indicándolo.



Este formulario usa un ObjectListView, un dll creado por [**Phillip Piper**](https://sourceforge.net/u/grammarian/) que mejora el elemento ListView agregando muchas funcionalidades de una manera sencilla.

### **Pantalla Área Personal**

**

Primera pantalla según clicamos

El formulario ÁreaPersonal está construido a partir de un tableLayoutPanel de 3 columnas y 2 filas. En la primera fila columnas 1 y 2, y en la segunda fila columnas 2 y 3 están fusionadas mediante ColumnSpan para acoplar los componentes y obtener una mejor visualización.

El diseño es responsive - como el resto de formularios - con lo que modificando el tamaño del area, se acoplarán los componentes, esto debido a que están montados sobre tableLayoutPanel. Posicionando el raton sobre las barras de representación de las notas, como de las porciones de la gráfica de notas medias, obtendremos una lista que recogerá todas las notas, o bien referente al trimestre sobre el que estamos colocados si estamos en la gráfica de notas, o bien una lista de las asignaturas y sus notas medias si estamos situados sobre la pieChart.

Hemos usado una extensión NuGet llamada LiveChart para WindowsForms para la representación de las notas del usuario.

En la primera fila, primera y segunda columna tenemos una CartesianChart en forma de ColumnSeries, dependiendo del curso en el que esté matriculado el usuario, constituida por 6 o por 7 Series. Estas series se cargarán con el nombre de las asignaturas y sus valores de los 3 o 2 trimestres. Los valores de estas series se obtendrán mediante la consulta a la base de datos como se describirá más adelante.

En la primera fila, tercera columna está colocado un GroupBox que se encarga de la inclusión de notas en la base de datos. Su composición interna es de un ListBox que carga mediante el UsuarioConectado, las asignaturas del curso actual en el que está matriculado actualmente y que cuya carga se explicará más adelante. Un TrackBar representante de los trimestres, que también dependerá del curso matriculado, ya que como todos sabemos, el primer curso está compuesto de 3 trimestres, y segundo curso tiene solo 2, con lo cual, en la carga de componentes, definirá un máximo dependiendo del curso recuperado. Para el valor de la nota colocamos un NumericUpDown con valor mínimo 0 y valor máximo 10, con dos posiciones decimales, acorde con el tipo de valor almacenado de las notas en la base de datos, que en cada pulsación subirá o bajará el valor 0,01. Por último un Button que introducirá en la base de datos la nota seleccionada en el NumericUpDown, de la asignatura seleccionada en el ListBox, y correspondiente al trimestre seleccionado en el TrackBar.

En la segunda fila, primera columna colocamos una PieChart, que dependerá también del curso del usuario, que al igual que con la gráfica de notas, cargará 6 o 7 PieSeries que contendrán las notas medias del usuario que se obtendrán a través de la base de datos y que se explicará mas adelante en el apartado de implementación.

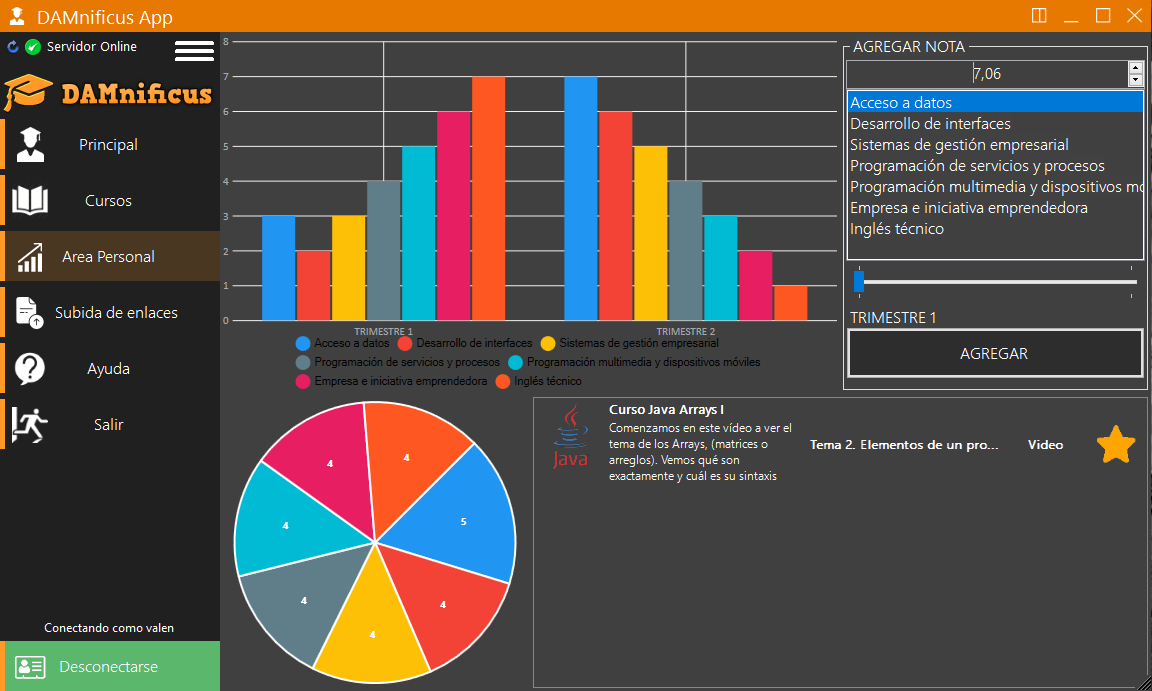
En la segunda fila, segunda y tercera columna hemos colocado, al igual que en el formulario Cursos, un ObjectListView que recoge de forma agrupada un listado de los enlaces subidos por el usuario que está logueado en ese momento.

Para la carga de datos de las liveCharts necesitaremos colecciones, con lo que tenemos que forzar en la base de datos la creación de notas dependiendo del curso en el que esté matriculado el usuario, todas con valor 0, borrándose o manteniéndose al cambiar de curso tal y como hayamos seleccionado en el formulario Ayuda.

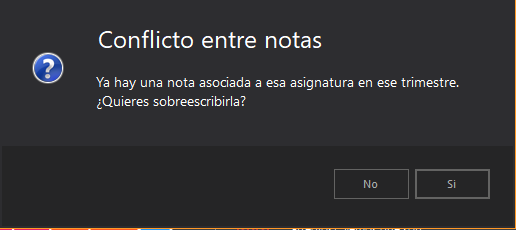
La primera carga de datos se hará desde el formulario Principal, ya que el método InitComponents() se encuentra en el constructor de la clase para mayor facilidad a la hora de acoplarlo en el panel.



Pantallazo de función agregar nota sin hacer login o no tener curso asignado



Pantalla cuando se ha realizado el login, mostrando las notas que posee el usuario en el curso que está matriculado, y la colección de enlaces que el usuario ha subido.



Mensaje al encontrar una nota mayor que 0 al querer agregar una nueva

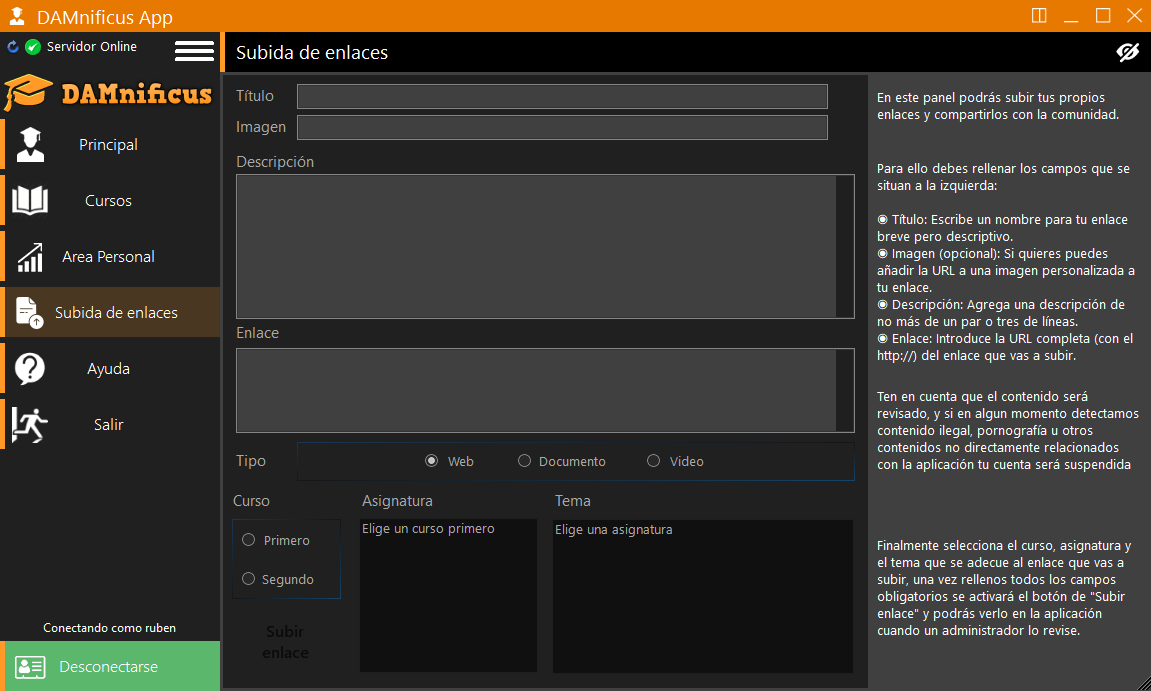


Nota agregada correctamente (notese la diferencia, primera barra azul de la gráfica)

### **Pantalla Subida de enlaces**

En este formulario el usuario podrá subir sus propios enlaces a la aplciación, pudiendo ser visibles por el resto de usuarios una vez hayan sido revisados por un administrador.

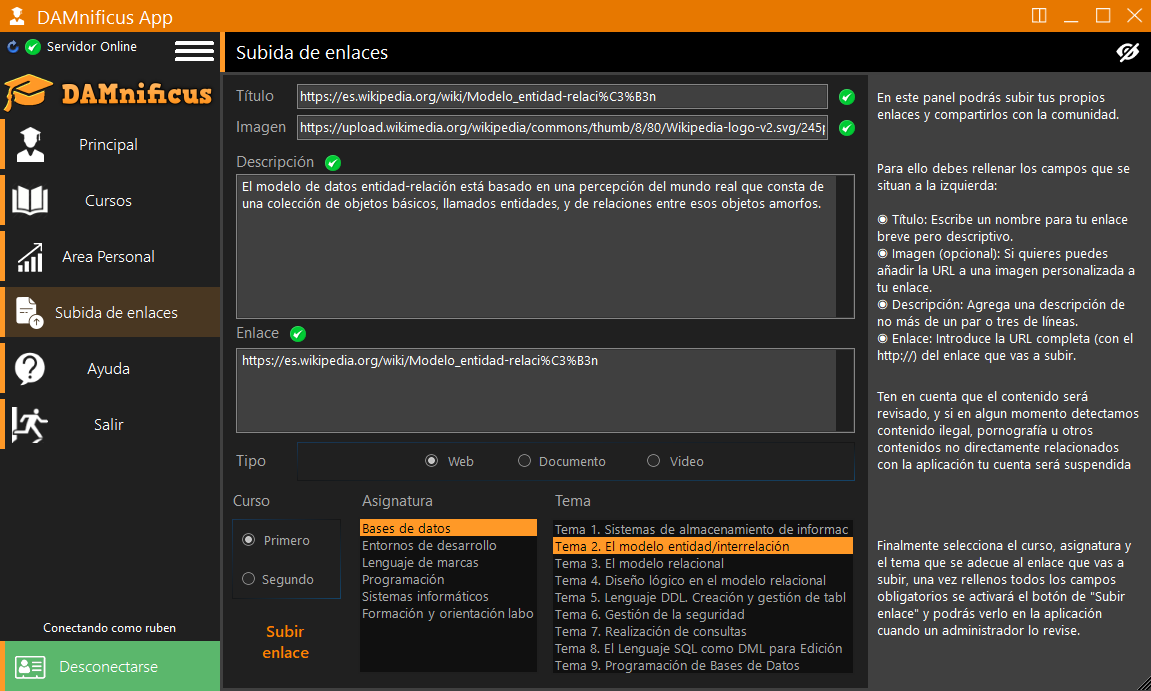
En el caso de que el usuario tenga permisos de administrador el enlace no necesitará revisión y podrá ser visto por lo demas usuarios en el momento que la subida se haga efectiva en el servidor.



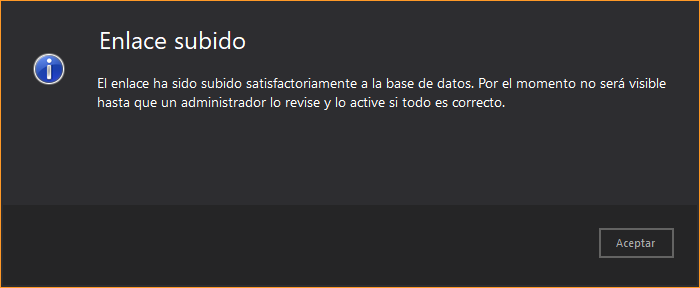
Otra característica de este formulario es que solo esta disponible para los usuarios registrados, si el usuario con permisos de invitado intenta acceder a dicha opcion a través del menu lateral, la aplicación mostrará un mensaje explicando la situación e invitará al usuario a registrarse, enviandolo al panel principal tras aceptar el cuadro de dialogo.



Si el usuario ya esta logueado en la aplicación éste podra rellenar los campos sin ningún problema. Todos los campos excepto el de imagen son obligatorios, toda la información de cómo se rellenan está disponible en el panel lateral de la derecha, que también puede ser ocultado en cualquier momento si el usuario asi lo desea.



Una vez estan todos los campos requeridos correctamente cumplimentados se activará el botón “Subir enlace”, el cual permitirá enviar la información del enlace al servidor para que sea validada e insertada en caso de estar todo correcto. En este casos e mostrará el siguiente cuadro de dialogo indicando que el proceso de alta a terminado satisfactoriamente. En este caso el enlace pasa a revision por haber sido subido por un usuario sin privilegios de administrador.



### **Pantalla ayuda**

En este apartado el usuario registrado dispondrá de algunas opciones como cambiar la contraseña, el curso actual (para el apartado de Área personal) o enviar un reporte de error o sugerencia.



Como ocurría en la pantalla de subida de enlaces, se muestra un panel lateral a la derecha con toda la información necesaria para que el usuario conozca todas las funciones que aquí se presentan. Este panel se puede ocultar pulsando el icono del ojo arriba a la derecha.

En el panel principal del formulario de ayuda tenemos como primera opción el cambio de contraseña. Para cambiar la contraseña es necesario que el usuario introduzca la que tiene actualmente primero, en este punto sólo se valida que se rellene el cuadro de texto con algo, no se verifica si la contraseña es correcta, esto se hará mas adelante por parte del servidor.

Una vez introducida la nueva contraseña se activará el botón “Aceptar” permitiendo enviar la información al servidor, donde en caso de ser correcto se devolverá un mensaje de confirmación en un cuadro de dialogo.



La siguiente opción es la de cambiar el curso actual del usuario, esta función permite que el área personal se cargue con las notas que el usuario introduzca en dicho formulario.

Por el contrario, si el usuario no se ha conectado y entra en este formulario sólo tendrá acceso a la opcion de recuperación de contraseña olvidada, en el cual podrá introducir su correo para que un token de autenticación sea enviado a esa dirección.

El token se deberá introducir en el campo que se habilitará para este cometido, una vez el token es introducido y validado se activarán otros dos campos de texto para introducir la nueva contraseña y cambiarla.

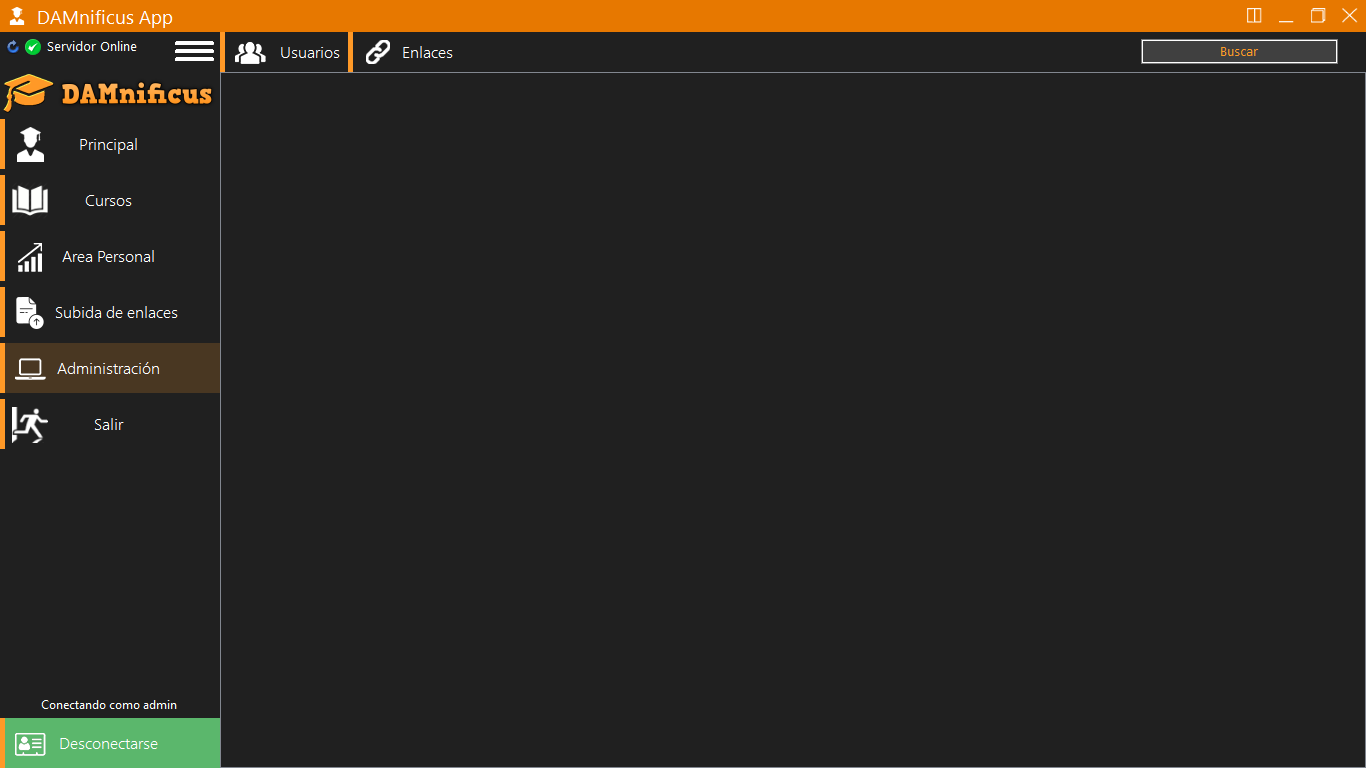
### **Pantalla administración**

Este formulario solo es visible para el usuario Admin o usuarios con privilegios de admin. Aquí podrán ver y gestionar los distintos usuarios y enlaces de la aplicación, pero con algunas distinciones entre estos ya que el usuario Admin tiene funcionalidades que uno con privilegios de admin no.

Diferencias entre los dos tipos de usuarios que pueden acceder a esta pantalla

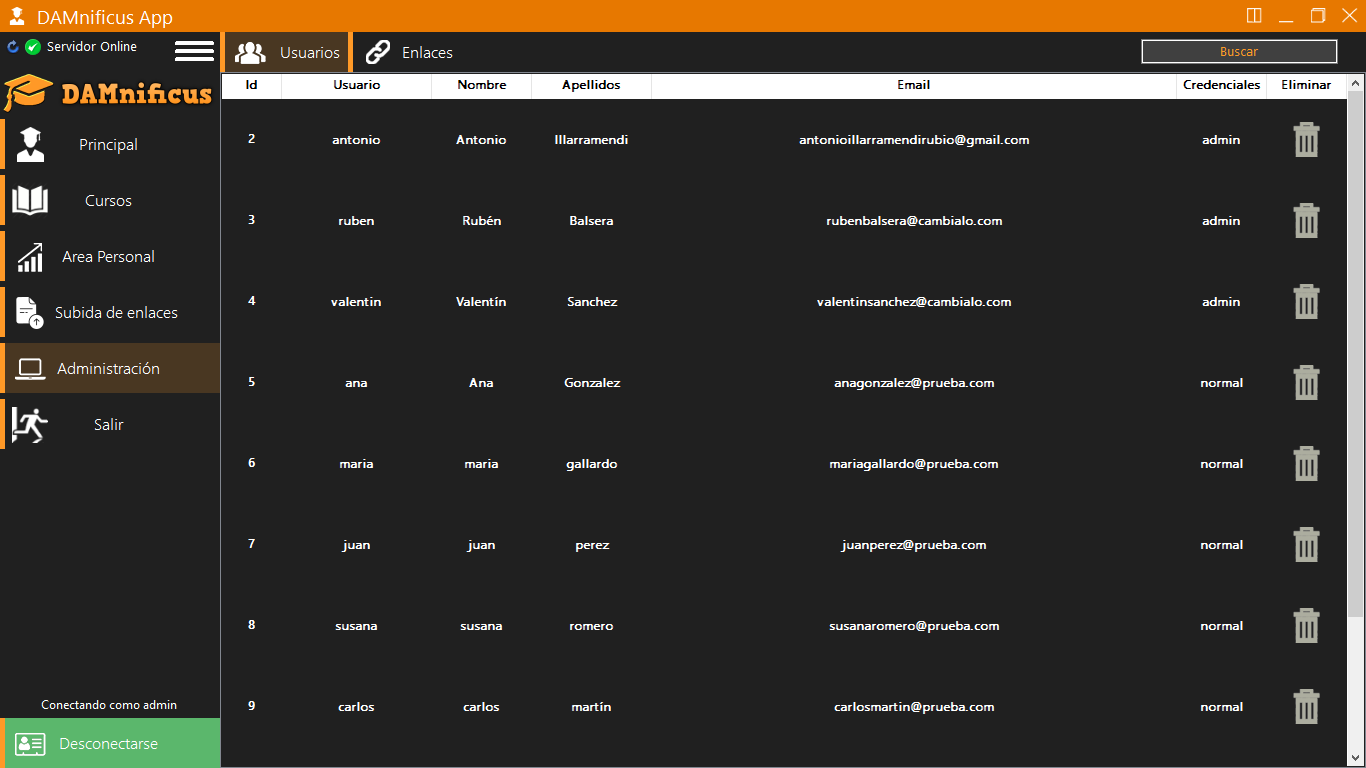
|  |  |  |
| --- | --- | --- |
| Funcionalidades | Admin | Privilegios  de admin |
| Cambiar a estado Activo un enlace | X | X |
| Cambiar a estado Revisión un enlace | X | X |
| Cambiar a estado Caído un enlace | X | X |
| Conceder privilegios de admin a un usuario normal | X | X |
| Quitar privilegios de admin a un usuario | X |  |
| Borrar un usuario normal | X | X |
| Borrar un usuario Admin | X |  |

Al hacer click sobre Administración veremos la siguiente pantalla donde tenemos que elegir lo que queremos visualizar, el listado de Usuarios o de Enlaces.

****

Si seleccionamos la pestaña Usuarios estando logueado como Admin:

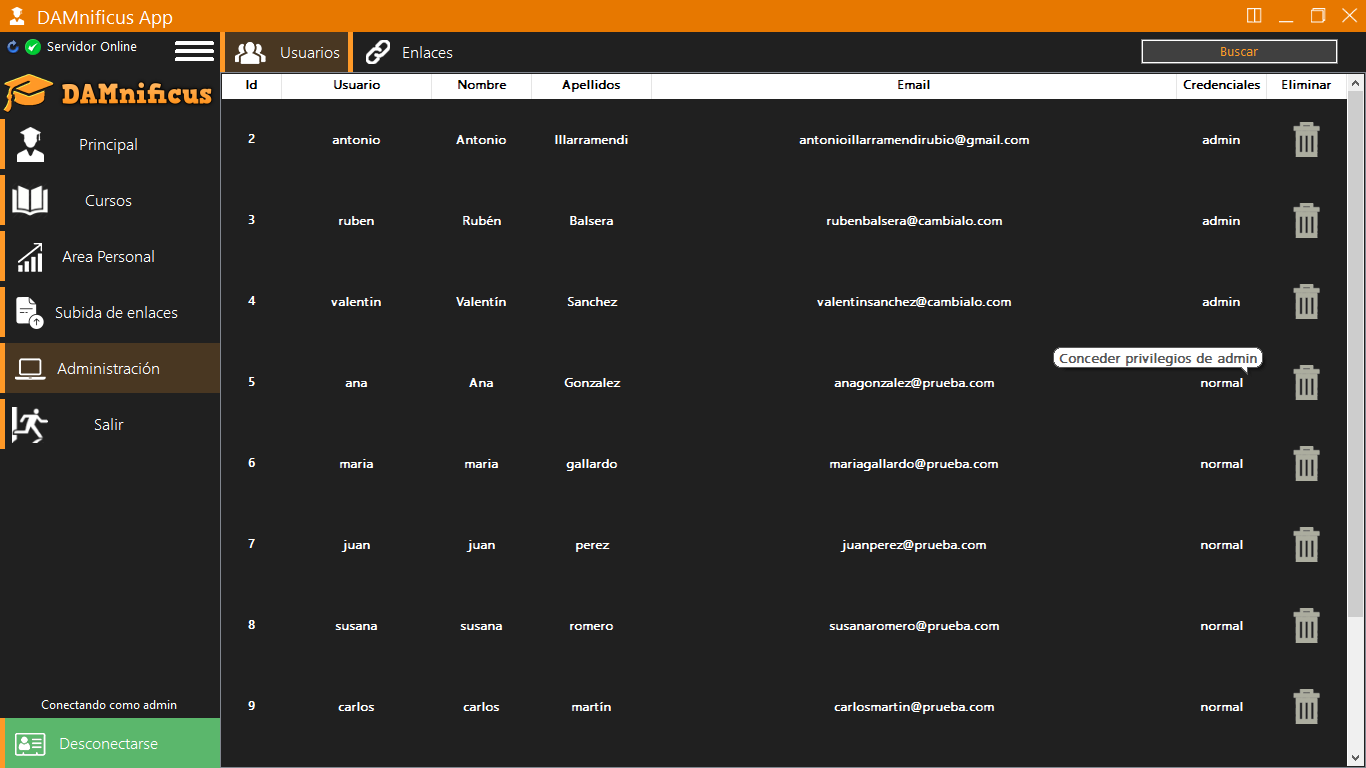
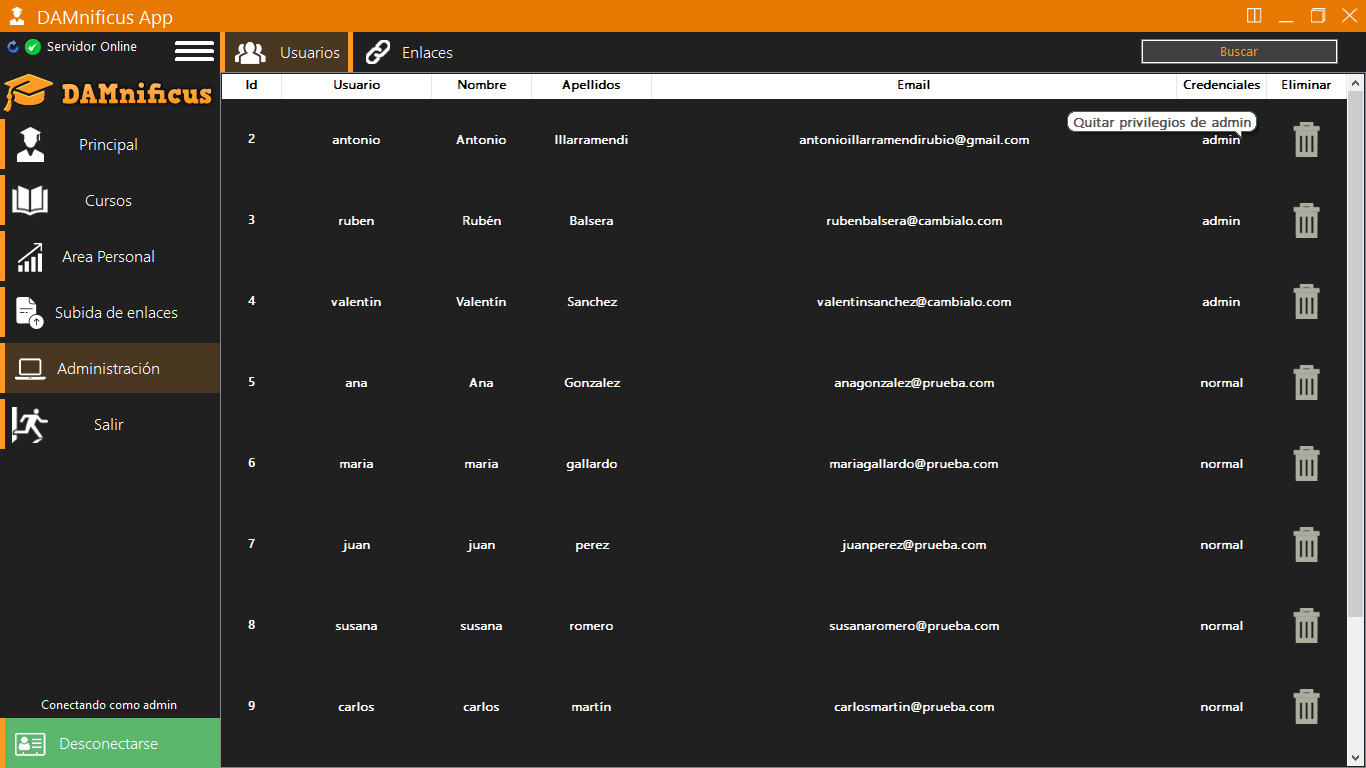
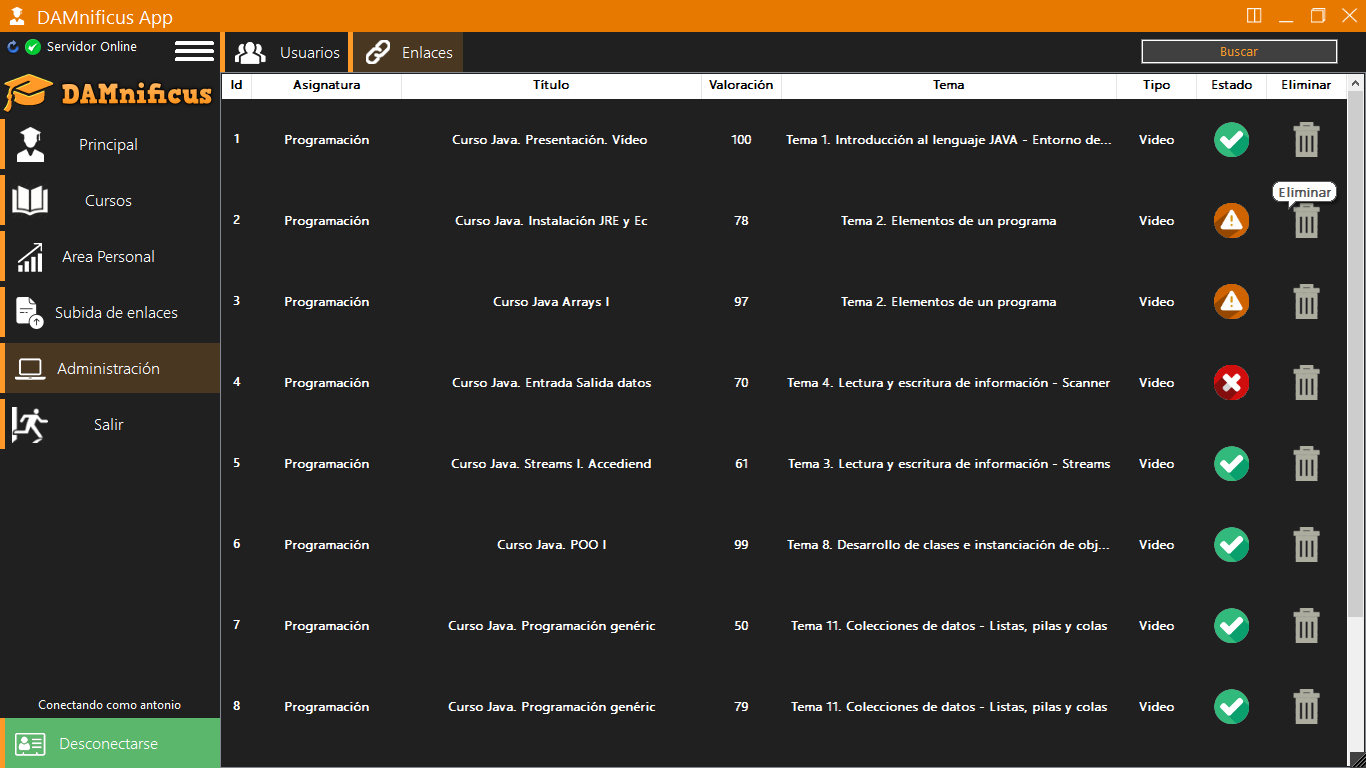
Nos mostrará la siguiente pantalla donde veremos a todos los usuarios que están registrados en la aplicación excepto al usuario Admin que es el nuestro. De esta manera evitaremos que pueda modificarse a sí mismo.



Este usuario es el principal de toda la aplicación con lo cual no tiene restricciones y puede borrar a todo tipo de usuarios independientemente de los privilegios que tengan, por ello tiene el icono de la papelera (eliminar) disponible en todos. Al hacer click sobre la papelera se muestra un mensaje de confirmación, si la respuesta es afirmativa, se mandara una petición al servidor para borrar ese usuario y si todo es correcto se eliminara de la lista.

Además puede conceder o quitar los privilegios de Admin a cualquier usuario sin previa confirmación, se envíara al servidor la petición de cambiar credenciales y si todo es correcto, cambiará de admin a normal o viceversa.

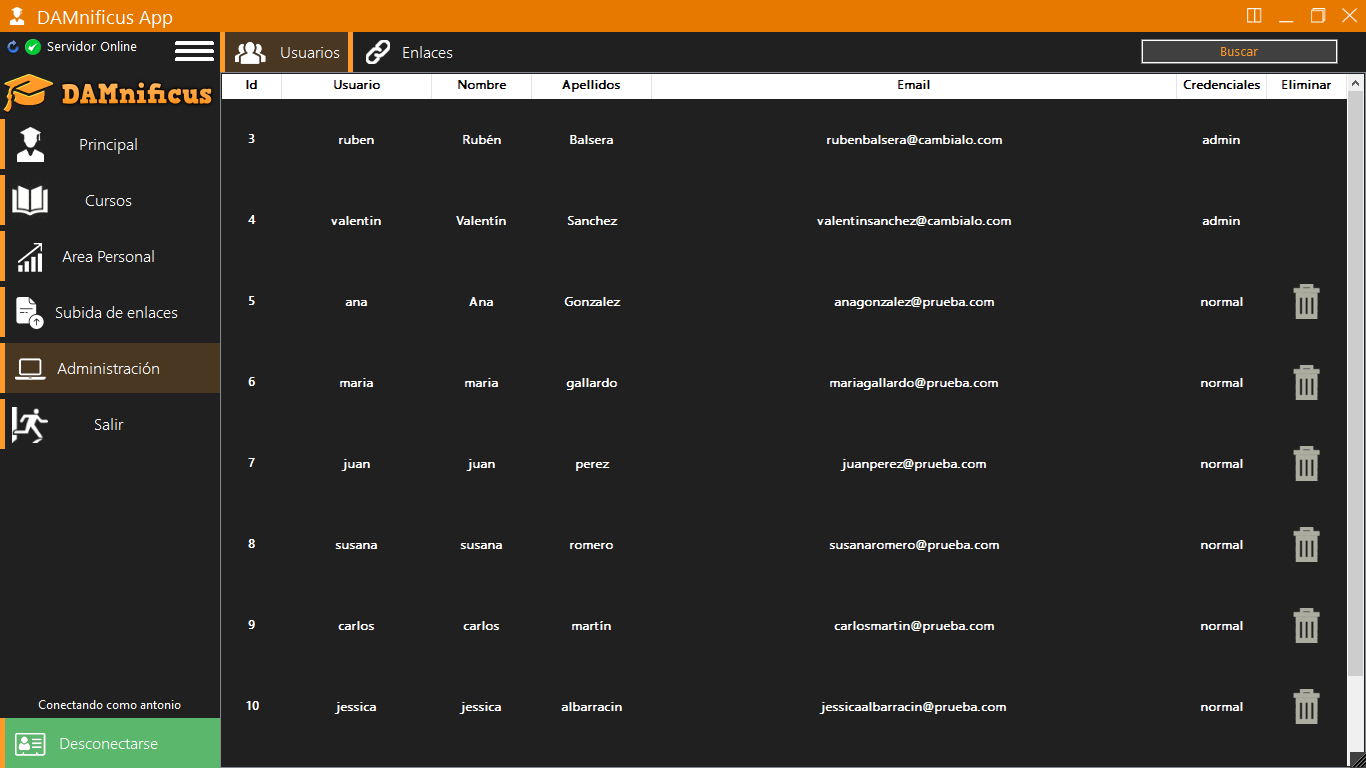
Mensajes Tooltip que se pueden visualizar en esta pantalla:



Las columnas Título y Tema también cuentan con tooltip para poder mostrar toda la información en caso de que sea más extensa que el espacio del que dispone.

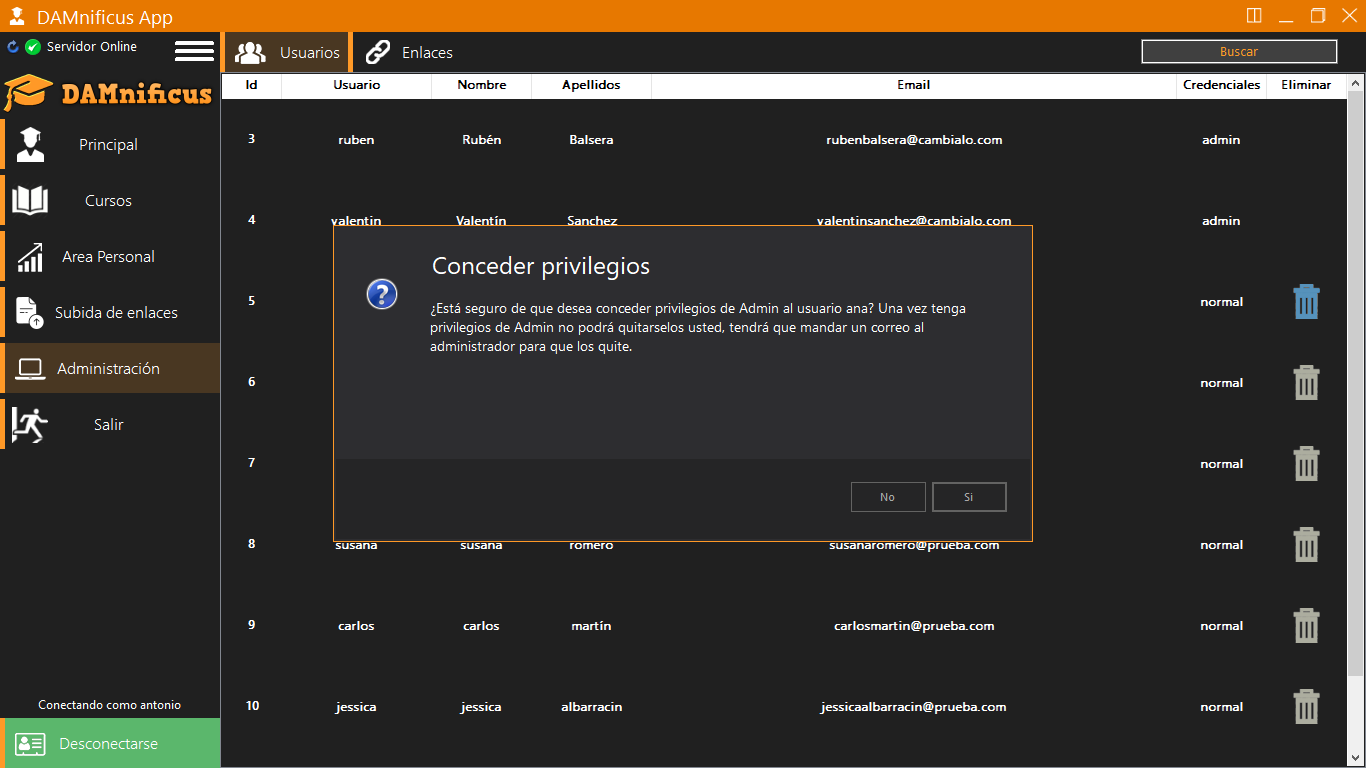
Estando logueado como usuario con privilegios de Admin:

Nos mostrará la siguiente pantalla donde veremos a todos los usuarios que están registrados en la aplicación excepto al usuario Admin, ya que no tiene que tener constancia de que existe y al nuestro propio con el que estemos logueados.



Este usuario puede borrar a todos los usuarios con credenciales “Normal”, pero no a los usuarios con credenciales de “Admin”. Por ello se ha deshabilitado la opción de eliminar quitanco el icono papelera.

También puede conceder los privilegios de Admin a un usuario Normal, al hacerlo se le muestra un mensaje de confirmación informando que esos privilegios no podrán ser revocados y que para ello tendrá que mandar un correo electrónico al Administrador de la aplicación.



Una vez seleccionada la opción “Si” hará el mismo proceso que con el usuario Admin explicado anteriormente. Al finalizar quitara el icono papelera de ese usuario.

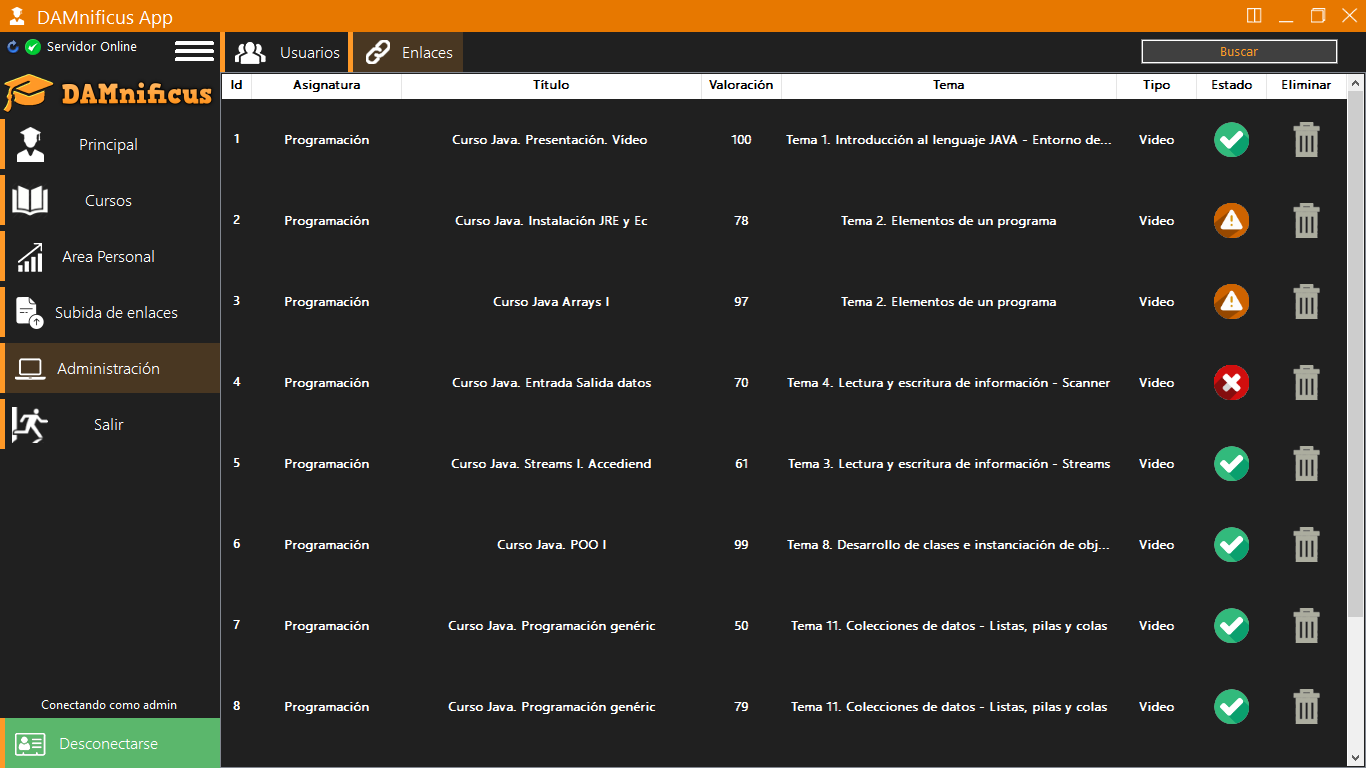
Los mensajes tooltip para este usuario serán los mismos excepto si el usuario es tiene privilegios de admin ya que estará deshabilitado.

Si seleccionamos la pestaña Usuarios:

En esta pestaña las funcionalidades son iguales para los dos tipos de usuarios.

Estando logueado como usuario Admin o con privilegios de Admin.

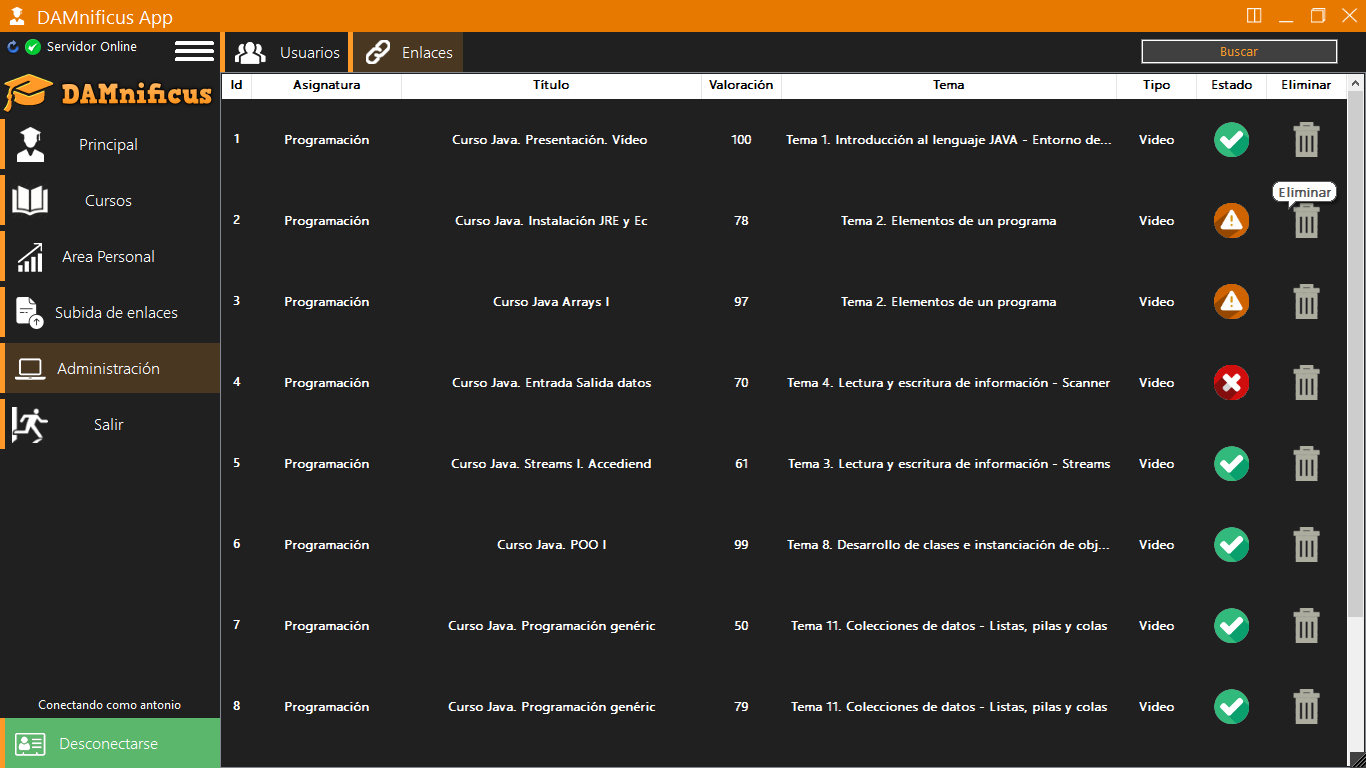
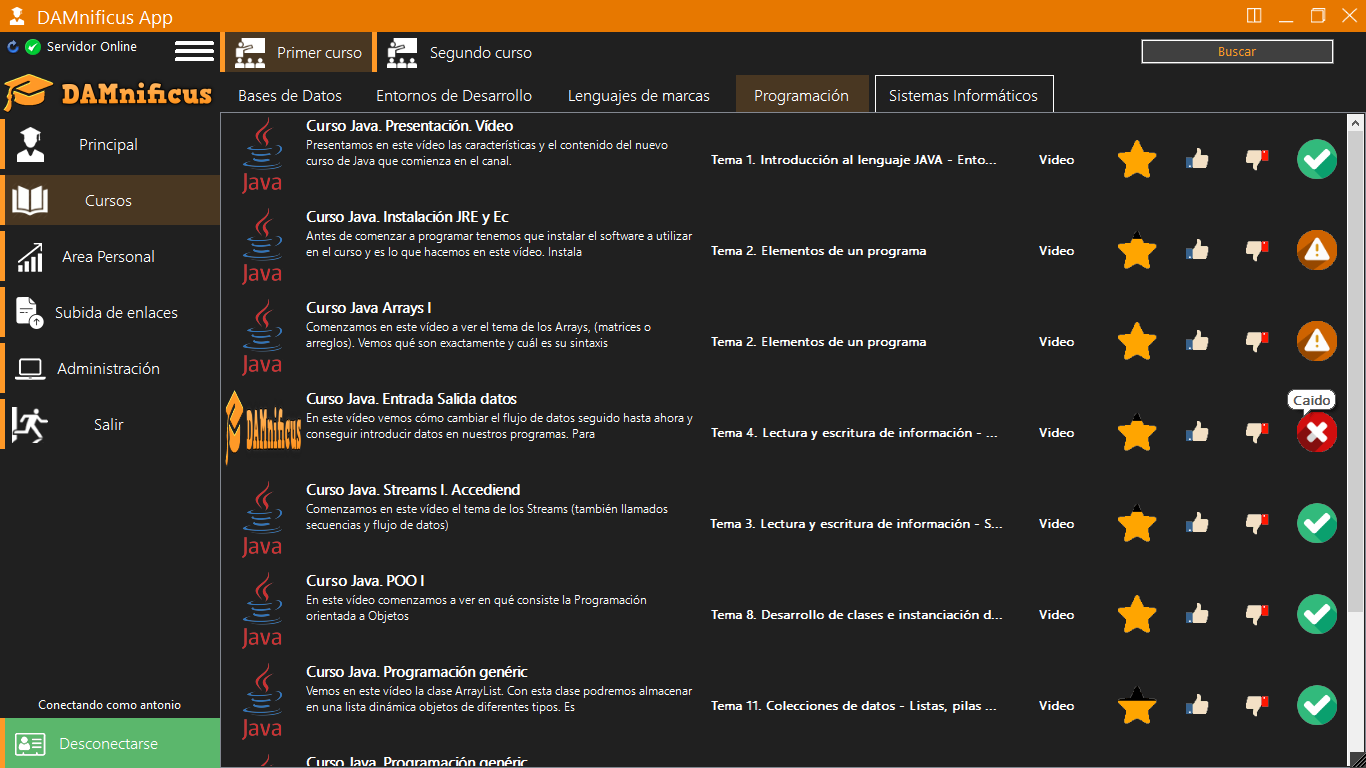
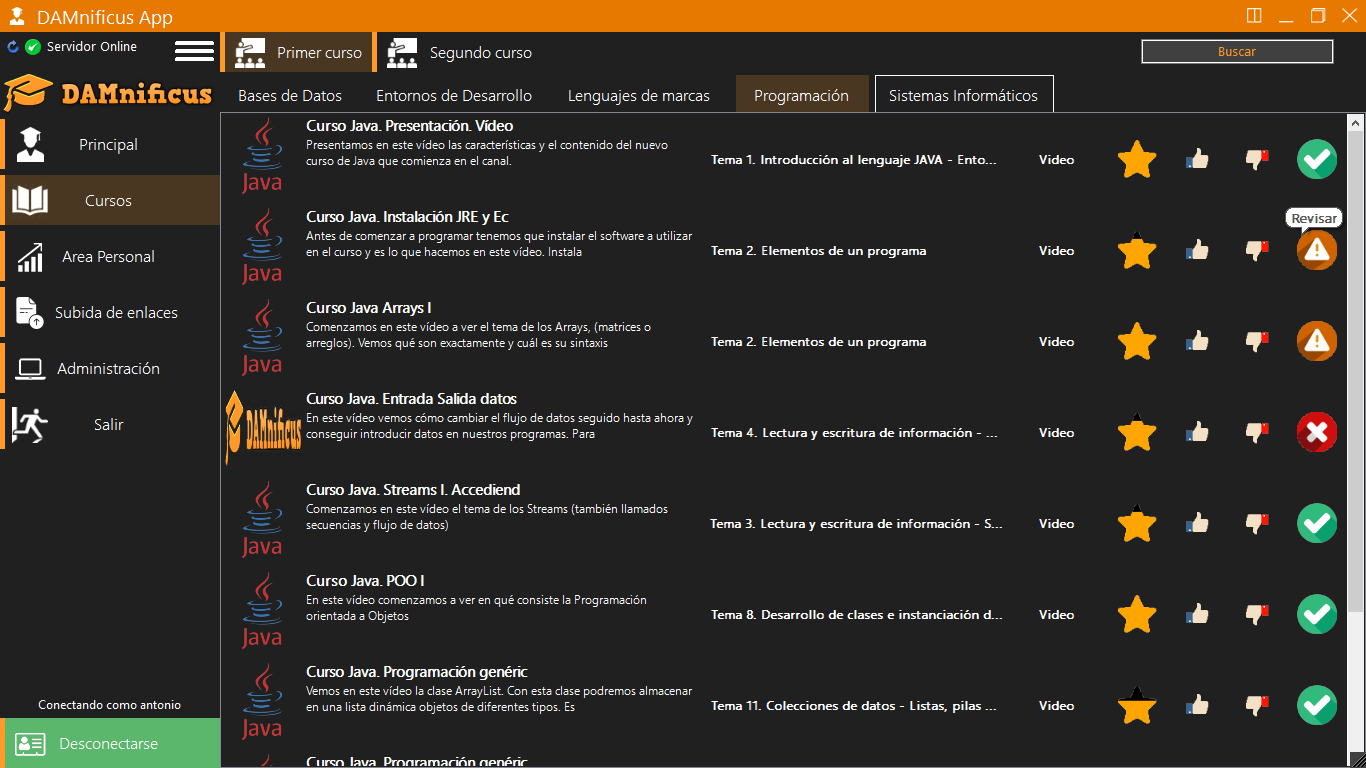
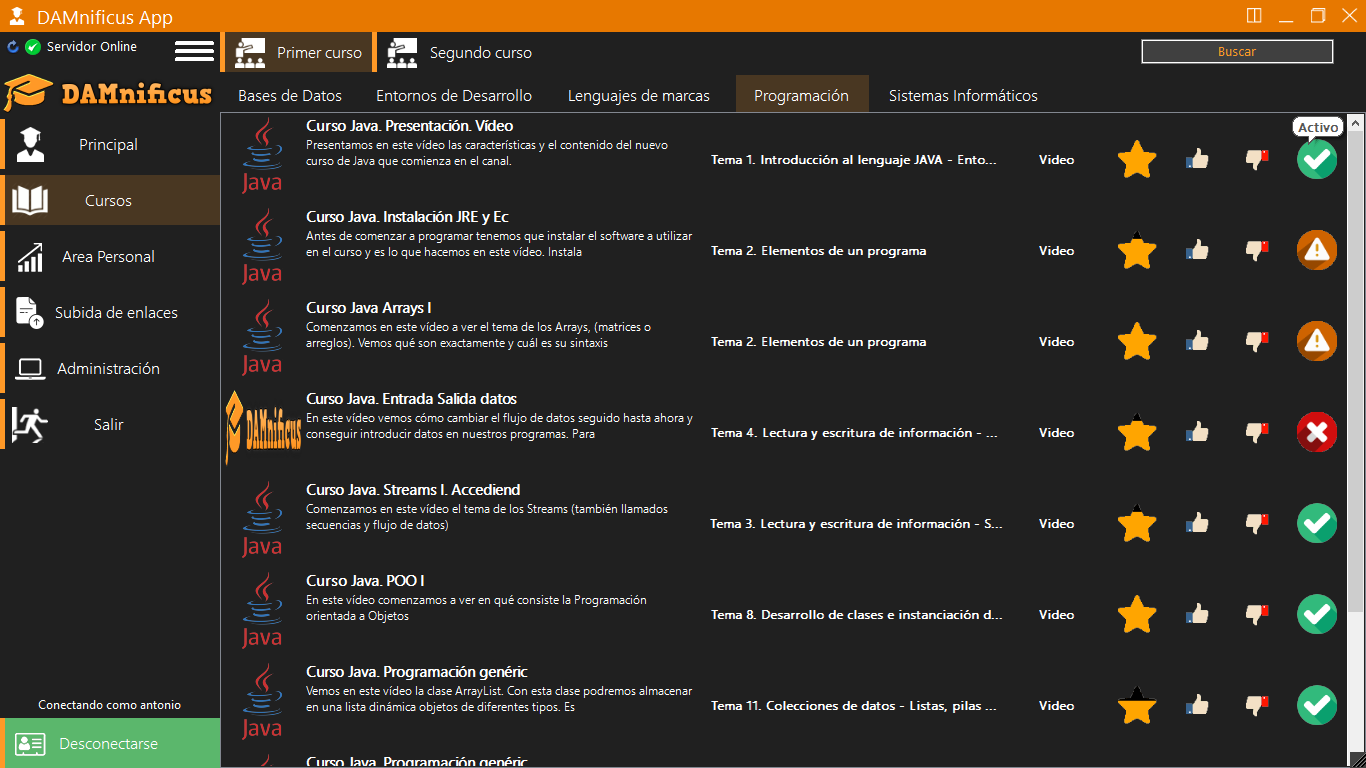
Nos mostrará la siguiente pantalla donde veremos todos los enlaces de la aplicación.



Podemos eliminar cualquier enlace si confirmamos que no funciona o si se trata de contenido inadecuado, haciendo click sobre el icono papelera. Nos mostrará un mensaje de confirmación, si la respuesta es afirmativa manda una pertición al servidor para eliminar el enlace seleccionado. Si todo es correcto esa fila se eliminará de la lista.

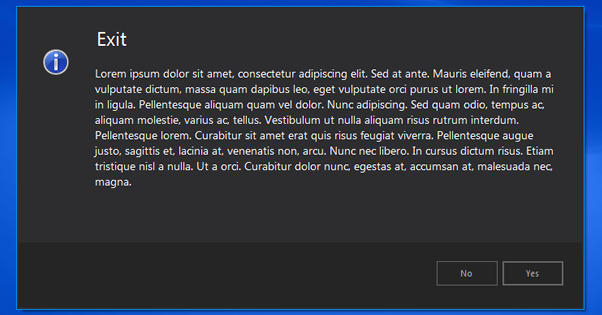
También podemos cambiar el estado de lo enlaces, por ejemplo si ha sido reportado como caido y no lo está. Haciendo click sobre el icono del estado, suponiendo que un enlace este “Activo” con el icono “Tic Verde”, se envía una petición al servidor de cambiar estado de “Activo” a “Revisar” si todo es correcto, cambiará el icono a “Triángulo advertencia”. Si se hace click sobre este se repetirá la acción cambiando de “Revisión” a “Caído”, si todo es correcto cambiará el icono a “Circulo rojo con X” y si se hace click sobre este último se repetirá la acción cambiando de “Caído” a “Activo”.

Mensajes Tooltip que se pueden visualizar en esta pantalla:



### **Mensajes de dialogo**

Como se ha podido observar en los pantallazos de la aplicación cliente hemos usado unos cuadros de dialogos personalizados, para que no desentonaran con el diseño general de la aplicación.



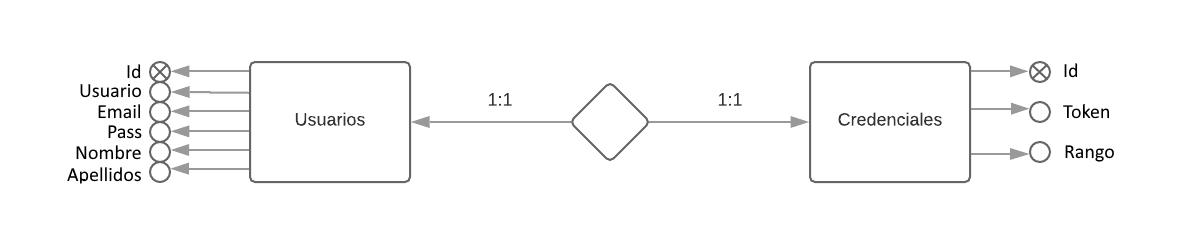
Éstos cuadros de dialogo han sido implementados gracias al código con licencia *The Code Project Open License (CPOL) 1.02* que se presenta en el siguiente enlace <https://www.codeproject.com/Articles/17253/A-Custom-Message-Box> y creado por el inglés **Syed M Hussain**.

# Diseño de la base de datos

En este apartado se tratan los aspectos de diseño relacionados con la base de datos del servidor. En un principio se muestra un diseño modelo entidad-relación, al final del apartado se muestra el detalle de tablas.

## Modelo Entidad-Relación

Para este programa hemos creado dos bases de datos independientes, una para los usuarios y sus credenciales; y otra para los enlaces y la información personal de los mismos.



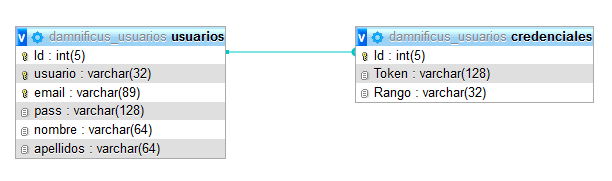
La primera ilustración muestra un simple esbozo del esquema entidad relación de la base de datos de los usuarios. Como puede verse, el usuario está en relación con los atributos de sus credenciales, como el rango y el token. En la siguiente imagen tenemos el esquema E/R de la base de datos de enlaces:

En esta segunda imagen se muestra el modelo ER que se creó para la base de datos de los enlaces, como se puede observar, el modelo es también bastante simple y no fue ningún problema trasladarlo al modelo relacional.

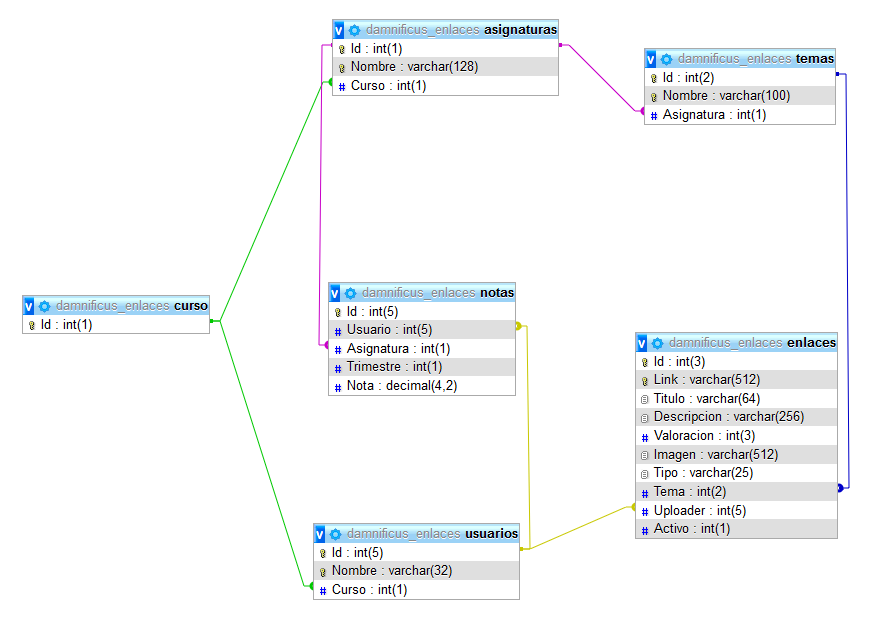
## Diseño final base de datos

Aquí podemos observar el diseño que finalmente se ha implantado en la base de datos, como ya vimos en el apartado anterior, la estructura del modelo ER facilitó en gran medida la creación de la base en MySQL.

En primer lugar, tenemos la base de datos de usuario, que se muestra de forma detallada a continuación:



En la siguiente imagen podemos ver también a todo detalle el diseño de la base de datos de los enlaces.



## Diseño de la seguridad

Con el fin de evitar ataques y brechas en la seguridad, se decide cifrar los datos de los usuarios como las contraseñas y cierta información sensible que éstos pudieran enviar, cómo el token de autenticación o sus datos personales.

Con esto se consiguen dos cosas, la primera, aunque alguien consiguiera acceso a la base de datos y extrajera las contraseñas de los usuarios, éstas estarían cifradas, teniendo así que recurrir al uso de una fuerza bruta muy costosa.

Y la segunda es que el envío del JSON al con la contraseña y los demás datos también está cifrado, dando como resultado en un cifrado doble para las contraseñas en el momento en que circulan por la red.

Además, el cifrado es bidireccional, las respuestas que el servidor envía al cliente son también cifradas.

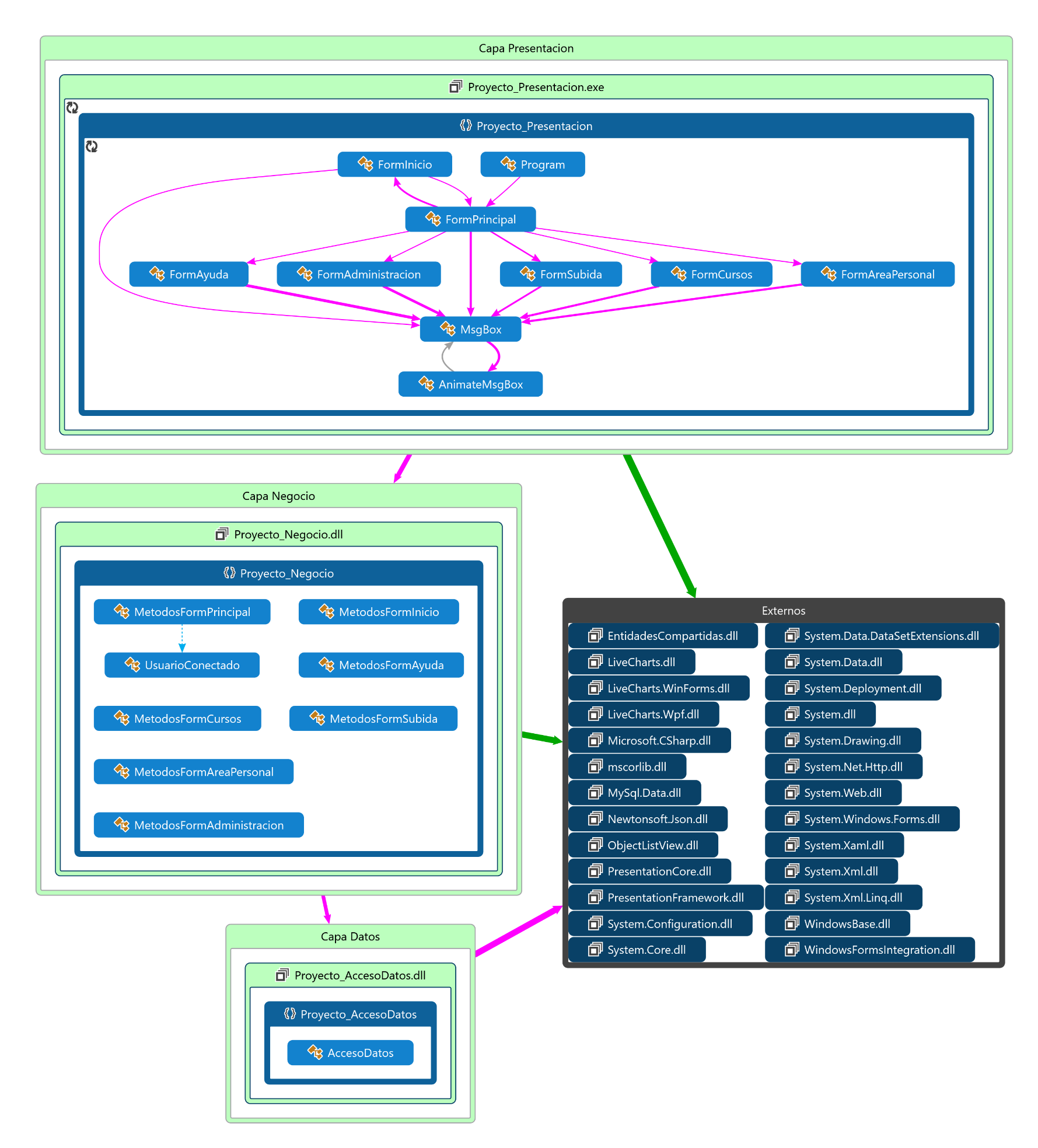
En el apartado de implementación se detallará más a fondo el cifrado y el envío de datos por JSON.

## Diagrama de clases

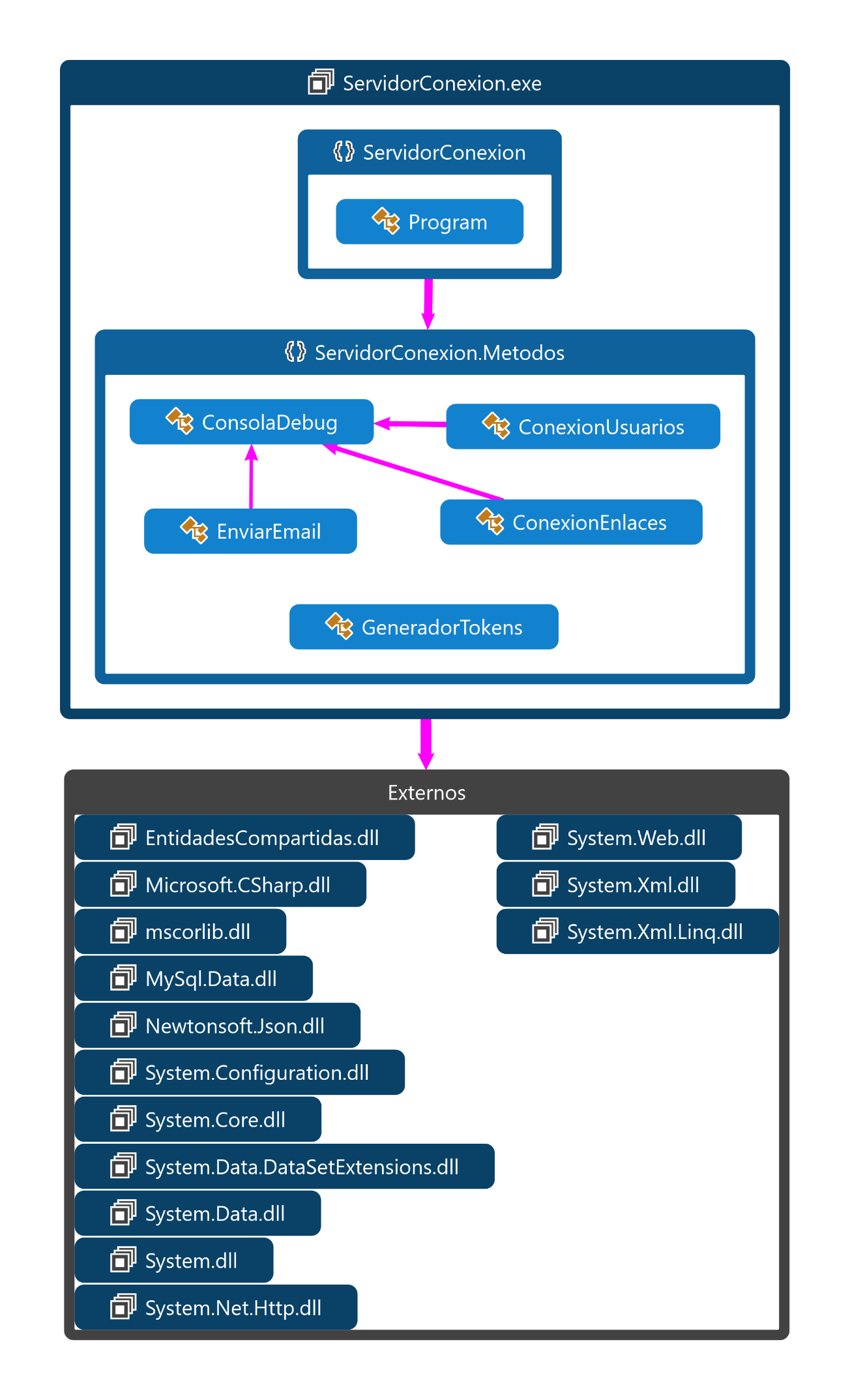
Como se puede deducir de los apartados anteriores, a parte del diseño a capas de la aplicación, existen entidades o clases que son compartidas entre los dos programas que componen el proyecto (cliente y servidor). El problema que nos encontramos fue la redundancia de algunas clases que eran idénticas para los dos ejecutables, y el peligro de cometer un error al modificar uno de ellos y no hacerlo igual en el otro.



Para ello, como se puede ver en el esquema anterior, se ha creado una biblioteca de clases .dll que será usado en ambas compilaciones y que nos ha facilitado el desarrollo del proyecto, ya que cuando se cambiaba alguna de estas clases estaba garantizado que se cambiaba en ambos programas.

La siguiente imagen representa el diagrama de clases de la aplicación cliente, y como las clases y las capas interactúan entre sí, véase que como se mencionó en la explicación del modelo a capas, las conexiones entre éstas son siempre a la siguiente capa, en la misma dirección. También hemos añadido las librerías externas usadas:

El diagrama de clases del servidor, como es obvio, es mucho más sencillo, pues su función únicamente es tratar las peticiones que envíen los clientes:



## Conclusiones

Se presenta este apartado al final del extenso capítulo de diseño, a modo de resumen, y con el objetivo de hacer una síntesis de las decisiones tomadas, que afectarán en gran medida en el siguiente capítulo, implementación de la aplicación. Se destaca, por tanto, del capítulo de diseño, los siguientes puntos:

* La arquitectura del proyecto se basa en el modelo cliente-servidor, donde el cliente será la aplicación de escritorio, y el servidor una base de datos basada en *MySQL*.
* Se ha elaborado un diagrama de pantallas específico con transiciones entre ellas y recursos que utilizarán.
* Se crean unas maquetas que intentarán acercarse lo máximo posible al producto final.
* Asimismo, se diseña la base de datos SQL con un diagrama entidad relación, especificando del mismo modo los campos de las tablas.
* Sólo los procedimientos almacenados en la capa Acceso a Datos podrán contactar con el servidor, el cual será el responsable de los accesos y modificaciones en la base de datos del proyecto, alojada en un equipo externo.
* Los datos que devuelva el servidor deben estar serializados en JSON, con el fin de que su tratamiento por la aplicación sea más sencillo.
* Los datos son encriptados para agregar la máxima seguridad posible al conjunto, en el caso de las contraseñas el cifrado es doble.

# IMPLEMENTACIÓN

En este capítulo se aborda la fase de implementación del proyecto. Se explicarán algunos detalles relevantes, así como las diferentes dificultades que se encontraron en el proceso.

En esta fase se implementa el diseño realizado en el capítulo anterior de diseño, basado en el catálogo de requisitos del capítulo de Análisis. Como se ha especificado en el capítulo anterior, se utilizará la plataforma .NET para ambos programas (cliente y servidor) y se utilizará una base de datos MySQL con XAMPP.

A lo largo del capítulo se tratarán los siguientes aspectos:

* Elementos básicos de la parte cliente, módulos, librerías utilizadas e interfaz.
* Aspectos relevantes de la parte servidor, así como parte de su implementación.

# Parte cliente

Como se ha especificado en capítulos anteriores, la parte de cliente la constituye la aplicación Android. En los siguientes apartados se intenta dar una visión de los elementos básicos que la componen.

## Elementos básicos

Los primeros elementos que tratar, y con el fin de dar una perspectiva general de la parte cliente, serán los diferentes módulos implementados. Con módulos se intenta representar funcionalidades relacionadas y agrupadas dentro de un mismo proyecto.

## Capa Acceso a datos

Ésta es una de las partes principales de la aplicación cliente, aquí se gestionan las peticiones que se van a enviar al servidor, tratándolas si fuera necesarios y finalmente encriptando los datos para que el envío sea lo más seguro posible.

Dentro de esta clase, el método más importante es **enviarPetición**, y es la responsable de hacer la conexión http y de enviar los datos necesarios a través de un JSON cifrado al servidor de la aplicación.

public async Task<Respuesta> **enviarPeticion**(string peticion, string usuario, string pass, string token, Dictionary<string, string> datos)

Como se puede observar, se trata de un método asíncrono, y en la mayoría de los casos veremos que la aplicación se compone básicamente de éstos, pues es de vital importancia que la aplicación siga respondiendo al usuario aun cuando la petición no ha sido respondida.

Una vez el servidor ha tratado la petición, éste envía una respuesta al cliente que hizo dicha petición. Esta respuesta será tratada en consonancia con la petición que se envió. Una vez tratada será enviada a la capa de Interfaz de usuario mediante la capa negocio, que se encargará de transportar los datos necesarios a la interfaz de usuario.

*En los proyectos Acceso a datos de la solución se encuentra el código comentado con toda la lógica de conexión con el servidor.*

## Capa de Negocio

En esta capa de la aplicación se tratan los datos o simplemente hace de intermediario entre las capas de interfaz y la de acceso a datos.

Aquí se encuentran los métodos para la lógica de la interfaz, llamados todos como *Métodos* seguido del nombre del formulario al que está referenciando, es decir, el formulario principal llamado *FormPrincipal* en la capa de presentación tendrá sus métodos en la clase *MetodosFormPrincipal* de la capa de negocio y así ocurre con el resto de los formularios de la aplicación.

Como ya se explicó en el apartado de Diseño a Capas, solo recibe las entradas desde la interfaz y solo envía datos a la capa de acceso a datos.

## Gestión de panel principal

Según el usuario inicia la aplicación, lo primero que se encontrará será el panel principal, en el apartado de Diseño se han comentado los diferentes componentes de los cuales se conforma, pero aquí detallaremos un poco más en lo que se refiere a la parte lógica.

Como ya se ha comentado en apartados anteriores decidimos hacer una interfaz desde cero, prescindiendo de la mayoría de los añadidos que contiene el ya clásico componente *Windows Form* y que no son muy potentes en lo que a personalización se refiere.

Como se estableció el atributo **FormBorderStyle** de los Formularios a *none*, desaparecían la mayoría de las funcionalidades básicas de una ventana de Windows 10, como los botones de control (minimizar, maximizar, cerrar), la barra de título y sus funciones o el redimensionado de la ventana.

Debido a esto se optó por programar manualmente dichos componentes a base de paneles y botones, además de añadir los eventos y métodos necesarios para volver a restaurar las funcionalidades perdidas.

Además, debido a la versatilidad que nos ofrecía esta nueva interfaz, pudimos añadir animaciones al inicio y final de la aplicación, a algunos menús y a los mensajes de dialogo.

*En los proyectos Proyecto\_Presentacion y Proyecto\_Negocio de la solución se encuentra el código comentado con los eventos y la lógica respectivamente.*

## Gestión de *login* de usuarios

En la pantalla principal, como ya hemos comentado en el apartado diseño, se encuentra la caja de *login*, en ésta el usuario puede introducir sus datos y acceder a las funciones completas de la aplicación.

Para llevar esto a cabo se obtienen el texto de los campos de usuario y contraseña (que han sido validados con anterioridad por la interfaz). Estos dos *strings* son cifrados y enviados a través del método *comenzarLogin* de la clase Acceso a Datos. En este método en primer lugar se hace una petición para obtener la *salt* del usuario introducido. La *salt* es la clave con la que se cifró la contraseña del usuario cuando se registró.

Una vez la *salt* ha llegado al cliente se usa para cifrar la contraseña introducida por el usuario en la caja de *login* para ser enviada de nuevo al servidor y que este último haga la comparación con la información guardada en la base de datos.

Si la comparación es correcta se generará un token de sesión que se guardará en la base de datos y que también será enviado al cliente como respuesta al *login* correcto.

Asimismo, a partir de este momento el usuario y contraseña no serán necesarios para las siguientes peticiones, pues el token será el que se envíe (convenientemente cifrado) para autenticar al usuario, comparándolo con el token guardado en la base de datos.

Si el nombre de usuario o la clave son incorrectas, el servidor enviará la respuesta pertinente al cliente informando de la situación.

Si el usuario decide desconectarse o salir de la aplicación se enviará una petición al servidor para borrar el token de sesión para que el usuario obtenga uno nuevo al volver a conectarse.

## Gestión de registro de usuarios

Para la gestión del registro decidimos que fuera lo más simple y cómodo posible para el usuario, evitando pasos engorrosos o pedir información poco relevante para la aplicación.

Además, se ha implementado la función de registro dentro del primer formulario que se carga al iniciar la aplicación, *FormInicio*, para que sea fácilmente accesible por el usuario, sin tener que acceder a laberínticos submenús.

Si el usuario ya está registrado y hace *login* en la aplicación desaparecerá la opción de registrarse, pues se considera que ese usuario ya está registrado obviamente.

Una vez comenzado el proceso de registro, el usuario tendrá que rellenar los textboxes que se muestran en el panel. Estos textboxes están validados y no permitirán en ningún caso que el usuario deje alguno vacío o la información no esté correctamente introducida. En el caso del email, se comprueba si es un email bien formado.

El textbox que se usa para el campo del “Nombre de usuario” tiene además una validación extra, comprobando si el nombre introducido existiera ya en la base de datos, con lo que se informa al usuario que debe escoger uno distinto.

Estas validaciones tienen su propio icono descriptivo y un tooltip informando del estado de la caja de texto.

Una vez todos los campos están validados y correctamente rellenados se activará el botón que permitirá a la aplicación enviar un correo a la dirección introducida por el usuario con un token que genera el servidor para autentificar dicha dirección de email.

*La generación de token y el envío de emails se detallarán más adelante en la Parte servidor.*

En esta parte del registro el usuario deberá consultar su correo e introducir el token que se le envió. La caja de texto donde se debe escribir (o pegar) el token lo validará, activándose el botón aceptar.

Si el usuario introduce el token mal o no lo introduce no podrá continuar con el registro y dicho token se destruirá en cuanto cambie de formulario desde el menú.

Cuando el usuario pulsa aceptar para finalizar el registro se envía una petición al servidor con los datos de dicho usuario para que éstos sean introducidos en las dos bases de datos de las que se compone la aplicación (usuarios y enlaces).

El proceso de introducción sigue una lógica transaccional, para evitar inconsistencias en la base de datos, si de algún modo hubiera un problema en alguna de las inserciones, se desharían las anteriores (rollback) y se informaría tanto en la interfaz como en el servidor de que ha surgido un problema en el registro del usuario.

Una vez que el registro se ha completado satisfactoriamente, el usuario puede pulsar aceptar de nuevo para loguearse automáticamente en la aplicación y poder acceder a todas las funciones que esta ofrece.

*En los proyectos Presentación y Negocio de la solución se encuentra el código comentado con los eventos (FormInicio) y la lógica (MetodosFormInicio).*

## Gestión área personal del usuario

**Obtención del Id de usuario:** Para realizar todas las consultas posteriores y simplificarlas a la hora de código SQL para la consulta, se hace una petición a través de las capas de Negocio y Acceso a Datos para la devolución de un dato tipo *Integer* para, en las futuras consultas, poder referenciar al usuario que está logueado directamente por su Id en la tabla de Usuarios y no tener que hacer una subconsulta a través de *UsuarioConectado.nombre*.

**Obtención del curso:** Al igual que con la obtención del Id de usuario, para futuras consultas necesitamos el curso en el que el usuario está actualmente matriculado, y el cual se cambia a través del formulario Ayuda. Se reutilizará exactamente la misma petición a nivel Acceso a Datos para la obtención del curso, salvo que el tratamiento a nivel de la capa de Negocio, ya que el tipo de datos devuelto es tipo *string* con el formato "cursoX" siendo X el curso actual, en este nivel solo se retornará un *Integer* referente al curso y se almacenará durante toda la ejecución del formulario. Con el curso recogido se inicializarán de forma diferente los componentes, destacando el TrackBar ya que si el curso es 1 tendrá 3 posiciones distintas y si es 2 tendrá solo 2. Sin embargo, si el curso que nos devuelve la petición es 0 o bien el usuario no ha hecho login en la aplicación, que también cargará 0, el formulario está preparado para mostrar un modelo de prueba que será interactivo hasta cierto punto pudiendo interactuar con todo salvo la sección de añadir nota que mostrará un mensaje de aviso de que no está disponible sin realizar un login o matricularse en un curso.

##### **Obtención de las asignaturas:** Para la carga de la ListBox dependiendo del curso en el que está matriculado el usuario, que hemos recogido en el paso anterior, cargaremos una lista de asignaturas u otra, obviamente con la misma petición, pero dependiente del curso. Se pasará a nivel de Negocio solamente el curso, que se añadirá a un diccionario y junto con el nombre de usuario se hará la petición al servidor con el nombre **sacarAsignaturas**. Nos devolverá una lista de strings que contendrá el nombre de las asignaturas del curso que hayamos pasado primeramente por parámetro y después añadido al diccionario. Con esta lista se cargará la ListBox de la parte derecha del formulario y de las leyendas de las gráficas de notas por trimestre y notas medias.

**Notas por trimestre:** La primera carga, al igual que la carga que se realiza tras la actualización de la nota en el formulario parte de una llamada al modelo de Negocio denominada **recogidaNotas**, al que se le pasan el curso en el que está matriculado actualmente el usuario (formularioAyuda), el Id del usuario y su nombre. Éste método de la capa de Negocio lo que hace es recoger estos parámetros de entrada y agruparlos dentro de un diccionario de datos constituido como <*string*><*string*>, y haciendo una llamada a través de la capa de Acceso a Datos, realizar una petición al servidor que devolverá una lista de *Doubles*, agrupados y colocados en orden según Id de asignatura, por asignatura y trimestre, con lo que una vez devuelta la lista al formulario, dependiendo del curso matriculado colocará de 3 en 3 (Primer curso) o de 2 en 2 (Segundo curso) en listas más pequeñas que se le pasarán como atributos a las distintas series de la gráfica de notas por trimestre.

**Notas medias:** Proceso exactamente igual que la carga de notas por trimestre, salvo que ahora la petición al servidor devolverá una lista más reducida, ya que la consulta a la base de datos hará un AVG agrupándolas por asignatura, con lo que, devuelta la lista al formulario, cada serie de la gráfica de notas medias constará únicamente de un solo valor. Hay que destacar que para que colores y colocación coincidan, notas por trimestre y notas medias han sido recolectadas en el mismo orden.

**Enlaces subidos por el usuario:**Misma carga que para su propio formulario, pero aplicada solo a esta parte de éste.

**Agregar nota:** Este proceso está diferenciado según exista nota mayor que 0 en el lugar que se quiere agregar, ya que, primeramente, se recogerán tanto usuario que realiza la petición, asignatura a la que se quiere añadir la nota, y trimestre donde se quiere colocar dicha nota. Con estos datos, al igual que las demás peticiones al servidor, se realizará con el nombre de usuario y los datos en un diccionario. Ahora bien, esta primera solicitud al servidor consta de una respuesta de si hay o no hay una nota igual a 0 en el lugar donde se quiere colocar dicha nota, si la nota es igual a 0, se enviará automáticamente la petición de agregar nota, pero si la primera solicitud devuelve que, sí que hay una nota introducida y es mayor que 0, se avisará al usuario de si quiere agregar la nueva nota o mantener la que estaba anteriormente. Si el usuario decide mantener la nota existente, el proceso acabará aquí, pero, sin embargo, si decide introducirla se realizará exactamente la misma petición que cuando la nota era 0, un simple *Update* en la base de datos para actualizar la nota anterior a la nueva. Por último, se informará al usuario de que se ha producido el cambio con éxito.

## Gestión de área cursos

Este formulario muestra todos los enlaces a videos, webs o documentos.

Al iniciar veremos dos botones, primer curso y segundo curso, que al pulsar sobre ellos de desplegará hacia abajo un conjunto de botones con cada una de las asignaturas correspondientes a ese curso, una vez pulsada una de ellas, se envía una petición al servidor mandándole el nombre de usuario, credenciales y asignatura que quiere visualizar, para que este le envíe una lista con los objetos enlace solicitados. Para obtener esta información tenemos la clase “UsuarioConectado.cs” que guarda información del usuario logueado como su nombre de usuario y sus credenciales.

Una vez en el servidor:

* Si tiene credenciales de Admin, hace una consulta a la Base de datos de todos los enlaces que tenga esa asignatura.
* Si no tiene credenciales de Admin, hace la misma consulta, pero con una condición más, que los enlaces no estén en estado “Caído”.

Si la consulta devuelve una o más filas, creamos un objeto Enlaces, de la clase enlaces que se encuentra en el proyecto Entidades Compartidas y los almacenamos en una lista que se envía al cliente.

Si la consulta no devuelve ninguna fila se envía una lista nula.

Cuando el cliente recibe la lista, comprueba que no sea nula en cuyo caso mostrará un mensaje en el ObjectListView indicando que no hay enlaces.

Si no es nula, se cargan las imágenes del link que el usuario introdujo como imagen, en el caso de algún error en la carga de dicha imagen, ponemos una por defecto creada por nosotros.

Cargadas las imágenes, entra en juego el *ObjectListView* que es el componente fundamental de este formulario.

El *ObjectListView* trabaja con objetos, si solo quisiéramos mostrar los atributos de los objetos que trae la lista recibida bastaría con pasarle la lista de objetos al *ObjectListView* y decirle desde diseño para cada columna que hemos creado previamente, que atributo queremos que nos muestre.

Nosotros hemos necesitado crear delegados para poder hacer comprobaciones de credenciales y en función de estas y de la información que trae cada objeto Enlace poder cambiar las imágenes que se muestran al usuario.

En este formulario se pueden realizar distintas funciones:

**Visualizar el enlace de tipo documento, web o video:** Al hacer doble clic sobre la fila que se quiere visualizar, se recoge el objeto que contiene esa fila, se lee su atributo “link” y abrimos el navegador con esa dirección.

**Voto positivo:** Al pulsar sobre este icono, obtenemos en objeto enlace que contiene la fila seleccionada y obtenemos su atributo “Id”. Enviamos al servido la petición “sumarValoración”, mandándole el “Id”, el nombre de usuario y sus credenciales. El servidor hace un *update* del enlace que corresponde con es “Id”, sumándole 1 a su columna valoración. Si se realiza correctamente envía la respuesta al cliente donde cambiamos la imagen de voto positivo y no permitiremos volver a valorar.

**Voto negativo:** realiza la misma acción que le voto positivo, con la diferencia de que resta 1 a la valoración.

**Reportar Link Caído:** Esta funcionalidad es para que el usuario nos pueda informar de que un link no funciona o no se corresponde con la información que muestra.Con pulsar sobre el icono “Círculo rojo con X”, muestra un mensaje de confirmación, si es afirmativo, obtenemos “Id” del objeto enlace que contiene la fila y enviamos una petición al servidor con los datos del usuario y el “Id” del enlace. El servidor hace un *update* en la BD del enlace que corresponde con ese “Id” cambiando la columna estado a “Revisión”. Si la actualización fue correcta, se envia un correo electronico a la cuenta del administrador de la aplicación informando que el usuario “x” a reportado el link “x” como caído para que este lo revise

**Fitrado en tiempo real:** Se realiza en el evento *TextChanged* del *TextBox* “Buscar”. *ObjectListView* dispone de un atributo llamado *ModelFilter* el cual igualamos a un nuevo *ModelFilter* que creamos con un delegado donde obtenemos el objeto de cada fila y de este su atributo “Titulo”. También obtenemos cada letra que el usuario introduce en el *TextBox* “Buscar”, que se envían a un método creado por nosotros que quita los acentos en caso de tenerlos. A continuación, se envían los dos *strings* a otro método también nuestro que comprueba si el “Titulo” contiene la palabra introducida por el usuario, en cuyo caso retorna verdadero y en este caso retornar el objeto para que lo muestre el *ObjectListView*.

## Gestión de subida de enlaces

Para este formulario se priorizó la sencillez y la facilidad por encima de funciones avanzadas o complicadas que pudieran despistar al usuario. Debido a ello este formulario divide en dos paneles, uno de introducción de datos y otro de ayuda.

El panel de introducción de datos se compone de cajas de texto para la introducción de datos como el nombre, la descripción el link del enlace o la imagen; dos grupos de *radioButtons* para tipo y curso, y para finalizar, dos *listBoxes* para la selección de asignatura y tema respectivamente.

Todos los *textBoxes* están validados del mismo modo que en el registro, apareciendo un icono y un *tooltip* para informar del estado de esa caja de texto.

En la última sección, donde se escoge el curso, los controles se irán activando en función el usuario vaya poniendo las opciones del enlace, es decir, una vez seleccione el curso se activará el *listBox* de asignaturas y una vez seleccione asignatura se activará el segundo *listBox* con la lista de temas de la asignatura seleccionada.

Una vez todos los campos están validados y se han seleccionado todas las características del enlace se activará el botón “Subir enlace” que permitirá al usuario subir su enlace a la aplicación.

Tras pulsar sobre el botón aparecerá un mensaje de confirmación, que dependerá de si el enlace se ha enviado correctamente a la base de datos y en caso positivo comprobará si el usuario tiene privilegios de “admin”. Esta última comprobación de privilegios cambiará el estado del enlace subido a “activo” si es por un usuario con privilegios de administrador o “en revisión” en caso contrario.

Con esta lógica se evita que un usuario con intenciones maliciosas pueda subir un enlace ilegal o comprometedor, todos los enlaces subidos por usuarios normales deben ser revisados por los administradores antes de poder ser activados.

Por último, tenemos el panel de ayuda, que simplemente muestra información e instrucciones al usuario para que éste no tenga ningún problema a la hora de subir sus propios enlaces.

Este panel se puede ocultar (y volver a mostrar) haciendo clic sobre el botón con el icono del ojo, pensado para aquellos usuarios que ejecuten la aplicación en miniportátiles, normalmente orientados a estudios y con una resolución menor a la habitual.

*En los proyectos Presentación y Negocio de la solución se encuentra el código comentado con los eventos (FormSubida) y la lógica (MetodosFormSubida).*

## Gestión de área ayuda

El siguiente formulario comparte ciertas similitudes con el anterior, pues nuevamente está dividido en dos partes, una de introducción de datos y otra de ayuda.

También tiene una característica peculiar, y es que solo es accesible para usuarios invitados o registrados sin privilegios de “admin”, pues su función principal como ya se explicó anteriormente en el apartado Diseño, es la de ofrecer funcionalidades de ayuda al usuario como recuperar o cambiar la contraseña, el curso, o reportar un bug o una sugerencia. Es por eso por lo que un usuario con privilegios de “admin” verá sustituido este panel, por el de Área administración, que se explicará en el siguiente apartado.

Otro de las características de este panel es que si se accede a él sin haberse conectado (*login*) solo estará accesible el apartado de recuperación de contraseña. Por el contrario, si el usuario está conectado, podrá acceder a todos los demás apartados menos el de recuperación de contraseña.

Para explicar la lógica vamos a dividir el panel en secciones:

**Cambiar contraseña:** Este apartado se compone de tres *textBoxes*, el primero para introducir la contraseña actual, y los dos siguientes para la nueva y su confirmación.

Si los tres campos se han rellenado correctamente se activará el botón “Cambiar contraseña”, si el usuario lo pulsa en ese momento se enviará la petición al servidor, que comprobará si la contraseña es correcta.

En caso negativo responderá al cliente informando, y este a su vez mostrará un mensaje al usuario indicando que la contraseña no es correcta.

En caso de ser correcta la contraseña el servidor se encargará de hacer un *UPDATE* sobre la base de datos con la nueva contraseña, si todo es correcto, se enviará la respuesta al cliente para que el usuario sea informado.

**Cambiar curso:** Aquí el usuario podrá elegir un curso y así poder introducir sus notas en el Área personal, o elegir ninguno y solo tener acceso a los enlaces propios en la esa área. A la hora de cambiar curso, se pensó inicialmente en el uso de disparadores para la creación de las notas a 0 en la base de datos, pero nos encontramos el problema de que concretamente el servicio de MySQL no ofrece la posibilidad de activar o desactivar los disparadores a voluntad mediante peticiones, con lo que hemos tenido que codificar un sistema algo mas rudimentario y manual a la hora de la creación de las notas. Cuando en el formulario se cambia el curso entre primero y segundo, se hace una peticion al servidor para saber que hacer con las notas que el usuario tiene guardadas, con lo que el usuario decidirá si quiere guardarlas para que posteriormente puedan volver a consultarse o bien el programa se deshaga de ellas. En este ultimo caso se realizará una petición al servidor de forma que haga un Delete sobre la tabla notas de la base de datos filtrando por usuario y curso. Si al curso que quiere cambiar el usuario es o bien primero o bien segundo, si ya hay notas existentes simplemente se cambiará el curso en la tabla de usuarios, sin embargo, si con anterioridad se ha decidido borrar las notas y se quiere volver al curso o bien si el usuario parte de curso 0 (no matriculado) y pasa a matricularse, en el mismo método de la petición, hará una llamada al metodo de creación manual de las notas, dependiendo del curso al que quiera moverse. Si todo ha ido bien, el programa nos avisará de ello, si no, todo seguirá igual que al principio.

**Reportar bug o sugerencia:** Se compone de dos *textBoxes*, uno para el título y otro para el cuerpo del mensaje, además están validados, y deben ser rellenados para que el botón de enviar se active.

Si se pulsa sobre el botón de enviar automáticamente el cliente enviará una petición al servidor con los datos introducidos. Una vez el servidor lo reciba, lo procesará y generará un email que se enviará a una cuenta que hemos habilitado para ello y que se explicará más a fondo en la sección de Envío de email.

**Contraseña olvidada:** En esta última sección nos encontramos con tres *textBoxes* más, de los cuales sólo el primero está habilitado. Aquí el usuario podrá introducir su correo electrónico (que será validada para comprobar que este bien formada), tras esto se activará el botón de “Enviar email”.

Si se pulsa este botón se enviará una petición al servidor que primero comprobará si el email existe en la base de datos, en caso negativo el servidor responderá indicando que el usuario no existe y se mostrará el mensaje en la interfaz.

En caso positivo se generará un token temporal que se enviará al email que el usuario haya introducido y se cambiará el *label* junto al *textBox* que ahora pedirá que se introduzca el token enviado.

Si el usuario introduce el token correctamente se habilitarán los dos textBoxes restantes para introducir la contraseña y confirmarla. Una vez haya rellenado estos dos campos correctamente se activará el botón de nuevo para poder cambiar la contraseña olvidada.

*En los proyectos Presentación y Negocio de la solución se encuentra el código comentado con los eventos (FormAyuda) y la lógica (MetodosFormAyuda).*

## Gestión de área administración

En este formulario se visualizan y administran todos los usuarios y enlaces que existen en la aplicación.

Esta área, solo será accesible si el usuario es el Admin o si tienes privilegios de Admin. La diferencia entre las funcionalidades de estos dos usuarios se explica en “Diseño de la Interfaz”, apartado “Pantalla Administración”. Para el resto de usuario será invisible.

El formulario consta de dos partes:

* **Usuarios:** Al pulsar sobre el botón “Usuarios”, se envía la petición de obtener todos los usuarioss al servidor, junto con el nombre del usuario y sus credenciales.

Cuando llega al servidor se trata de la siguiente manera:

* Si el nombre del usuario y su credencial es “Admin”, hacemos una consulta a la base de datos solicitando todos los usuarios excepto el suyo.
* Si el nombre del usuario es distinto de “Admin” y su credencial es “Admin”, hacemos la misma consulta con una condición más, que el nombre de usuario sea distinto de “Admin”, ya que los usuarios con privilegios no deben visualizar al usuario Administrador de la aplicación.

Si la consulta devuelve una o mas filas, creamos un objeto Usuario de la clase Usuario que se encuentra en el proyecto Entidades Compartidas, y a continuación lo almacenamos una lista para enviarla al cliente.

Si la consulta no devuelve ninguna fila se envía una lista nula.

Cuando el cliente recibe la lista del servidor comprueba que no sea nula en cuyo caso mostrará un mensaje en el ObjectListView indicando que no hay usuarios.

Si no es nula, comprobamos el nombre de usuario. Si es Admin, el ObjectListView muestra estos objetos cada uno en una fila mostrando el icono de papelera para el borrado. Si no es Admin, solo muestra este icono para los objetos usuario cuyo atributo “credenciales” sea “Normal”.

En esta pestaña se pueden realizar distintas funciones:

**Cambio de credenciales**: Al pulsar sobre las credenciales de un usuario, obtenemos el objeto Usuario y de este sus atributos “Id” y “credenciales” que contiene la fila seleccionada.

Si el atributo “credenciales” es Admin y el nombre del usuario conectado no es Admin, no se realiza ninguna acción.

Si el atributo “credenciales” es Admin y el nombre del usuario conectado es Admin o el atributo “credenciales” es Normal, se envía la petición de cambiar credenciales al servidor, junto con el nombre del usuario conectado, el atributo id y la nueva credencial que se le quiere asignar. El servidor hace un *update* sobre el usuario que corresponde con ese “Id”. Si se realiza correctamente envía la respuesta al cliente donde cambiamos el texto de la nueva credencial.

Para el usuario con credenciales Admin, antes de realizar esta acción se le muestra un mensaje de confirmación donde se informa que si le concede privilegios de Admin a un usuario no podrá revertir los cambios.

**Borrado de usuarios:** Al pulsar sobre el icono papelera se muestra un mensaje de confirmación de borrado. Si la respuesta es afirmativa, obtenemos el objeto Usuario y de este sus atributos “Id” que contiene la fila seleccionada para enviarla al servido con la petición de borrar usuario.

Cuando llega la petición al servidor se hace una transacción que consta de:

* Primero un *update* sobre la tabla enlaces donde todos los enlaces que haya subido el usuario con el “Id” que le hemos mandado, se cambian al usuario Admin.
* Segundo un *delete* en la tabla usuarios del usuario que corresponda a ese “Id” que borra todas las referencias que tenga en la base de datos ya que las tablas las claves foráneas (*foreign keys*) tienen la propiedad “borrado en cascada” (*on delete cascade*).

Si en alguno de estos procesos ocurriese un error, la transacción se anularía (*rollback*), si no, se comente (*commit*).

* **Enlaces:** Al pulsar sobre el botón “Enlaces”, se produce la misma acción ya explicada en el formulario Cursos cuando tienes privilegios de Admin, con la diferencia que se obtienen todos los enlaces de la base de datos en vez de una sola asignatura.

En esta pestaña se pueden realizar distintas funciones:

**Cambiar estado de un enlace:** Al pulsar sobre el icono de estado, obtenemos el objeto Enlace y de este sus atributos “Id” para enviarlo al servidor con la petición cambiar estado de enlace. En el servidor se hace un *update* del enlace que corresponde con ese “Id”. Los diferentes estados que puede tener son activo, revisión y caído.

Si todo es correcto envía la respuesta al cliente donde cambiamos la imagen en función del atributo estado del objeto enlace.

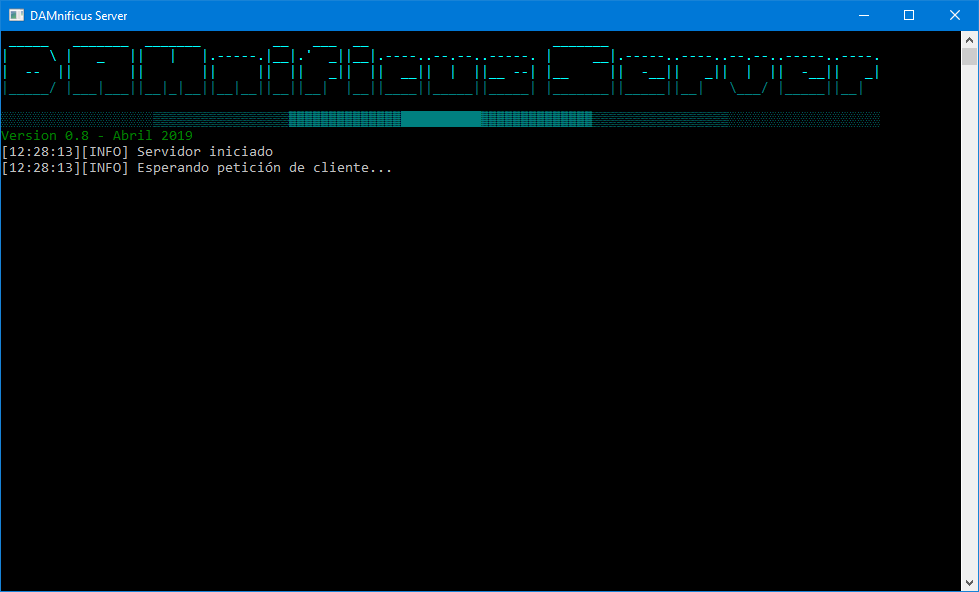
**Borrar un enlace:** Al pulsar sobre el icono papelera, muestra un mensaje de confirmación. Si es afirmativo, obtenemos el objeto Enlace y de este sus atributos “Id” para enviarlo al servidor con la petición borrar de enlace. En el servidor se hace un *delete* del enlace que corresponde con ese “Id”.

Si todo es correcto envía la respuesta al cliente donde actualizamos el ObjectListView y ya no mostrará este enlace.

# Parte servidor

A continuación, se describe parte relevante de la codificación del lado del servidor del proyecto. Para poder realizar esta fase, fue necesaria la instalación de XAMPP con PHPMyAdmin en nuestros equipos.

El objetivo de esta instalación era poder trabajar y realizar las pruebas en *localhost*, con el fin de poder implantar más adelante los ficheros y la base de datos en un servidor simulado en otro equipo.



## Programa servidor

Una parte muy importante del proyecto es el servidor, donde las peticiones son recogidas y tratadas para hacer el trabajo correspondiente en el servidor y ofrecer funcionalidad al cliente.

Para esta parte servidor hemos optado por una aplicación de consola para la plataforma Windows, escrita de nuevo en C#.

Esta aplicación es más simple y se compone de pocas clases, siendo la principal *Program.cs*, que es sobre la que se ejecuta el método *main* del servidor y la que se va a explicar en este apartado.

La conexión se establece a través de la clase HttpListener, la aplicación servidor y cliente tienen una archivo de configuración externo (app.config) donde se configura la dirección ip donde se va a hacer la escucha.

httpListener = new HttpListener

{

Prefixes = { "http://[ip de escucha]:8080/damnificus/" },

};

Una vez establecida la conexión mediante la llamada a httpListener.Start() el servidor quedará a la espera de peticiones, que llegarán por parte de los clientes.

En un principio, las primeras versiones del servidor trabajaban de forma interactiva, tratando las peticiones sobre el mismo hilo donde se ejecutaba el método *main*. Debido a la naturaleza de la aplicación este funcionamiento era poco óptimo si varios clientes estaban conectados a ella al mismo tiempo, tardando mucho éste en responder a las peticiones o permitiendo que éstas se perdieran.

Por esta razón se optó por dotar al servidor de tareas multihilo donde cada petición es ejecutada de forma independientemente en un nuevo hilo. Este hilo terminará cuando su petición sea respondida por el servidor, o sobrepase el tiempo límite de conexión.

La plataforma C# ofrece una solución potente a la vez que sencilla a este respecto, en el namespace *System.Threading.Tasks* se encuentra el método Task.Run() que permitirá ejecutar el método que trata las peticiones, que se explicará más adelante, en un pool de hilos.

Para más información sobre este método visite: <https://docs.microsoft.com/es-es/dotnet/api/system.threading.tasks.task.run?view=netframework-4.8>

Tras esta breve explicación sobre el funcionamiento interactivo del servidor, podemos continuar con el proceso de tratamiento de peticiones. Si un cliente envía una petición y es recibida por el servidor, éste abrirá un *stream* y leerá los datos serializados en JSON que contiene la dicha petición.

Tras deserializar el JSON recibido en un objeto de la clase Petición (Esta clase se encuentra en el proyecto de EntidadesCompartidas), este objeto llamado *peticionActual* es enviado como argumento al método **procesarPeticion**, que además como se ha explicado anteriormente, se ejecutará en un nuevo hilo:

ConsolaDebug.escribirEnConsola("INFO", "¡Conectado con un cliente!");

System.IO.Stream body = request.InputStream;

System.Text.Encoding encoding = request.ContentEncoding;

System.IO.StreamReader reader = new System.IO.StreamReader(body, encoding);

string datosJson = reader.ReadToEnd

var objetoJSON = JObject.Parse(datosJson);

// Serializa el objeto JSON en un objeto .NET

Peticion peticionActual = objetoJSON.ToObject<Peticion>();

// Procesa cada petición de forma asincrona en un threadpool

Task.Run(() => procesarPeticion(peticionActual, response));

El método procesar recibe además de la petición, un objeto response de la clase HttpListenerResponse, que se usará para enviar la respuesta al cliente desde dicho método.

public static void procesarPeticion(Peticion peticionActual, HttpListenerResponse response)

Una vez el método es llamado, lo primero que se encarga es de descifrar los datos del JSON recibido, tras lo cual se pasará directamente a un bloque de condicionales donde se evaluará la petición recibida y se tratará correctamente.



Una vez el bloque de condicionales encuentre la petición que se ha recibido la tratará. Aquí se pueden dar distintos casos en función de la petición, es decir, hay peticiones como la de *status* que sólo requieren una respuesta simple del servidor, para comprobar que la conexión funciona; pero también hay peticiones como la de login o de registro que requieren hacer operaciones con las bases de datos como lecturas o escrituras.

*En la clase Program.cs de la solución ServidorConexion se encuentra el código comentado con el tratamiento de cada una de sus peticiones comentadas.*

Un punto interesante que destacar en el tratamiento de las peticiones es que algunas de éstas requieren hacer inserciones o modificaciones sobre las BBDD, y se ejecutarán en distintos hilos, esto puede provocar condiciones de carrera entre hilos y que terminen en una corrupción de la base de datos, haciendo inservible la aplicación.

Para solucionar dicho problema se ha usado la instrucción lock, la cual adquiere el bloqueo de exclusión mutua de un objeto determinado, ejecuta un bloque de instrucciones y luego libera el bloqueo. Mientras se mantiene un bloqueo, el subproceso que lo mantiene puede volver a adquirir y liberar el bloqueo. Ningún otro subproceso puede adquirir el bloqueo y espera hasta que se libera.

Estos bloqueos solo se activan en los métodos de la clase conexión correspondiente, que en nuestro caso son dos (conexionUsuarios y conexionEnlaces).

## Envío de email

Durante el desarrollo pensamos que era importante implementar un sistema de envío de correos en la aplicación, así nos serviría para comunicarnos con el usuario y viceversa. Así mismo creamos un correo en la plataforma gratuita de Google Gmail:

[damnificusjovellanos@gmail.com](mailto:damnificusjovellanos@gmail.com)

Para la autenticación del usuario decidimos usar el clásico método de enviar un email al usuario con un token para poder ser posteriormente introducido en la aplicación y demostrar que éste tiene acceso a la cuenta que quiere registrar.

Un caso parecido ocurre cuando un usuario utiliza la función de Contraseña olvidada, que se explicó en el apartado de cliente.



Como se puede comprobar en la imagen el correo no es un simple texto plano, le hemos incluido una plantilla html y css para que tenga un diseño elegante y profesional, además de *responsive*. El email enviado se verá igual de bien tanto desde un móvil, una Tablet o un ordenador.

El último de los casos en que se usa esta clase es para el formulario de Ayuda, donde el usuario puede enviar un reporte o sugerencia al correo que hemos creado.

*En la clase EnviarEmail.cs de la solución ServidorConexion se encuentra el código comentado con la lógica y las plantillas html y css.*

## Generador de token

Esta es la clase más sencilla de la que se compone el servidor, y sirve únicamente a crear cadenas alfanuméricas que se utilizarán como claves de autentificación en distintos puntos de la aplicación.

Se ha procurado utilizar un algoritmo lo suficientemente complejo como para no obtener tokens repetidos con facilidad, usando nosotros una cadena de 64 caracteres para este propósito.

*En la clase GeneradorTokens.cs de la solución ServidorConexion se encuentra el método de generación.*

## Base de datos

Como se describe en el capítulo anterior sobre el Diseño, se crea una base de datos relacional con las tablas descritas en ese capítulo, teniendo en cuenta claves primarias y claves externas. Se ha llevado a cabo desde la aplicación phpMyAdmin y la implementación de los esquemas a la aplicación fue sencilla, pues se ha podido trasladar sin ningún tipo de transformación o problema alguno.

En el anterior apartado sobre el programa servidor se ha comentado que algunas de las operaciones sobre la base de datos estaban protegidas por el mecanismo lock.

Para mantener una buena proporción entre seguridad y rendimiento se ha usado esta instrucción sólo en las operaciones de escritura sobre las BBDD, dejando las consultas libres del mecanismo, para no ralentizar en demasía.

Además, se han protegido los accesos a la base de datos usando consultas parametrizadas y no concatenadas, evitando así inyecciones peligrosas de SQL.

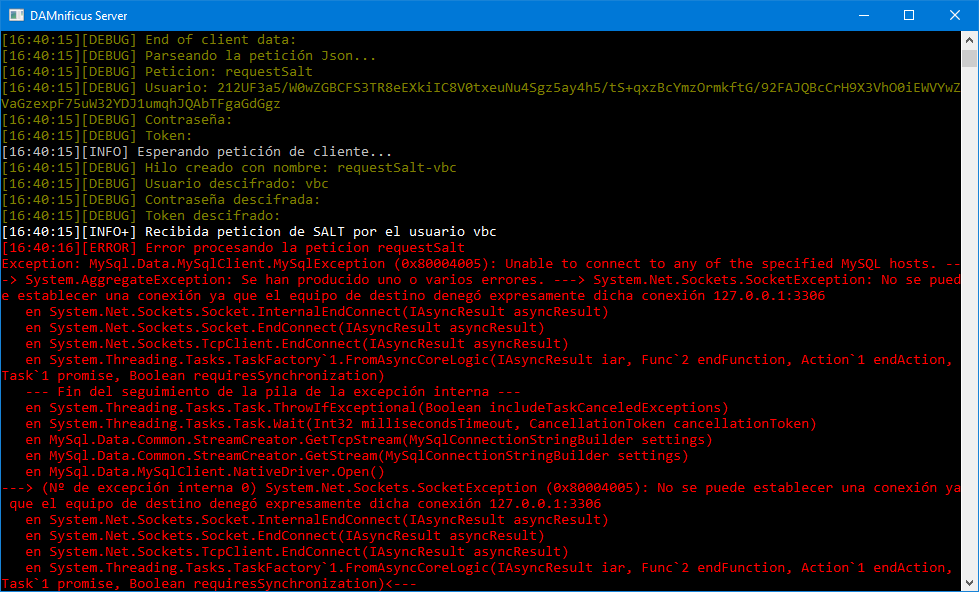
*En las clases conexionUsuarios.cs y conexionEnlaces.cs de la solución ServidorConexion se encuentra el código comentado con todas las operaciones sobre las dos bases de datos.*

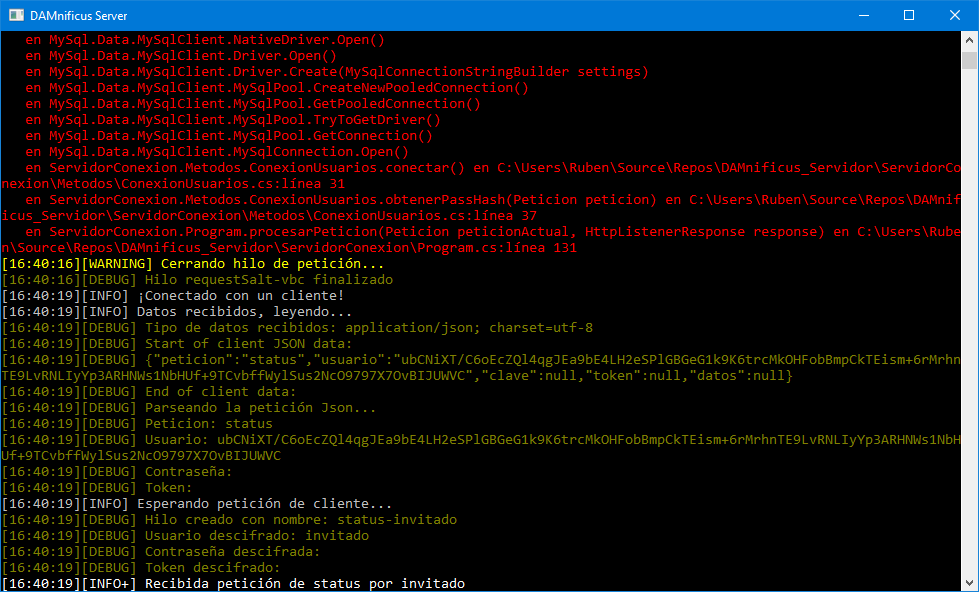
## Escritura en consola

Cuando decidimos que el servidor sería una aplicación de consola fue para tener un seguimiento claro y en directo de lo que estaba ocurriendo, mostrando trazas de lo que va sucediendo.

Esta idea la tomamos de algunas aplicaciones como Tomcat o Maven que utilizan sistemas de logs parecidos.

Para hacerlo aún más potente creamos una clase para escribir en la consola y poder personalizar la salida a la consola con distintos colores en función del mensaje o poder desactivar los mensajes de debug con sólo cambiar una variable.





Como se puede observar en las imágenes, en función del tipo del mensaje, éste se mostrará con una cabecera y un color diferente, permitiendo recolectar fácilmente información.

*En la clase ConsolaDebug.cs de la solución ServidorConexion se encuentra los métodos arriba comentados.*

# Archivos de configuarción

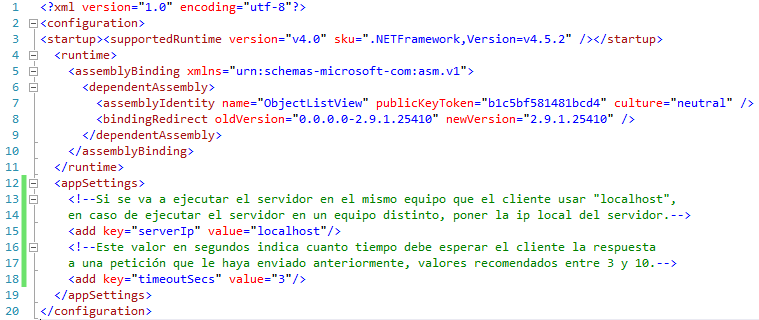
Tanto el cliente como el servidor tienen cada uno un archivo de configuración donde se pueden modificar algunos parámetros. Estos archivos están situados en la carpeta donde se encuentra el ejecutable y sus nombres son:

* ServidorConexion.exe.config
* App Damnificus.exe.config



Como se puede observar en la imagen, el archivo de configuración tiene una estructura XML y podemos encontrar los parámetros de configuración en el apartado *<appSettings>.*

* serverIp: Aquí se indica la ip (sin puerto) que el servidor va a utilizar para la escucha. Si el servidor es cargado en el mismo equipo que el cliente se deberá dejar el valor por defecto *localhost*, pero si el servidor se va a ejecutar en una máquina dedicada, se deberá poner la ip que le corresponda en la red. Por ejemplo “192.168.1.140”.
* debugConsole: Este valor indica si se han de mostrar las trazas de depuración en la consola del servidor, solo se recomienda activar esta opción si se está en un entorno de desarrollo. Para producción se recomienda desactivarla.



En el archivo de configuración del cliente podemos encontrar también el parámetro *serverIP*, que funciona igual que en el servidor, excepto que en este caso se debe poner la ip del equipo servidor y no la del cliente. Si los dos programas se ejecutan en el mismo equipo se puede dejar en *“localhost”.*

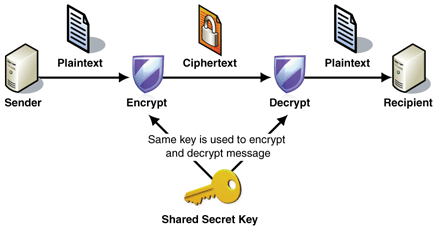
El segundo parámetro *timeoutSecs* indica el tiempo en segundos que el cliente debe esperar la respuesta del servidor antes de lanzar un mensaje de error de conexión al usuario, se recomienda poner valores entre 3 y 10.

# Seguridad y encriptación

El curso de Desarrollo de aplicaciones multiplataforma no integra muy extensamente esta disciplina, por ello decidimos no crear desde cero nosotros una encriptación, pues hubiera sido muy laborioso y costoso en tiempo y seguramente el resultado no fuera a ser mejor que otras opciones ya existentes.

Así pues, hicimos una ardua investigación donde nos documentamos sobre varias maneras de encriptar datos en C#, en un primer momento el cifrado solo se pensó para las contraseñas, así que se implementó un cifrado asimétrico SHA-2 que se utiliza en un gran número de herramientas de seguridad y protocolos. Algunos de ellos son TLS, SSL, PGP, SSH, S/MIME, IPsec y Bitcoin. Dentro del conjunto de funciones hash criptográficas que lo componen nosotros no decidimos por SHA-512 ya que es el más resistente. La potencia de esta encriptación es que una vez generada la clave ya no podrá conocerse la clave original.

Además, para cada Hash generamos una Salt diferente por lo que contraseña es totalmente diferente con lo que, si llegarán a descifrar una, no podrían descifrar todas.



Al poco tiempo de desarrollo caímos en el problema del envío, si el cifrado era simétrico el envío de datos se podía interceptar y usar las claves ya cifradas para obtener acceso ilegal a un usuario.

Así pues, decidimos que debíamos aplicar otra capa más de seguridad, usando un nuevo cifrado de 256 bits con un algoritmo PBKDF2, para cifrar los datos que se envíen a través de JSON entre el cliente, servidor y viceversa.

*En las clases Clave y CifradoJson del proyecto EntidadesCompartidas se encuentra el código comentado detallando el proceso.*

El código usado como base a este segundo cifrado se ha obtenido del siguiente enlace:

<https://stackoverflow.com/questions/10168240/encrypting-decrypting-a-string-in-c-sharp>

## Conclusiones

Este apartado se redacta a modo de resumen al final del capítulo de implementación con el fin de condensar los aspectos más importantes y algunas de las dificultades encontradas.

Se han desarrollado los módulos previstos en la fase de diseño, las interfaces y las tareas asíncronas que completan la funcionalidad de la aplicación.

Dado conocimiento adquirido durante el curso, tanto en las asignaturas de *Programación de servicios y procesos* como en *Programación* o *Desarrollo de interfaces*, no han surgido problemas que no hayan podido ser solucionados con una vuelta a los apuntes, una búsqueda en internet o consultando las librerías necesarias.

La elección de un lenguaje nuevo para nosotros como C# (aunque bastante parecido a Java) también nos impulsó a documentarnos y tratar de adquirir nuevos conocimientos para aplicarlos de forma eficiente en la aplicación, donde en todo momento hemos perseguido que tuviera un acabado lo más profesional posible.

Por otro lado, sí surgió alguna complicación en la conexión del cliente y servidor cuando ambos estaban ejecutándose en distintos equipos físicos de la red. Así como alguna tarea que duró un poco más de lo previsto, donde hubo que refrescar conocimientos sobre html/css y procedimientos almacenados adquiridos en las asignaturas de Bases de datos.

# PRUEBAS

Dado que el servidor está a la escucha de peticiones por parte de los clientes que se conecten a él y no pueden realizarse pruebas en modo Isolated, las pruebas que se han realizado comprueban el funcionamiento tanto de cliente como de servidor, con lo que ello conlleva de saber qué resultados tenemos y los que vamos a obtener al hacer las peticiones. Hay que destacar que cada prueba se ha realizado con varios equipos conectados simultáneamente para comprobar el correcto funcionamiento de todos y cada uno de los hilos de ejecución.

Tras terminar la codificación de cada prototipo se ha realizado una serie de pruebas con el objetivo de intentar detectar posibles errores.

En todos los prototipos, el procedimiento de pruebas ha sido el que sigue.

En primer lugar, se han realizado pruebas de caja negra, escogiendo funciones que podían dar problemas debido a su complejidad, o a su conexión con el servidor. Se han probado paso a paso intentando dar solución a los problemas encontrados, con el objetivo de evitar errores futuros mayores.

En segundo lugar, se realizan pruebas de caja blanca, con conocimiento del código, y examinando las partes o módulos aparentemente más problemáticos, que normalmente coincidían con los módulos marcados como erróneos en las pruebas de caja negra.

Por último, y una vez solucionados los errores de cada prototipo, se llevan a cabo pruebas con usuarios finales.

A continuación, se enumeran las diferentes pruebas en cada uno de los prototipos.

* **Prototipo 1:** Se prueba el registro de usuario fallido y satisfactorio y el inicio de sesión desde la aplicación. Así mismo los distintos formularios con respuesta prevista.
* **Prototipo 2:** Pruebas sobre cambios en panel de ayuda / administración.
* **Prototipo 3:** Se prueban subidas de enlaces y registros de notas.

En todas las fases de pruebas se va comprobando la correcta adición de los datos a la base de datos del proyecto.

En la siguiente tabla se observa el listado de pruebas realizadas. La mayoría de las pruebas que se intentaron fueron operaciones no permitidas para comprobar que el sistema lo detecta y toma medidas al respecto.

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Salida esperada** | **Prototipo** |
| Registro | Registro de usuario en la BD | 1 |
| Registrarse sin indicar algún campo | No permite el registro | 1 |
| Registrarse con un correo electrónico que no tiene formato Email | No permite el registro | 1 |
| Intento de registro de un usuario ya existente | No permite el registro | 1 |
| Dar de baja un usuario siendo admin | Se da de baja el usuario y se elimina de la BD | 1 |
| Inicio de sesión | Permite el inicio de sesión en la aplicación | 1 |
| Inicio de sesión sin indicar algún campo | No permite el inicio de sesión | 1 |
| Inicio de sesión con usuario y contraseña que no coinciden | No permite el inicio de sesión | 1 |
| Se intenta editar el usuario con algún dato vacío | No edita los datos del usuario | 2 |
| Se editan usuarios y enlaces siguiendo los criterios | Edición correcta | 2 |
| Se accede al panel de área personal, carga modelo | Carga correcta del formulario | 2 |
| Se intenta subir un enlace sin especificar algún campo | Ni siquiera hay opción | 3 |
| Se sube un enlace detallando cada campo correctamente | El enlace se sube correctamente | 3 |
| Se intenta agregar nota sin hacer un login o sin tener curso asignado | No se puede y el cliente avisa de ello | 3 |
| Se agrega nota con usuario identificado | La nota cambia | 3 |
| Se intentan conectar dos usuarios con la misma cuenta | El servidor rechaza la segunda conexión | 3 |
| Se intenta acceder a cursos sin conexión al servidor | No se muestran cursos, ni la aplicación da ningún error | 3 |
| Se sube un enlace con la misma dirección a uno existente | La aplicación lanza un mensaje con el error | 3 |
| Se carga servidor sin XAMPP iniciado | La aplicación servidor avisa del problema | 3 |
| Se intenta usar una contraseña de menos de 8 caracteres | La interfaz impide seguir hasta introducir una contraseña ok | 3 |
| Ocurre excepción irreversible en servidor | El servidor comunica el error y se reinicia automáticamente | 3 |

# MANUAL DE USO

Al tratarse de una aplicación medianamente compleja y dividida, y que las diferentes partes (cliente-servidor-bbdd) que la componen deben configurarse adecuadamente antes de arrancarlo todo por primera vez.

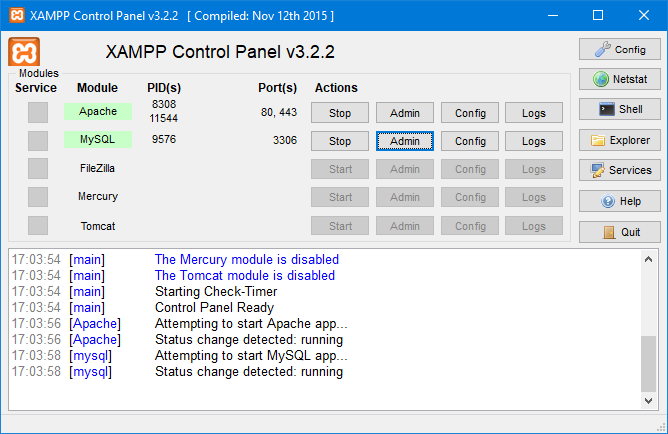
Lo primero que se necesita es instalar XAMPP, que se puede encontrar en la carpeta *Base de datos* dentro del pendrive facilitado al profesorado o descargar del siguiente enlace a su página oficial:

<https://www.apachefriends.org/es/index.html>

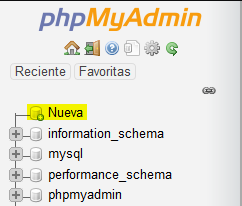
Se recomienda instalar todos los componentes que están seleccionados por defecto, una vez finalizada la instalación, se deberá abrir el programa XAMPP y activar los módulos Apache y MySQL.

Tras esto hay que pulsar sobre el botón Admin que está en la fila del módulo MySQL. Accederemos a una web llamada phpMyAdmin donde se pueden crear y gestionar bases de datos MySQL.

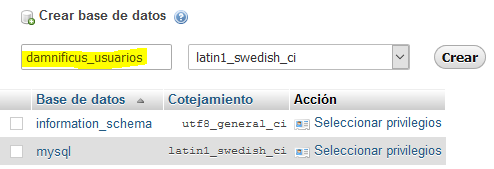
En la siguiente imagen se muestra marcado el botón:



Una vez estemos en la web de administración de MySQL podremos observar que a la izquierda tenemos un panel con las distintas bases de datos que se instalan por defecto.



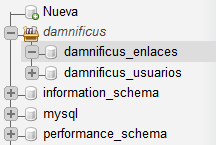
Debemos pulsar sobre la pestaña “Nueva” tal y como se ve en la imagen, y aparecerá un cuadro para crear una nueva base de datos:



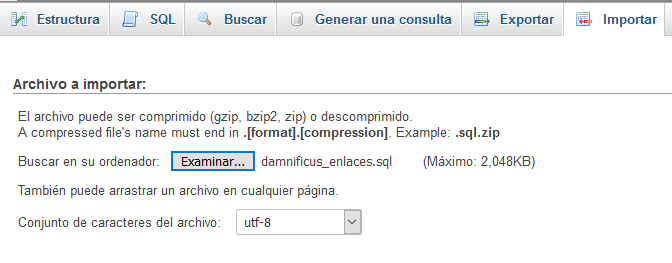
En este cuadro deberemos crear las dos bases de datos que utiliza el servidor para funcionar, llamándolas a cada una como se indica a continuación:

* damnificus\_usuarios
* damnificus\_enlaces

Una vez creadas, deberá aparecernos en el menú de la izquierda los siguientes elementos tal y como se observa en la imagen:



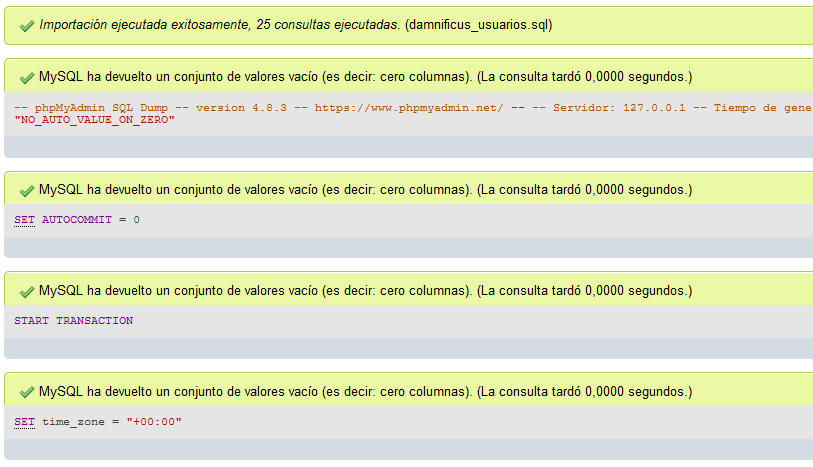
Tras esto ya podemos importar los scripts de carga, para ello seleccionamos primero una de las bases de datos, por ejemplo, damnificus\_enlaces, y pulsamos en la pestaña Importar del menú superior:



Una vez en esta ventana deberemos pulsar sobre Examinar y buscar el documento:

damnificus\_enlaces.sql

Con el script seleccionado solo tenemos que pulsar aceptar para que se ejecuten las sentencias necesarias en el servidor para cargar los datos correctamente.



Una vez hayamos visto que se han ejecutado todas las sentencias correctamente debemos repetir estos pasos en la otra base de datos, damnificus\_usuarios, cargando su script sql y ejecutándolo del mismo modo.

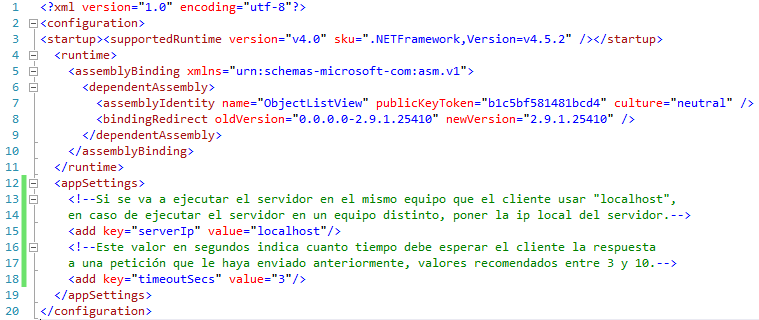
Una vez hecho esto ya estarían preparadas las dos bases de datos necesarias para el buen funcionamiento de la aplicación.

El siguiente paso es configurar la aplicación servidor, para ello debemos abrir con un editor de texto el archivo *ServidorConexion.exe.config* que se encuentra en la misma carpeta que el ejecutable del mismo nombre.

En él encontraremos los distintos parámetros que podemos modificar como ya se ha explicado en la implementación del servidor.



Tras configurar el servidor debemos hacer lo mismo con el cliente, en este caso el archivo se llama *App Damnificus.exe.config* y se encuentra en la carpeta donde está el ejecutable del cliente.

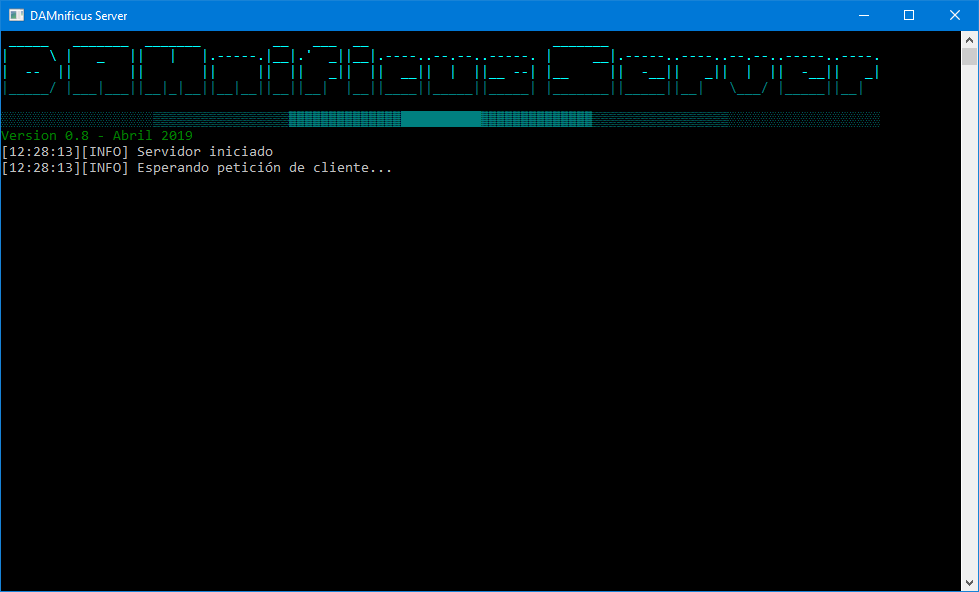


En el caso del parámetro serverIp deberemos poner el mismo valor que hayamos indicado en el archivo de configuración del servidor.

El segundo parámetro lo podemos dejar como viene por defecto.

Ya con todos estos pasos ejecutados correctamente podemos ejecutar en primera instancia el servidor.

Es importante remarcar que para que el servidor funcione en una maquina dedicada (en red) se ha de ejecutar éste con privilegios de administrador, sino no permitirá la escucha en la red. Esto no es necesario si los dos programas corren en la misma máquina.



Una vez veamos que el servidor está funcionando y esperando una petición ya podremos ejecutar la aplicación cliente que, si todo se ha configurado correctamente, conectará automáticamente con el servidor haciendo una petición de status a éste.

Con esto finaliza el manual de uso y de instalación de los componentes necesarios.

# CONCLUSIONES

Este capítulo se dedica a las conclusiones finales y personales del proyecto, extraídas a lo largo de todo el proceso de desarrollo del mismo.

Se han cumplido todos los objetivos y requisitos mencionados en los capítulos. El principal de ellos era crear una aplicación móvil para Android capaz de ofrecer los servicios de la empresa a usuarios por medio de este canal.

La aplicación desarrollada ofrece la búsqueda y posible subida de enlaces, además permite que el usuario pueda gestionar notas y sus enlaces en su Área personal, así como el contacto con los administradores correo electrónico.

Atendiendo al desarrollo técnico del proyecto, fue difícil sintetizar todas las ideas y requisitos que había sobre la mesa. Una vez realizadas las fases de análisis y diseño, las fases posteriores fueron encauzadas rápidamente. Por lo que las primeras fases de planificación, análisis y diseño resultaron ser de suma importancia.

Además de haber aprendido a desarrollar un proyecto de principio a fin, pasando por todas sus fases, hemos adquirido nuevos conocimientos, o ampliado muchos de ellos sobre la plataforma .NET y C#, el modelo a capas, la asincronicidad y el uso de varios hilos entre otros conocimientos necesarios que se han requerido durante todas las fases.

# RECURSOS MATERIALES

Por parte de Rubén Balsera, se han usado dos equipos para el desarrollo, un ordenador de sobremesa y un portátil, además de un teléfono móvil y una Tablet para comprobar que los emails se veían correctamente en distintos dispositivos.

Por parte de Valentín Sánchez, otros cuatro equipos para desarrollo, dos sobremesa y dos portátiles, dos de ellos para desarrollo, y el resto ya en fases finales, para pruebas de comunicación entre cliente y servidor de la aplicación desde las distintas máquinas.

Por parte de Antonio Illarramendi, se han utilizado dos equipos, uno para el desarrollo y los dos para realizar pruebas.

Finalmente, no olvidarnos de las cantidades de papel y bolígrafos o rotuladores que se han usado, sobre todo en la parte de diseño, donde se dibujaron bocetos tanto de las bases de datos, como de las distintas interfaces que componen la aplicación. Aunque la parte de pruebas y arreglo de errores también se llevó otra gran cantidad de material.

# BIBLIOGRAFÍA

Programación por capas  
<https://es.wikipedia.org/wiki/Programaci%C3%B3n_por_capas>

Modelo de prototipos  
<http://es.wikipedia.org/wiki/Modelo_de_prototipos>

Man In The Middle, ataques, vulnerabilidades y como prevenirlas

<http://www.veracode.com/security/man-in-the-middle-attack>

Red de Innovacion de Seguridad Nacional  
<https://www.nsin.us/>

Serialización JSON con .NET  
<http://www.devjoker.com/contenidos/catss/505/JSONNET-Mejorando-la-serializacion-JSON-con-NET.aspx>

Tasks  
<https://docs.microsoft.com/es-es/dotnet/api/system.threading.tasks.task.run?view=netframework-4.8>

Locks  
<https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/keywords/lock-statement>

LiveCharts para Windows Forms  
<https://lvcharts.net/App/examples/v1/wf/start>

ObjectListView

<http://objectlistview.sourceforge.net/cs/index.html>