



JEE DESPLIEGUE PARTE 2

Integración MySQL

- Hay que crear la base de datos tanto en el equipo de desarrollo como en el servidor de despliegue.
- También es necesario integrarla en el entorno de desarrollo.
- No es obligatorio usar el mismo sistema, de hecho es habitual usar un servidor ligero en desarrollo (MariaDB por ejemplo) y un servidor empresarial en producción (Oracle por ejemplo). El despliegue también incluye la tarea de configurar y sincronizar adecuadamente ambos sistemas.
- En nuestro caso trabajaremos con dos sistemas MySQL, e intentaremos automatizarlo lo más posible.

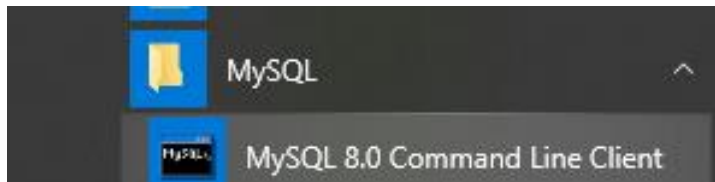
MySQL CREACIÓN BASE DATOS

- Creamos un fichero llamado biblioteca.sql con los comandos necesarios.

```
SET GLOBAL time_zone = '+3:00';  
create database biblioteca;  
create user bibliotecario identified by  
'bibliotecario';  
grant all privileges on biblioteca.* to  
'bibliotecario';  
use biblioteca;  
create table libros (isbn INT(13) primary key,  
titulo varchar(30) not null, autor varchar(30)  
not null);
```

MySQL CREACIÓN BASE DATOS

- ▶ Para ejecutar el script hay dos opciones:
 - ▶ Desde línea de comandos
`mysql -u root -p < biblioteca.sql`
 - ▶ Desde dentro de mysql
`mysql> source biblioteca.sql`
- ▶ En el equipo de desarrollo usaremos la segunda opción:



```
mysql> source C:\Users\eltxa\OneDrive\Documentos\eclipse-workspace\Biblioteca\scripts\biblioteca.sql
Query OK, 1 row affected (0.01 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Database changed
Query OK, 0 rows affected, 1 warning (0.05 sec)
```

MySQL CREACIÓN BASE DATOS

- ▶ En el servidor de despliegue usaremos otro método (HAY MUCHISIMAS OPCIONES).
- ▶ Creamos un archivo llamado biblioteca_remote.sh con el siguiente contenido:

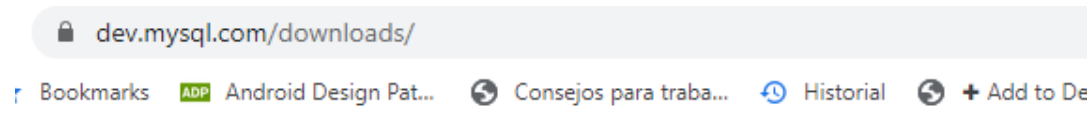
```
echo "profesor" | sudo -S mysql -u root -pprofesor -e "SET GLOBAL  
time_zone = '+3:00';"  
echo "profesor" | sudo -S mysql -u root -pprofesor -e "create database  
biblioteca;"  
echo "profesor" | sudo -S mysql -u root -pprofesor -e "create user  
bibliotecario identified by 'bibliotecario';"  
echo "profesor" | sudo -S mysql -u root -pprofesor -e "grant all  
privileges on biblioteca.* to 'bibliotecario';"  
echo "profesor" | sudo -S mysql -u root -pprofesor -D biblioteca -e  
"create table libros (isbn INT(13) primary key, titulo varchar(30) not  
null, autor varchar(30) not null);"
```

Lo ejecutamos con Putty:

```
putty.exe -ssh usuario@IP -pw "password" -m  
"biblioteca_remote.sh"
```

MySQL INTEGRACIÓN CON ECLIPSE

- Para que la aplicación pueda comunicarse con MySQL hay que incluir el driver JDBC. Se descarga de la página oficial de MySQL.



SQL Community Downloads

MySQL Yum Repository

MySQL APT Repository

MySQL SUSE Repository

MySQL Community Server

MySQL Cluster

- C API (libmysqlclient)
- Connector/C++
- Connector/J
- Connector/NET
- Connector/Node.js
- Connector/ODBC

MySQL INTEGRACIÓN CON ECLIPSE

[General Availability \(GA\) Releases](#) [Archives](#) 

Connector/J 8.0.22

Select Operating System:

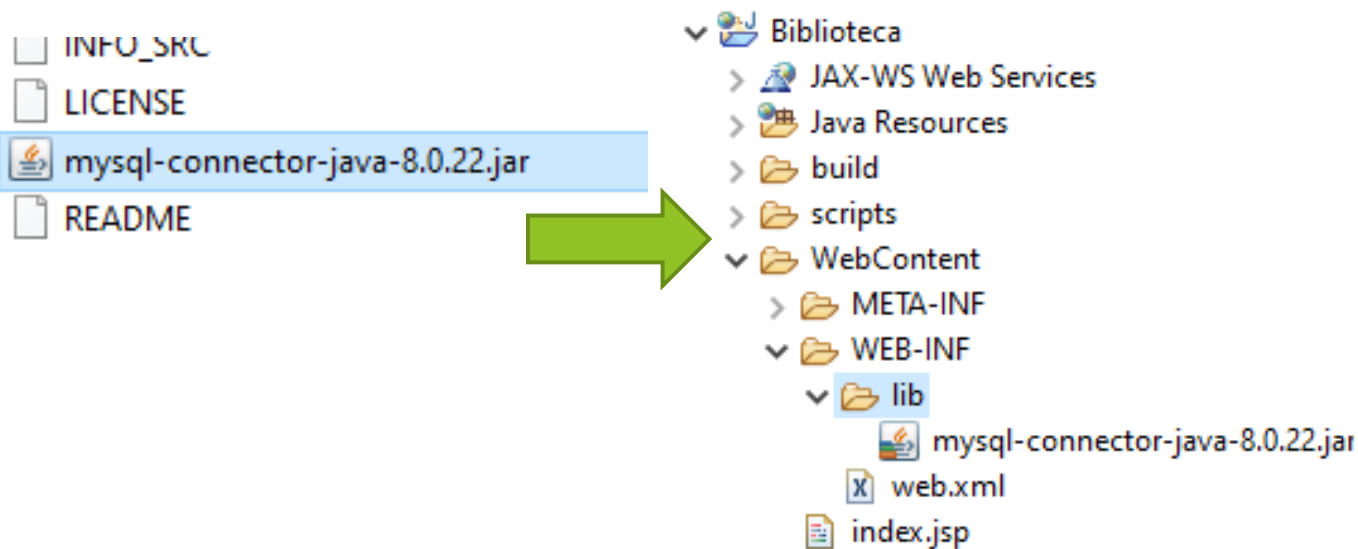
Platform Independent

Looking for previous GA versions?

Platform Independent (Architecture Independent), Compressed TAR Archive (mysql-connector-java-8.0.22.tar.gz)	8.0.22	3.8M	Download
MD5: eb4e915366543844a80a06ea111ec6f7 Signature			
Platform Independent (Architecture Independent), ZIP Archive (mysql-connector-java-8.0.22.zip)	8.0.22	4.5M	Download
MD5: 2aaf6a62a2f3330730ec77b1c025646f Signature			

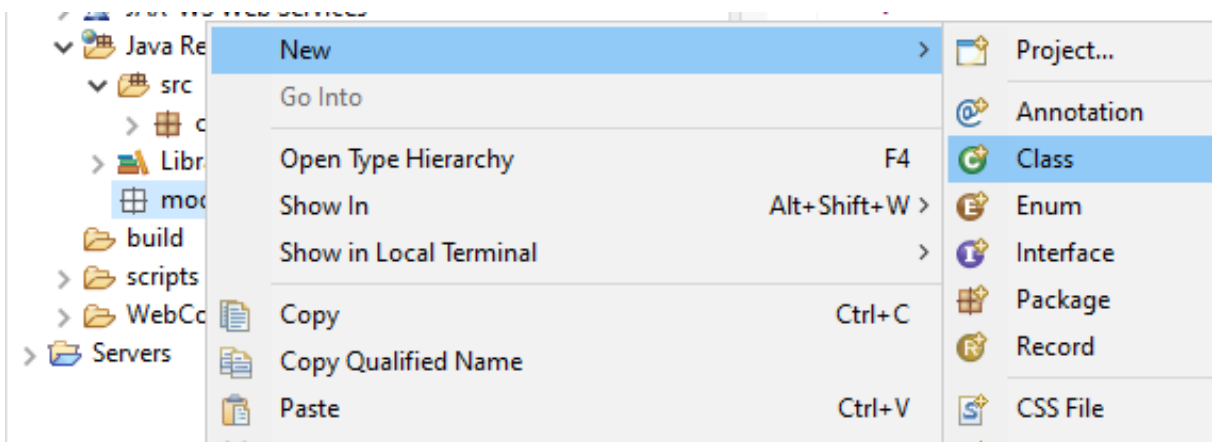
MySQL INTEGRACIÓN CON ECLIPSE

- Descomprimos el archivo descargado y copiamos el archivo mysql-connector-java-XXX.jar a la carpeta WebContent/WEB-INF/lib del proyecto.



DAO Libros / JDBC

- ▶ Vamos a programar nuestra aplicación para que pueda trabajar con datos de la tabla libros usando JDBC.
- ▶ Creamos en src un nuevo package llamado modelo.
- ▶ Dentro de modelo creamos una clase Libro que pueda contener los datos de un libro.



DAO Libros / JDBC

- Añadimos a la clase Libro un constructor con parámetros, getters y setters y un método ToString() mediante la opción source (botón derecho sobre la clase).

```
package modelo;

public class Libro{

    private int isbn;
    private String titulo;
    private String autor;

    public Libro(int isbn, String titulo, String autor) {
        super();
        this.isbn = isbn;
        this.titulo = titulo;
        this.autor = autor;
    }

    public int getIsbn() {
        return isbn;
    }

    public void setIsbn(int isbn) {
        this.isbn = isbn;
    }
}
```

DAO Libros / JDBC

- ▶ Ahora crearemos dentro del package modelo una clase llamada LibroDAO que se encargará de acceder a la base de datos usando JDBC y trabajar con los libros (CRUD).
- ▶ Como miembros incluiremos los datos necesarios para la conexión así como objetos que usaremos para realizar operaciones sobre la base de datos.
- ▶ En el constructor crearemos una conexión capturando los posibles errores.
- ▶ Todos los errores se capturan y se lanzan como RuntimeException, de ese modo fluyen hacia arriba en la aplicación hasta la interfaz de usuario.

DAO Libros / JDBC

```
public class LibroDAO {

    private String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
    private String DB_URL = "jdbc:mysql://localhost/biblioteca";
    private String DB_USER = "bibliotecario";
    private String DB_PASS = "bibliotecario";
    private Connection conn = null;
    private Statement stm = null;
    PreparedStatement ps = null;
    private ResultSet rs = null;

    public LibroDAO() throws RuntimeException {
        super();
        // TODO Auto-generated constructor stub
        try {
            Class.forName(JDBC_DRIVER);
            conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASS);
        } catch (ClassNotFoundException e) {
            throw new RuntimeException("ERROR: failed to load MySQL JDBC driver.",e);
        } catch (SQLException e) {
            throw new RuntimeException("ERROR: failed to access database.",e);
        }
    }
}
```

DAO Libros / JDBC

- El método finalize se ocupa de cerrar todos los objetos usados. No hacer esto es poco eficiente ya que podemos dejar abiertas muchas conexiones a la base de datos.

```
@Override
protected void finalize() throws Throwable {
    // TODO Auto-generated method stub
    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException e) {
            throw new RuntimeException("Error cerrando la conexión",e);
        }
    }
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException e) {
            throw new RuntimeException("Error cerrando el resultset",e);
        }
    }
}
```

DAO Libros / JDBC

- Creamos también dos métodos para añadir un libro a la base de datos y para devolver todos los libros.

```
public boolean insertar(Libro libro) throws RuntimeException {  
    try {  
        ps = conn.prepareStatement("insert into libros values(?,?,?)");  
        ps.setInt(1, libro.getIsbn());  
        ps.setString(2, libro.getTitulo());  
        ps.setString(3, libro.getAutor());  
        int i = ps.executeUpdate();  
  
        if (i==1) {  
            return true;  
        }  
    } catch (SQLException e) {  
        throw new RuntimeException("ERROR: failed to insert libro",e);  
    }  
    return false;  
}
```

DAO Libros / JDBC

```
public ArrayList<Libro> getLibros() throws RuntimeException {
    try {
        stm = conn.createStatement();
        String consulta = "select * from libros";
        rs = stm.executeQuery(consulta);
        ArrayList<Libro> libros = new ArrayList<Libro>();
        while (rs.next()) {
            Libro libro = new Libro(rs.getInt("isbn"),rs.getString("titulo"),
                                    rs.getString("autor"));
            libros.add(libro);
        }

        return libros;
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        throw new RuntimeException("failed to create statement",e);
    }
}
```

AÑADIR LIBRO

- ▶ Vamos a programar la inserción de libros en la aplicación.
- ▶ Añadimos un formulario en la página index.jsp para añadir un nuevo libro.

```
<div>
  <form action="insertar" method="post">
    <label for="isbn">ISBN</label>
    <input type="text" id="isbn" name="isbn" placeholder="ISBN..">

    <label for="titulo">Título</label>
    <input type="text" id="titulo" name="titulo" placeholder="Titulo..">

    <label for="autor">Autor</label>
    <input type="text" id="autor" name="autor" placeholder="Autor..">

    <input type="submit" value="Submit">
  </form>
</div>
```


AÑADIR LIBRO

- Creamos también un archivo CSS para aplicar estilos. Se llamara styles.css y estará en una nueva carpeta llamada css dentro de WebContent.
- Indicamos en index.jsp que use styles.css



```
<title>Biblioteca</title>
<link rel="stylesheet" href="css/styles.css">
</head>
```

```
input[type=text], select {
    width: 200px;
    padding: 12px 20px;
    margin: 8px 0;
    display: inline-block;
    border: 1px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
}

input[type=submit] {
    width: 200px;
    background-color: #4CAF50;
    color: white;
    padding: 14px 20px;
    margin: 8px 0;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}

input[type=submit]:hover {
    background-color: #45a049;
}

div {
    width: 250px;
    border-radius: 5px;
    background-color: #f2f2f2;
    padding: 20px;
}
```

AÑADIR LIBRO

← → 📄 💰 ▼ <http://localhost:8080/Biblioteca/>

ISBN

Título

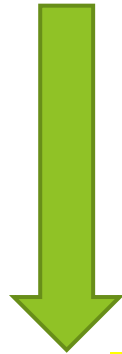
Autor

Submit

AÑADIR LIBRO

- Ahora vamos a modificar nuestro servlet para que procese las peticiones de inserción.

```
<div>  
  <form action="insertar" method="post">
```



El servlet se encarga de procesar los dos tipos de peticiones que tenemos hasta ahora

```
@WebServlet(urlPatterns = {"", "/insertar"})  
public class BibliotecaController extends HttpServlet {  
  private static final long serialVersionUID = 1L;
```

AÑADIR LIBRO

- El método doGet quedará así. Crea un objeto Libro con los datos recibidos del formulario y lo añade a la base de datos mediante LibroDAO

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    RequestDispatcher despachador = null;
    if (request.getServletPath().equals("")) {
        despachador = request.getServletContext().getRequestDispatcher("/index.jsp");
    } else if (request.getServletPath().equals("/insertar")) {
        try {
            LibroDAO libroDAO = new LibroDAO();
            Libro libro = new Libro(Integer.parseInt(request.getParameter("isbn")),
                                    request.getParameter("titulo"), request.getParameter("autor"));
            libroDAO.insertar(libro);
        } catch (NumberFormatException e) {
            // TODO Auto-generated catch block
            request.setAttribute("error", e.getMessage());
        } catch (RuntimeException e) {
            // TODO Auto-generated catch block
            request.setAttribute("error", e.getMessage());
        }
        despachador = request.getServletContext().getRequestDispatcher("/");
    }
    despachador.forward(request, response);
}
```


AÑADIR LIBRO

- Hacemos una prueba.

ISBN

Título

Autor



```
mysql> select * from libros;
+-----+-----+-----+
| isbn | titulo   | autor  |
+-----+-----+-----+
| 1    | El Quijote | Cervantes |
+-----+-----+-----+
1 row in set (0.00 sec)
```

LISTA DE LIBROS

- ▶ Ahora vamos a incluir en la página una tabla con los libros que existen en la base de datos.
- ▶ En el Servlet obtenemos los libros mediante el método `getLibros()` de `LibroDAO` y se los pasamos a `index.jsp` mediante un atributo en la request.

```
if (request.getServletPath().equals("")) {
    try {
        LibroDAO libroDAO = new LibroDAO();
        ArrayList<Libro> libros;
        libros = libroDAO.getLibros();
        request.setAttribute("libros", libros);
    } catch (RuntimeException e) {
        // TODO Auto-generated catch block
        request.setAttribute("error", e.getMessage());
    }
    despachador = request.getServletContext().getRequestDispatcher("/index.jsp");
} else if (request.getServletPath().equals("/insertar")) {
```

LISTA DE LIBROS

- Y en index.jsp obtenemos el atributo con la lista de libros y generamos una tabla con los datos.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ page import="java.util.ArrayList" %>
<%@ page import="modelo.Libro" %>

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Biblioteca</title>
<link rel="stylesheet" href="css/styles.css">
</head>
<body>
<table>
<tr>
<th><b>ISBN</b></th>
<th><b>Titulo</b></th>
<th><b>Autor</b></th>
</tr>

<% ArrayList<Libro> libros=(ArrayList<Libro>)request.getAttribute("libros");
    if (libros != null){
        for(Libro l:libros){%>
            <tr>
                <td><%=l.getIsbn()%></td>
                <td><%=l.getTitulo()%></td>
                <td><%=l.getAutor()%></td>
            </tr>

            <%}
        }%>
    </table>
```

LISTA DE LIBROS

ISBN	Título	Autor
1	El Quijote	Cervantes
2	La Regenta	Clarín

ISBN

Título

Autor

Submit

GESTIÓN DE ERRORES

- En la aplicación no se están mostrando los errores que se producen. Se capturan en el controlador y se pasan como atributo de la request a index.jsp.

```
request.setAttribute( libros , libros);
} catch (RuntimeException e) {
    // TODO Auto-generated catch block
    request.setAttribute("error",e.getMessage());
}
despachador = request.getServletContext().getRequestDispatcher("/insertar");
if (request.getServletPath().equals("/insertar")) {
    try {
        LibroDAO libroDAO = new LibroDAO();
        Libro libro = new Libro(Integer.parseInt(request.getParameter("titulo")),request.getParameter("autor"));
        libroDAO.insertar(libro);
    } catch (NumberFormatException e) {
        // TODO Auto-generated catch block
        request.setAttribute("error",e.getMessage());
    } catch (RuntimeException e) {
        // TODO Auto-generated catch block
        request.setAttribute("error",e.getMessage());
    }
}
```

- Para mostrarlos añadimos el siguiente código en index.jsp

```
<% String error = (String)request.getAttribute("error");
    if (error!=null){ %>
        <p><%=error%></p>
    } %>
```

GESTIÓN DE ERRORES

ISBN	Título	Autor
1	El Quijote	Cervantes
2	La Regenta	Clarín

ISBN

1

Título

La Divina Comedia

Autor

Dante

×

Submit



ERROR: failed to insert libro

ISBN	Título
1	El Quijote
2	La Regenta

GESTIÓN DE MENSAJES

- ▶ Generalmente las aplicaciones necesitan mostrar mensajes a los usuarios, aparte de los errores. Por ejemplo, al realizar una operación en la base de datos.
- ▶ Vamos a añadir un sistema de mensajes sencillo, similar a la gestión de errores. Desde el controlador pasaremos los mensajes a las páginas jsp mediante atributos de la request.

```
Libro libro = new Libro(Integer.parseInt(request.getParameter("isbn")),  
    request.getParameter("titulo"), request.getParameter("autor"));  
libroDAO.insertar(libro);  
request.setAttribute("info", "Libro " + libro + " añadido");
```

- ▶ Para mostrarlos añadimos el siguiente código en index.jsp.

```
<% String info = (String)request.getAttribute("info");  
    if (info!=null){ %>  
        <p><%=info%></p>  
    <%}%>
```

GESTIÓN DE MENSAJES

ISBN

Título

Autor ×



Libro Libro [isbn=3, titulo=La Divina Comedia, autor=Dante] añadido

ISBN	Título	Autor
1	El Quijote	Cervantes
2	La Regenta	Clarín
3	La Divina Comedia	Dante

DESPLIEGUE

- ▶ Hacemos commit llamado “JDBC” y lo subimos a GitHub
- ▶ Ejecutamos el script de despliegue.

putty.exe -ssh usuario@IP -pw "password" -m "deploy.sh"

The screenshot shows a web browser window with the address bar displaying "192.168.17.154:8080/Biblioteca/". The page features a header with navigation links: "Aplicaciones", "Bookmarks", "ADP Android Design Pat...", and "Consejos para traba...". Below the header is a table with three columns: "ISBN", "Título", and "Autor". The table is currently empty. Below the table is a form with three input fields labeled "ISBN", "Título", and "Autor", each containing a placeholder text "ISBN..", "Titulo..", and "Autor.." respectively. A green "Submit" button is located at the bottom of the form.

ISBN	Título	Autor
------	--------	-------

ISBN ISBN..

Título Titulo..

Autor Autor..

Submit

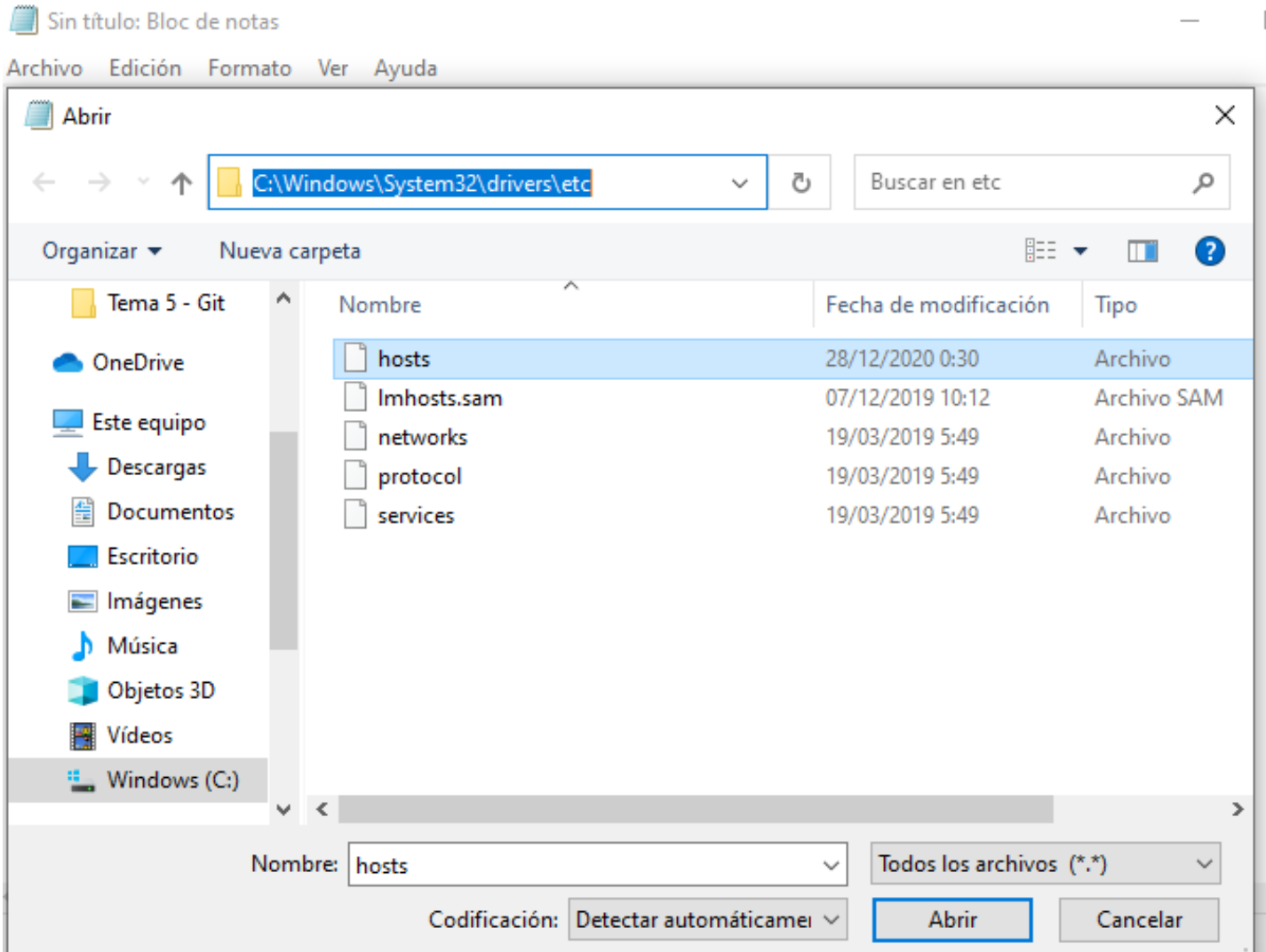
DESPLIEGUE - NGINX

- ▶ Estamos accediendo a la aplicación mediante la url:
<http://IP:8080/Biblioteca>
- ▶ En un caso real tendríamos un dominio y un certificado para usar HTTPS. En el servidor vamos a instalar Nginx y lo usaremos para configurar el acceso a nuestra aplicación.

```
sudo apt install nginx
```

- ▶ En primer lugar para simplificar la práctica vamos a editar el archivo hosts de Windows y de ese modo evitamos la instalación y configuración de un servidor DNS.
- ▶ Para ello ejecutamos el bloc de notas como Administrador y abrimos el archivo
c:\windows\system32\drivers\etc\hosts

DESPLIEGUE - NGINX

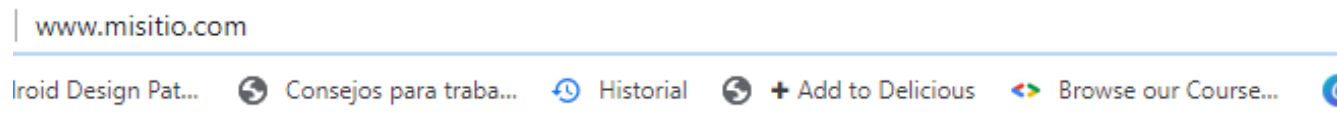


DESPLIEGUE - NGINX

- Añadimos al final la siguiente línea, sustituyendo la IP por la de vuestro servidor:

```
192.168.17.154 www.misitio.com
```

- Comprobamos desde Windows que podemos acceder con el dominio.



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

DESPLIEGUE - NGINX

- Vamos a crear un archivo llamado tomcat-basic.conf en la carpeta /etc/nginx/conf.d

- Incluimos en el archivo lo siguiente:

```
proxy_cache_path /tmp/NGINX_cache/ keys_zone=backcache:10m;
```

```
map $http_upgrade $connection_upgrade {  
    default upgrade;  
    ''      close;  
}
```

```
upstream tomcat {  
    # Use IP Hash for session persistence  
    ip_hash;  
  
    # List of Tomcat application servers  
    server 192.168.17.154:8080;  
  
}
```

```
server {  
    listen 80;  
    server_name www.misitio.com;  
  
    # Redirect all HTTP requests to HTTPS  
    location / {  
        return 301 https://$server_name$request_uri;  
    }  
}
```

DESPLIEGUE - NGINX

```
server {
    listen 443 ssl http2;
    server_name www.misitio.com;

    ssl_certificate      /etc/ssl/certs/misitio.crt;
    ssl_certificate_key  /etc/ssl/private/misitio.key;

    ssl_session_cache    shared:SSL:1m;
    ssl_prefer_server_ciphers on;

    # Load balance requests for /tomcat-app/ across Tomcat application servers
    location /Biblioteca/ {
        proxy_pass http://tomcat;
        proxy_cache backcache;
    }

    # Return a temporary redirect to the /tomcat-app/ directory
    # when user requests '/'
    location = / {
        return 302 /Biblioteca/;
    }

    #location /css/ {
    #    root /home/profesor/tmp;
    #}

    # WebSocket configuration
    location /wstunnel/ {
        proxy_pass https://tomcat;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $connection_upgrade;
    }
}
```

DESPLIEGUE - NGINX

- Es necesario crear el certificado digital para poder usar https. Para ello se ejecuta el siguiente comando teniendo en cuenta que en COMMON_NAME hay que introducir el valor www.misitio.com.

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -  
keyout /etc/ssl/private/misitio.key -out  
/etc/ssl/certs/misitio.crt
```

- Reiniciamos nginx y comprobamos que podemos acceder con el dominio



The screenshot shows a web browser interface. The address bar displays a warning "No es seguro" (Not secure) and the URL "misitio.com/Biblioteca/". Below the address bar, there are navigation icons and a search bar. The main content area features a table with three columns: ISBN, Título, and Autor. The table contains one row with the values 1, El Quijote, and Cervantes. Below the table, there are input fields for ISBN and Título.

ISBN	Título	Autor
1	El Quijote	Cervantes

ISBN

Título