Manual de programador de **ClienteFTP**

V 0.1

Audiencia

El presente documento está dirigido a todo personal técnico encargado del mantenimiento de la aplicación, así como, toda aquella persona que pudiera necesitar la compresión del código de programación utilizado para el desarrollo de la aplicación.

Aplicabilidad

El presente manual describe la programación realizada sobre el entorno de **ClienteFTP** para el desarrollo de una aplicación cliente ftp en un pc multiplataforma ya que está desarrollado con el lenguaje de programación Java.

Propósito

Este manual tiene como propósito la descripción de la estructura de programación llevada a cabo para el desarrollo de la aplicación previamente explicada, así como el manejo y gestión de variables, librerías y además herramientas de programación utilizadas en el proyecto, con el fin de que, cualquier personal técnico y con conocimientos de programación pueda entender y comprender la estructura de programación llevada a cabo.

Introducción

¿Qué es ClienteFTP?

ClienteFTP es un proyecto libre desarrollado para poder hacer uso de un cliente ftp como su nombre indica para conectarnos a un servidor ftp

Características

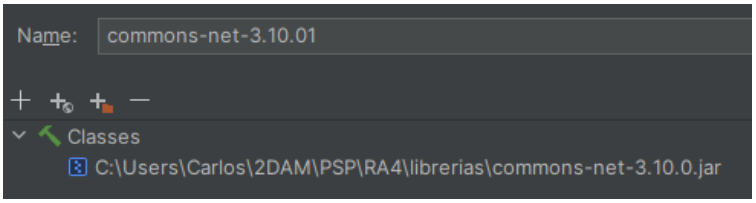
Incluye características como subir un archivo local al servidor o descargarnos un archivo del servidor en nuestra ruta local. También podremos ver los directorios remotos compartidos del servidor al usuario que estemos usando. Podremos generar un informe con el log del usuario dependiendo de las operaciones que haya realizado a lo largo de la sesión.

Proyecto

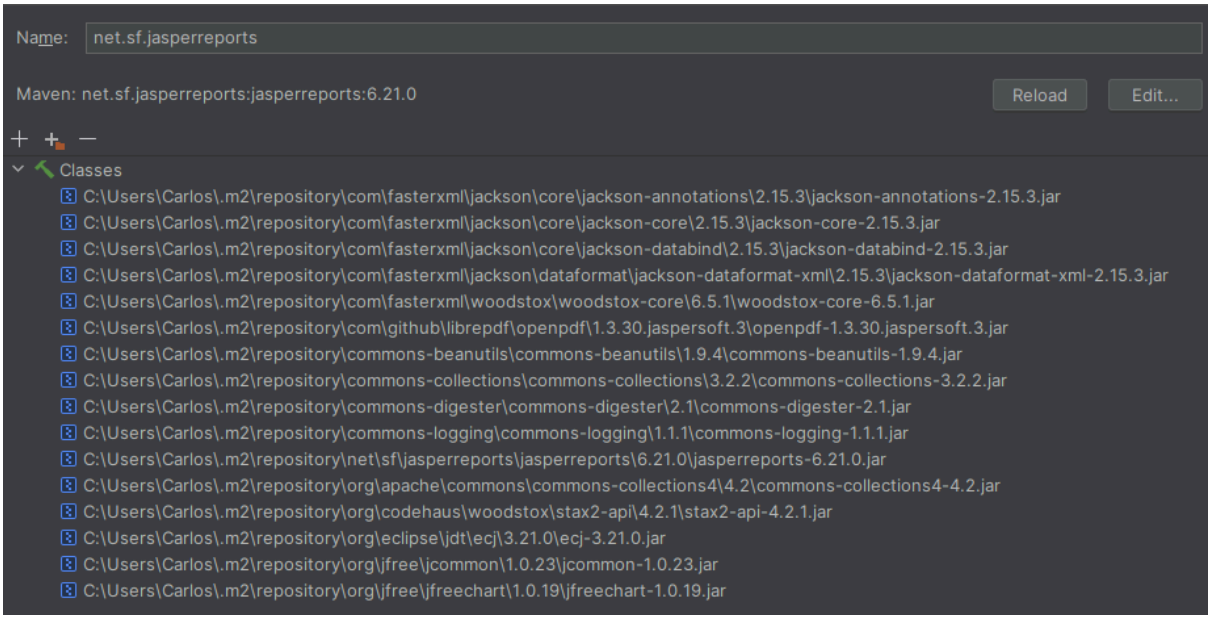
Librerías necesarias

Las librerías que necesitaremos para nuestro proyecto son las siguientes:

- commons-net-3.10.01 → para realizar conexiones y comunicaciones ftp.

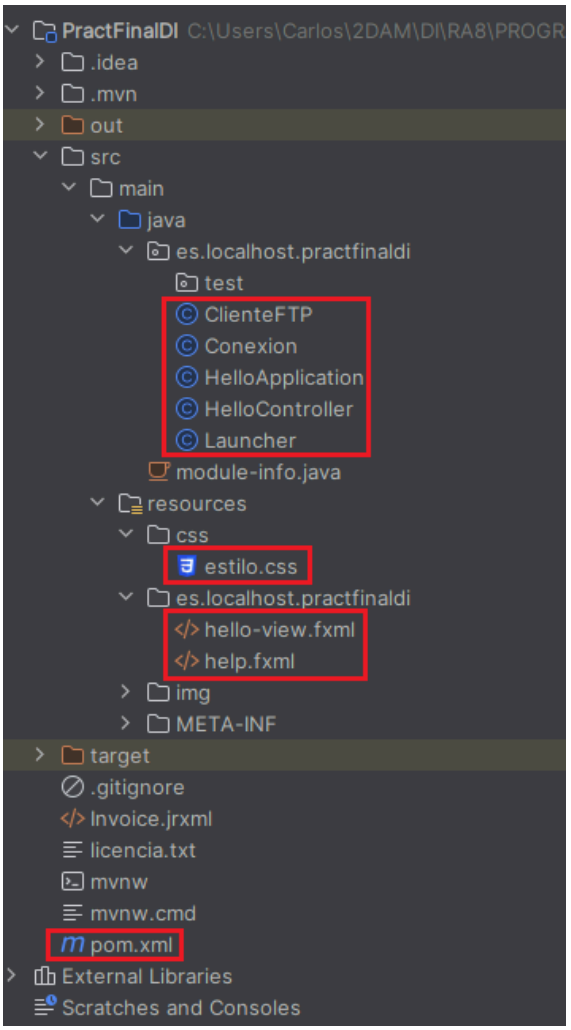


- net.sf.jasperreports → configuración y visualización de informes con jaspersoft.



Estructura

La estructura de nuestro proyecto ClienteFTP está formada por los siguientes directorios y archivos:



- **ClienteFTP**: clase que contiene los métodos necesarios para gestionar las operaciones que realiza el cliente en el servidor ftp.

- **Conexion**: clase que gestiona la conexión con la base de datos para insertar los registros del log de la aplicación.

- **HelloAplication**: clase principal que ejecuta la aplicación.

- **HelloController**: clase que controla todos los métodos y eventos de la aplicación.

- **Launcher**: clase que se ejecuta a hora de redistribuir la aplicación para ejecutar la aplicación.

- **estilo.css**: hoja de estilos .css de la aplicación para modificar el aspecto visual.

- **hello-view.fxml**: layout principal con estructura .xml que contiene todos los componentes que forma la aplicación.

- **help.fxml**: layout de ayuda que se ejecutará como una ventana emergente para proporcionar ayuda al usuario para realizar operaciones en el servidor.

- **pom.xml**: describe el proyecto, sus dependencias y componentes externos, y el orden de construcción de los elementos.

Código

Clase ClienteFTP

Método de conexión al servidor ftp

```
/**
    * Método que realiza la conexión con el servidor FTP.
    *
    * @param host Dirección IP del servidor.
    * @param usuario Usuario con el que nos autenticaremos en el login.
    * @param constras Contraseña con la que nos autenticaremos junto al usuario.
    * @param textArea Salida de los mensajes.
    */
   public boolean connect(String host, String usuario, String constras, TextArea textArea) {
      Conexion c = new Conexion("root", "", "ejemplo");
      boolean resultado = false;
      try {
         clienteFTP.connect(host);
         int codResp = clienteFTP.getReplyCode();
         if (!FTPReply.isPositiveCompletion(codResp)) {
            c.insertReg("ERROR", "Conexión rechazada con código de respuesta", LocalDateTime.now());

            textArea.appendText("ERROR: Conexión rechazada con código de respuesta " + codResp +
"\n");
            System.exit(2);
         }

         clienteFTP.enterLocalPassiveMode();
         clienteFTP.setFileType(FTP.BINARY_FILE_TYPE);

         if (usuario != null && constras != null) {
            boolean loginOK = clienteFTP.login(usuario, constras);
            if (loginOK) {
               c.insertReg("INFO", "Login con usuario " + usuario + " realizado", LocalDateTime.now());
               c.insertReg("INFO", "Conexión establecida", LocalDateTime.now());
               c.insertReg("INFO", "Directorio actual en servidor: " + clienteFTP.printWorkingDirectory(),
LocalDateTime.now());

               textArea.appendText("INFO: Login con usuario " + usuario + " realizado.\n");
               textArea.appendText("INFO: Conexión establecida.\n");
               textArea.appendText("INFO: Directorio actual en servidor: " +
clienteFTP.printWorkingDirectory() + "\n");
               resultado = true;
            }
         }
      } catch (IOException e) {
         c.insertReg("ERROR", "Conectando al servidor", LocalDateTime.now());
         textArea.appendText("ERROR: conectando al servidor");
         e.printStackTrace();
      }
      return resultado;
   }
```

Método para descargar archivos del servidor ftp

```
/**
    * Método para obtener ficheros del servidor FTP.
    *
    * @param fichServidor Fichero a obtener del servidor.
    * @param usuario Usuario con el que nos autenticaremos en el login.
    * @param constras Contraseña con la que nos autenticaremos junto al usuario.
    * @param host Dirección IP del servidor.
    * @param rutaLocal Ruta del cliente donde se guardará el archivo del servidor.
    * @param textArea Salida de los mensajes.
    */
   public void get(String host, String fichServidor, String usuario, String constras, String rutaLocal,
TextArea textArea) {
       Conexion c = new Conexion("root", "", "ejemplo");
       try {
           connect(host, usuario, constras, textArea);

           if (!clienteFTP.isConnected()) {
               c.insertReg("ERROR", "No se pudo conectar al servidor FTP", LocalDateTime.now());

               textArea.appendText("ERROR: No se pudo conectar al servidor FTP.");
               return;
           }

           String tamFichEnServidor = clienteFTP.getSize(fichServidor);
           if (tamFichEnServidor == null) {
               c.insertReg("ERROR", "Fichero " + fichServidor + " no existe en servidor", LocalDateTime.now());

               textArea.appendText("ERROR: Fichero " + fichServidor + " no existe en servidor.\n");
               return;
           }

           String nomFichLocal = rutaLocal + File.separator +
fichServidor.substring(fichServidor.lastIndexOf('/') + 1);
           try (FileOutputStream fos = new FileOutputStream(nomFichLocal)) {
               clienteFTP.retrieveFile(fichServidor, fos);
           }
           c.insertReg("INFO", "Se ha intentado copiar fichero " + fichServidor + " a fichero local " +
nomFichLocal, LocalDateTime.now());
           c.insertReg("INFO", "Respuesta del servidor: " + clienteFTP.getReplyString(),
LocalDateTime.now());

           textArea.appendText("INFO: Se ha intentado copiar fichero " + fichServidor + " a fichero local " +
nomFichLocal + ".\n");
           textArea.appendText("INFO: Respuesta del servidor:\n====\n" + clienteFTP.getReplyString() + "
====\n");

           int codResp = clienteFTP.getReplyCode();
           if (FTPReply.isPositiveCompletion(codResp)) {
               c.insertReg("INFO", "Servidor informa de que se ha completado satisfactoriamente la acción",
LocalDateTime.now());

               textArea.appendText("INFO: Servidor informa de que se ha completado satisfactoriamente la
acción.");
```

```
            showAlert("INFORMATION_MESSAGE", "Transferencia OK.", "Se ha completado de forma
exitosa la transferencia del archivo.", Alert.AlertType.INFORMATION);
        } else {
            c.insertReg("ERROR", "Servidor informa de que NO se ha completado satisfactoriamente la
acción", LocalDateTime.now());

            textArea.appendText("ERROR: Servidor informa de que NO se ha completado satisfactoriamente
la acción.");
        }
    } catch (IOException e) {
        c.insertReg("ERROR", "Conectando al servidor", LocalDateTime.now());

        textArea.appendText("ERROR: Conectando al servidor");
        e.printStackTrace();
    }
}
```

Método para subir archivos al servidor ftp

```
/**
 * Método para enviar ficheros al servidor FTP.
 *
 * @param host Dirección IP del servidor.
 * @param fichLocal Fichero a enviar al servidor.
 * @param usuario Usuario con el que nos autenticaremos en el login.
 * @param contras Contraseña con la que nos autenticaremos junto al usuario.
 * @param rutaRemota Ruta a guardar el archivo local en el servidor remoto ftp.
 * @param textArea Salida de los mensajes.
 */
public void post(String host, String fichLocal, String usuario, String contras, String rutaRemota, TextArea
textArea) {
    File fLocal = new File(fichLocal);
    Conexion c = new Conexion("root", "", "ejemplo");

    if (!fLocal.exists() || !fLocal.isFile()) {
        c.insertReg("ERROR", "Fichero " + fichLocal + " no existe", LocalDateTime.now());

        textArea.appendText("ERROR: Fichero " + fichLocal + " no existe.\n");
        return;
    }
    try {
        connect(host, usuario, contras, textArea);

        String nomFichRemoto = rutaRemota + "/" + fLocal.getName();
        clienteFTP.storeFile(nomFichRemoto, new FileInputStream(fichLocal));

        c.insertReg("INFO", "Se ha intentado copiar fichero local al servidor, con nombre " + fichLocal,
LocalDateTime.now());
        c.insertReg("INFO", "Respuesta del servidor: " + clienteFTP.getReplyString(),
LocalDateTime.now());

        textArea.appendText("INFO: Se ha intentado copiar fichero local al servidor, con nombre " +
fichLocal + ".\n");
        textArea.appendText("INFO: Respuesta del servidor:\n====\n " + clienteFTP.getReplyString() + "
====\n");
```

```
            int codResp = clienteFTP.getReplyCode();
            if (FTPReply.isPositiveCompletion(codResp)) {
                c.insertReg("INFO", "Servidor informa de que se ha completado satisfactoriamente la acción",
LocalDateTime.now());

                textArea.appendText("INFO: Servidor informa de que se ha completado satisfactoriamente la
acción.");
                showAlert("INFORMATION_MESSAGE", "Transferencia OK.", "Se ha completado de forma
exitosa la subida del archivo al servidor.", Alert.AlertType.INFORMATION);
            } else {
                c.insertReg("ERROR", "Servidor informa de que NO se ha completado satisfactoriamente la
acción", LocalDateTime.now());

                textArea.appendText("ERROR: Servidor informa de que NO se ha completado satisfactoriamente
la acción.");
            }

        } catch (IOException e) {
            c.insertReg("ERROR", "Conectando al servidor", LocalDateTime.now());

            textArea.appendText("ERROR: Conectando al servidor");
            e.printStackTrace();
        }
    }
```

Método que cierra conexión con el servidor ftp

```
/**
     * Método que cierra la conexión con el servidor ftp.
     * @param textArea Salida de los mensajes.
     */
    public void closeConnection(TextArea textArea){
        Conexion c = new Conexion("root", "", "ejemplo");
        try {
            clienteFTP.disconnect();
            c.insertReg("INFO", "Conexion cerrada", LocalDateTime.now());

            textArea.appendText("INFO: conexión cerrada.");
            showAlert("INFORMATION_MESSAGE", "Conexión cerrada.", "Se ha cerrado la conexión al
servidor ftp de forma existosa.", Alert.AlertType.INFORMATION);
        } catch (IOException e) {
            c.insertReg("ERROR", "No se pudo cerrar la conexión", LocalDateTime.now());

            textArea.appendText("ERROR: No se pudo cerrar la conexión.");
        }
    }
```

Clase Conexion

Método que establece conexión con BD

```
private Connection conectar(){
        Connection conexion = null;
```

```
        String url = URL + db;
        try{
            Class.forName(DRIVER);

            conexion = DriverManager.getConnection(url, user, password);
        }
        catch (SQLException e) {
            System.out.println("Error loading connection...");
            e.printStackTrace();
        }
        catch (ClassNotFoundException e) {
            System.out.println("Error loading driver...");
        }
        return conexion;
    }
```

Método que inserta los registros del log en BD

```
public void insertReg(String tipo, String mensaje, LocalDateTime fechaHora) {
        String sql = "INSERT INTO log (tipo, mensaje, fecha_hora) VALUES (?, ?, ?)";

        try (
                Connection conn = conectar();
                PreparedStatement pstmt = conn.prepareStatement(sql);
        ) {
            pstmt.setString(1, tipo);
            pstmt.setString(2, mensaje);
            pstmt.setTimestamp(3, Timestamp.valueOf(fechaHora));

            pstmt.executeUpdate();

            System.out.println("Registro insertado correctamente en la tabla log.");
        } catch (SQLException e) {
            System.out.println("Error al insertar registro en la tabla log: " + e.getMessage());
        }
    }
```

Método que borrar los logs de BD al iniciar la aplicación

```
public void delReg(){
        try {
            Connection conn = conectar();

            Statement sentencia = conn.createStatement();

            String sql = "DELETE FROM log";
            sentencia.executeUpdate(sql);

            sentencia.close();
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
```

```
    }
  }
```

Clase HelloController

Método que muestra la estructura de directorios servidor ftp

```
private void mostrarDirectorioRemoto() {
    if (cliente != null && cliente.isConnected()) {
        cargarDirectorioRemoto(currentRemoteDirectory);
    } else {
        showAlert("Error", "No conectado", "No se ha establecido una conexión FTP válida.",
Alert.AlertType.ERROR);
    }
  }

  private void cargarDirectorioRemoto(String path) {
    try {
        FTPFile[] files = cliente.getFTPClient().listFiles(path);
        TreeItem<String> rootItem = new TreeItem<>(path);
        for (FTPFile file : files) {
            TreeItem<String> newItem = new TreeItem<>(file.getName());
            rootItem.getChildren().add(newItem);
            if (file.isDirectory()) {
                newItem.getChildren().add(new TreeItem<>());
            }
        }
        tvRemoto.setRoot(rootItem);
        currentRemoteDirectory = path;
        remoteDirectoryStack.push(path);
    } catch (IOException e) {
        showAlert("ERROR_MESSAGE", "Error al obtener archivos remotos", "No se pudieron obtener los
archivos remotos.", Alert.AlertType.ERROR);
    }
  }
```

Método que muestra la estructura de directorios locales

```
private void mostrarDirectorioLocal() {
    File localRoot = new File(System.getProperty("user.dir"));
    TreeItem<String> rootItem = new TreeItem<>(localRoot.getName());
    populateTreeView(localRoot, rootItem);
    tvLocal.setRoot(rootItem);
  }

  private void populateTreeView(File file, TreeItem<String> parentItem) {
    if (file.isDirectory()) {
        File[] files = file.listFiles();
        if (files != null) {
            for (File childFile : files) {
```

```
                TreeItem<String> newItem = new TreeItem<>(childFile.getName());
                parentItem.getChildren().add(newItem);
                populateTreeView(childFile, newItem);
            }
        }
    }
}
```

Método que muestra un informe con el log de la aplicación

```
private void mostrarInforme(String reportFile, String host, String db, String user, String passwd ){
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection connection = DriverManager.getConnection("jdbc:mysql://" + host + ":3306/" + db, user,
passwd);
        JasperReport jr = JasperCompileManager.compileReport(reportFile);
        JasperPrint jp = JasperFillManager.fillReport(jr, null, connection);
        JasperViewer.viewReport(jp);
        connection.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Base de datos

Se ha implantado una base de datos para guardar los log de la aplicación y a posteriori mostrarlo en un informe en jaspersoft.
La base de datos contiene la siguiente estructura:

Field	Type	Null	Key	Default	Extra
id_log	int(5)	NO	PRI	*NULL*	auto_increment
tipo	varchar(10)	NO		*NULL*	
mensaje	varchar(150)	NO		*NULL*	
fecha_hora	datetime	NO		*NULL*	

Los registros del log de la aplicación se mostrarían de la siguiente forma almacenados en la base de datos:

			id_log	tipo	mensaje	fecha_hora
	Editar	Copiar Borrar			Login con usuario usuario realizado	2024-03-07 15:38:35
	Editar	Copiar Borrar			Conexión establecida	2024-03-07 15:38:36
	Editar	Copiar Borrar	195	INFO	Directorio actual en servidor: /	2024-03-07 15:38:36
	Editar	Copiar Borrar	196	INFO	Conexion cerrada	2024-03-07 15:38:45
	Editar	Copiar Borrar	197	INFO	Login con usuario pepe realizado	2024-03-07 15:38:52
	Editar	Copiar Borrar	198	INFO	Conexión establecida	2024-03-07 15:38:52
	Editar	Copiar Borrar	199	INFO	Directorio actual en servidor: /	2024-03-07 15:38:52
	Editar	Copiar Borrar	200	INFO	Login con usuario pepe realizado	2024-03-07 15:39:22
	Editar	Copiar Borrar	201	INFO	Conexión establecida	2024-03-07 15:39:22
	Editar	Copiar Borrar	202	INFO	Directorio actual en servidor: /	2024-03-07 15:39:22
	Editar	Copiar Borrar	203	INFO	Se ha intentado copiar fichero /cruduser/pom.xml a...	2024-03-07 15:39:22
	Editar	Copiar Borrar	204	INFO	Respuesta del servidor: 226 Transfer OK	2024-03-07 15:39:22
	Editar	Copiar Borrar	205	INFO	Servidor informa de que se ha completado satisfact...	2024-03-07 15:39:22

Haga clic en la flecha desplegable para cambiar la visibilidad de la columna.

La aplicación al iniciarse establece conexión con el servidor de base de datos para poder almacenar los registros, por ello es importante que tengamos configurada una base de datos, en caso contrario no funcionará.

```
@Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        Conexion c = new Conexion("root", "", "ejemplo");
        c.delReg();
    }
```