*Documentación en una Aplicación JavaFX*

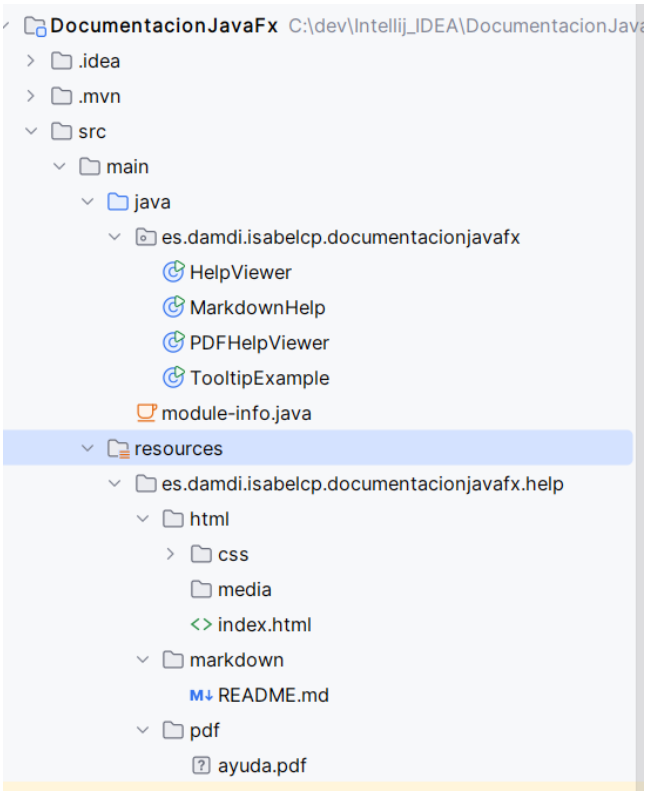
*Documentación en una Aplicación JavaFX*

## Introducción

En una aplicación JavaFX, es fundamental proporcionar un sistema de ayuda para los usuarios. Aunque JavaHelp era una solución popular en aplicaciones Java basadas en Swing, en JavaFX existen varias opciones más modernas y flexibles para implementar un sistema de documentación.

Este documento explora diferentes formas de integrar ayuda en una aplicación JavaFX.

Creamos el proyecto:



**Opciones para Mostrar Ayuda en JavaFX**

## 1. Uso de WebView con HTML

Para utilizar HelpViewer en tu aplicación JavaFX, no necesitas una dependencia adicional específica en el pom.xml, ya que WebView es parte del paquete javafx.web, el cual está incluido en JavaFX. Sin embargo, si estás usando Maven, necesitas asegurarte de que JavaFX esté correctamente configurado en tu proyecto.

### Agregar JavaFX en pom.xml

Si aún no tienes JavaFX configurado en tu proyecto Maven, agrega lo siguiente en tu pom.xml:

```
<dependencies>
  <!-- Dependencia de JavaFX -->
  <dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-controls</artifactId>
    <version>21</version>
  </dependency>

  <dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-web</artifactId>
    <version>21</version>
  </dependency>
</dependencies>
```

Una de las opciones más versátiles es utilizar un WebView para mostrar archivos HTML dentro de la aplicación.

### Explicación del código:

- Se usa un WebView para cargar un archivo HTML local (index.html).
- Se obtiene la ruta del recurso usando getClass().getResource().
- Se establece la escena y se muestra la ventana de ayuda.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.web.WebView;
import javafx.stage.Stage;

import java.net.URL;

public class HelpViewer extends Application {

   @Override
   public void start(Stage primaryStage) {
      // Crear un componente WebView para mostrar el contenido HTML
      WebView webView = new WebView();

      // Obtener la URL del archivo HTML dentro de los recursos
      URL url = getClass().getResource("help/html/index.html");

      if (url != null) {
         // Cargar el archivo HTML en el WebView
         webView.getEngine().load(url.toExternalForm());
      } else {
         System.err.println("⚠ No se encontró el archivo HTML en el classpath.");
         webView.getEngine().loadContent("<html><body><h2>Error: No se pudo cargar la ayuda.</h2></body></html>");
      }

      // Configurar la escena y mostrar la ventana de ayuda
      primaryStage.setScene(new Scene(webView, 800, 600));
      primaryStage.setTitle("Ayuda - Documentación");
      primaryStage.show();
   }

   public static void main(String[] args) {
      launch(args);
   }
}
```

## 2. Uso de Markdown con Flexmark

Si prefieres escribir documentación en Markdown, puedes cargar un archivo .md, convertirlo a HTML y mostrarlo en un WebView.

https://github.com/vsch/flexmark-java

**Explicación del código:**

- **Carga del archivo Markdown:** loadMarkdown("src/main/resources/help/Readme.md") lee el contenido del archivo Readme.md y lo almacena como una cadena de texto.
- **Conversión de Markdown a HTML:**
  - Parser parser = Parser.builder().build(); crea una instancia del analizador de Markdown.
  - HtmlRenderer renderer = HtmlRenderer.builder().build(); crea un renderizador que convertirá el Markdown a HTML.
  - Node document = parser.parse(markdownContent); analiza el contenido de Markdown y lo convierte en una estructura de nodos.
  - String htmlContent = renderer.render(document); convierte la estructura de nodos en código HTML válido.
- **Carga del HTML en WebView:** webView.getEngine().loadContent(htmlContent); carga el HTML generado en el visor web.

**Agregar dependencia en pom.xml:**

```
<dependency>
  <groupId>com.vladsch.flexmark</groupId>
  <artifactId>flexmark-all</artifactId>
  <version>0.64.8</version>
</dependency>
```

```
import com.vladsch.flexmark.html.HtmlRenderer;
import com.vladsch.flexmark.parser.Parser;
import com.vladsch.flexmark.util.ast.Node;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.web.WebView;
import javafx.stage.Stage;
import java.io.IOException;
import java.net.URISyntaxException;
import java.net.URL;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
```

```
public class MarkdownHelp extends Application {

    @Override
    public void start(Stage primaryStage) {
        // Cargar el contenido del archivo Markdown
        String markdownContent = loadMarkdown();

        // Convertir Markdown a HTML utilizando Flexmark
        Parser parser = Parser.builder().build();
        HtmlRenderer renderer = HtmlRenderer.builder().build();
        Node document = parser.parse(markdownContent);
        String htmlContent = renderer.render(document);

        // Crear un WebView para mostrar el contenido HTML generado
        WebView webView = new WebView();
        webView.getEngine().loadContent(htmlContent);

        // Configurar la escena y mostrar la ventana
        primaryStage.setScene(new Scene(webView, 800, 600));
        primaryStage.setTitle("Ayuda en Markdown");
        primaryStage.show();
    }

    /**
     * Método para cargar el contenido del archivo Markdown desde los recursos del classpath.
     *
     * @return Contenido del archivo Markdown como String, o un mensaje de error si ocurre un
problema.
     */
    private String loadMarkdown() {
        // Obtener la URL del archivo Markdown en resources
        URL resourceUrl = getClass().getResource("help/markdown/Readme.md");

        if (resourceUrl == null) {
            System.err.println("⚠ El archivo Markdown no se encontró en el classpath.");
            return "Error: No se pudo encontrar el archivo Markdown.";
        }

        try {
            // Convertir la URL en una ruta válida para leer el archivo
            Path path = Paths.get(resourceUrl.toURI());
            return Files.readString(path); // Método más eficiente que `readAllBytes()`
        } catch (IOException | URISyntaxException e) {
            System.err.println("❌ Error al cargar el archivo Markdown: " + e.getMessage());
            return "Error al cargar el archivo Markdown.";
        }
    }
```

## 3. Mostrar PDF con PDFViewerFX

Si tienes manuales en formato PDF, puedes integrarlos con la biblioteca PDFViewerFX.

https://github.com/Dansoftowner/PDFViewerFX

**Explicación del código:**

- Busca el archivo ayuda.pdf dentro de los recursos (src/main/resources/help/pdf/).
- Convierte la URL en un File y carga el PDF.
- Si el archivo existe, lo muestra en la ventana; si no, muestra un mensaje de error.

Agregar dependencia en pom.xml:

```
<repositories>
  <repository>
    <id>jitpack.io</id>
    <url>https://jitpack.io</url>
  </repository>
</repositories>
```

```
<dependency>
  <groupId>com.github.Dansoftowner</groupId>
  <artifactId>PDFViewerFX</artifactId>
  <version>0.8</version>
</dependency>
```

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import com.dansoftware.pdfdisplayer.PDFDisplayer;

import java.io.File;
import java.io.IOException;
import java.net.URISyntaxException;
import java.net.URL;

public class PDFHelpViewer extends Application {
  @Override
  public void start(Stage primaryStage) {
    // Crear el visor PDF
    PDFDisplayer displayer = new PDFDisplayer();
    Scene scene = new Scene(displayer.toNode());

    // Obtener la URL del PDF
    URL pdfUrl = getClass().getResource("help/pdf/ayuda.pdf");
```

```
if (pdfUrl == null) {
        System.err.println("⚠ El archivo PDF no se encontró en el classpath.");
        return;
    }

    try {
        // Convertir la URL en un archivo y cargar el PDF
        File pdfFile = new File(pdfUrl.toURI());
        displayer.loadPDF(pdfFile);
    } catch (URISyntaxException | IOException e) {
        System.err.println("❌ Error al cargar el PDF: " + e.getMessage());
        return;
    }

    // Configurar y mostrar la ventana
    primaryStage.setTitle("Ayuda - Manual de Usuario");
    primaryStage.setScene(scene);
    primaryStage.show();
  }

  public static void main(String[] args) {
    launch(args);
  }
}
```

## 4. Uso de Tooltips y Diálogos de Ayuda Contextual

Para mostrar ayuda breve en ciertos elementos, puedes usar Tooltip o Alert.

**Explicación del código:**

- Se crea un Button con un Tooltip para mostrar ayuda contextual.
- Se usa Alert para mostrar información emergente.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.scene.control.Alert.AlertType;

public class TooltipExample extends Application {
   @Override
   public void start(Stage primaryStage) {
      // Crear un botón con el texto '?'
      Button helpButton = new Button("?");

      // Crear un tooltip con un mensaje informativo
      Tooltip tooltip = new Tooltip("Haz clic aquí para obtener más información.");
      Tooltip.install(helpButton, tooltip);

      // Definir la acción del botón cuando se hace clic
      helpButton.setOnAction(event -> {
         // Mostrar ventana de Alerta de advertencia
         Alert alert = new Alert(AlertType.WARNING);
         alert.setTitle("Alerta");
         alert.setHeaderText("Precaución");
         alert.setContentText("Ten en cuenta la información proporcionada.");
         alert.showAndWait();

         // Mostrar ventana de Información adicional
         Alert infoAlert = new Alert(AlertType.INFORMATION);
         infoAlert.setTitle("Información");
         infoAlert.setHeaderText("Detalles Adicionales");
         infoAlert.setContentText("Esta es una ventana de información adicional.");
         infoAlert.showAndWait();
      });

      // Crear un contenedor VBox y agregar el botón
      VBox root = new VBox(helpButton);

      // Crear la escena con dimensiones específicas
      Scene scene = new Scene(root, 200, 100);

      // Configurar la ventana principal
      primaryStage.setScene(scene);
```

```
        primaryStage.setTitle("Ayuda Contextual");
        primaryStage.show();
    }

    // Método principal para lanzar la aplicación
    public static void main(String[] args) {
        launch(args);
    }
}
```

## 5. Conclusión

Dependiendo de las necesidades de la aplicación, se pueden combinar distintas estrategias para ofrecer ayuda y documentación a los usuarios. Implementando estas soluciones, podemos ofrecer una experiencia de usuario completa dentro de una aplicación JavaFX.