Sistemas Interactivos

Plataforma Multi-App

Juan José Méndez Torrero
i42metoj@uco.es

Universidad de Córdoba

9 de mayo de 2019

Índice

1. Introducción e identificación del problema

La plataforma **Multi-App** consiste en una aplicación en la que se encuentran una serie de aplicaciones distintas. Esta plataforma nace de la necesidad de no tener que tener muchas aplicaciones abiertas en el escritorio, ya que pocas veces un usuario normal utiliza todas y cada una de esas aplicaciones. **Multi-App** aborda el problema de tener que tener una aplicación para cada una de las siguientes tareas:

1. Correo electrónico.
2. Lista de tareas (*To-do*).
3. Noticias.
4. Reproductor de música.

Multi-App incluye cuatro secciones para cada una de las tareas. Esto hace que nuestro ordenador no tenga la necesidad de estar ejecutando estas cuatro aplicaciones a la vez, sino que, con una sola aplicación podamos consultar el correo entrante o incluso reproducir música.

Este problema a sido abordado ya por otras aplicaciones, aunque la gran mayoría son aplicaciones de pago. Un ejemplo parecido a **Multi-App** es la aplicación **Franz** (https://meetfranz.com/), la cuál también es gratuita. Esta aplicación nos permite tener, en una misma aplicación, conversaciones tanto de correo, Whatsapp, realizar videollamadas con Skype, etc.

2. Especificación de requisitos

En esta sección se abordará la cuestión de cuáles son los requisitos de usuario y cuáles son los requisitos de sistema.

2.1. Requisitos de usuario

Los requisitos de usuario son declaraciones en lenguaje natural y en diversos diagramas de los servicios del sistema y de las restricciones bajo las que debe operar un usuario. Los requisitos son los siguientes:

- **RU-01:** El usuario de la aplicación no necesitará introducir ni nombre/usuario, ni contraseña para poder hacer uso de la aplicación.
- **RU-02:** El usuario podrá guardar sus noticias y canciones favoritas para poder volver a verlas en el futuro.
- **RU-03:** El usuario podrá entrar en otra aplicación sin tener la necesidad de salir de la que está utilizando en ese momento.
- **RU-04:** El usuario tendrá la posibilidad de refrescar la sección de noticias.

2.2. Requisitos de sistema

Los requisitos de sistema establecen detalladamente las funciones y restricciones que debe cumplir el sistema para cumplir con éxito y de forma satisfactoria los requisitos de usuario. Los requisitos de sistema de la plataforma **Multi-App** son los siguientes:

1. **RS-01:** La aplicación deberá ser ejecutada en uno de los siguientes sistemas operativos: Mac OS, Windows o Linux.
2. **RS-02:** La computadora no ha de estar conectada a internet para poder funcionar.
3. **RS-03:** Tener instalada la última versión operativa de Java.
4. **RS-04:** Tener acceso a un teclado y ratón para poder moverse entre las distintas aplicaciones.

3. Diseño y especificación de la interfaz

En esta sección se abordará el problema del diseño y especificación de la interfaz. Primero se explicará la pantalla principal, ya que tiene un diseño distinto al resto de las pantallas. Con respecto al diseño de todas y cada una de las pantallas, está realizado de tal manera que el usuario no tenga que aumentar o disminuir el tamaño de la pantalla para poder apreciar todos los componentes.

Como hemos comentado, primero vamos a hablar de la primera pantalla (Figura 1)

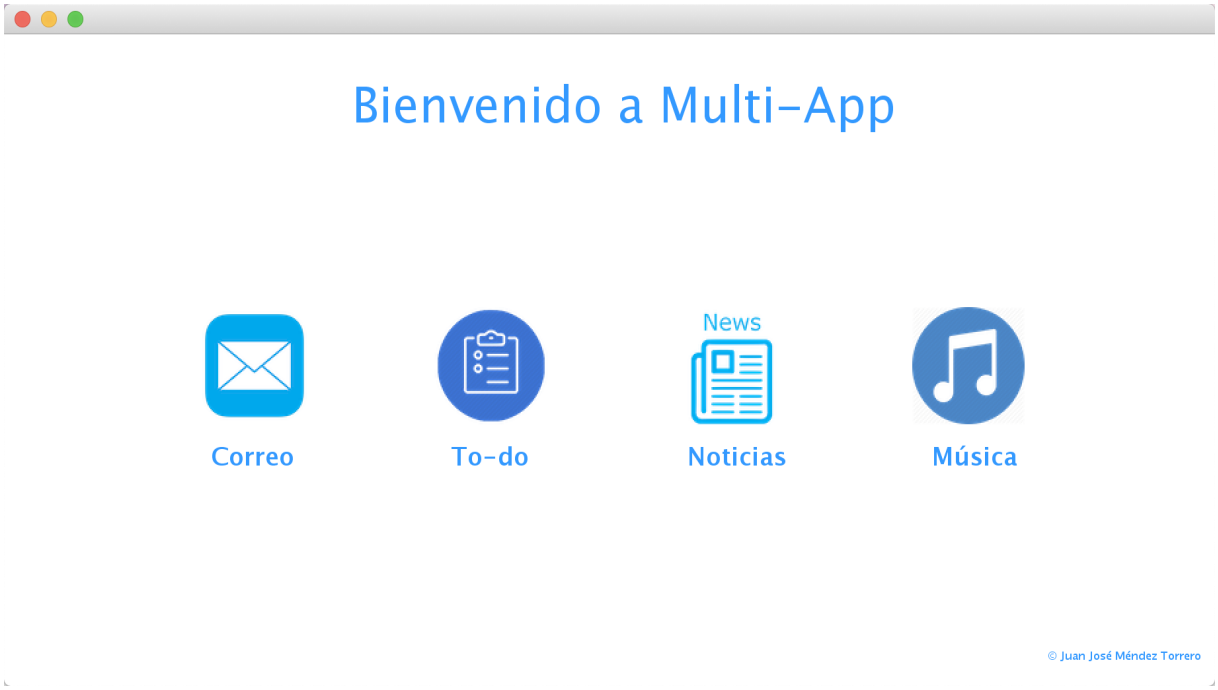

Figura 1: Pantalla principal de **Multi-App**

Como vemos, el usuario nada mas iniciar la aplicación, se encontrará con cuatro opciones distintas. Con respecto al diseño de esta pantalla, se ha intentado hacerla lo más minimalista posible, ya que así el usuario no necesitará un manual de uso para saber cómo funciona.

Seguidamente, con respecto al resto de pantallas, todas tienen el mismo diseño. Primero, una barra lateral en la que se incluyen las diferente acciones que se puede hacer en cada aplicación junto con un botón *Home* que llevará al usuario a la pantalla principal. Segundo, un título que representará a la pantalla en la que estamos. Finalmente, un contenedor en el que se muestra el contenido de la aplicación. Por ejemplo, vemos que en la pantalla de la aplicación **Noticias** (Figura 2) se encuentran todas estas secciones.

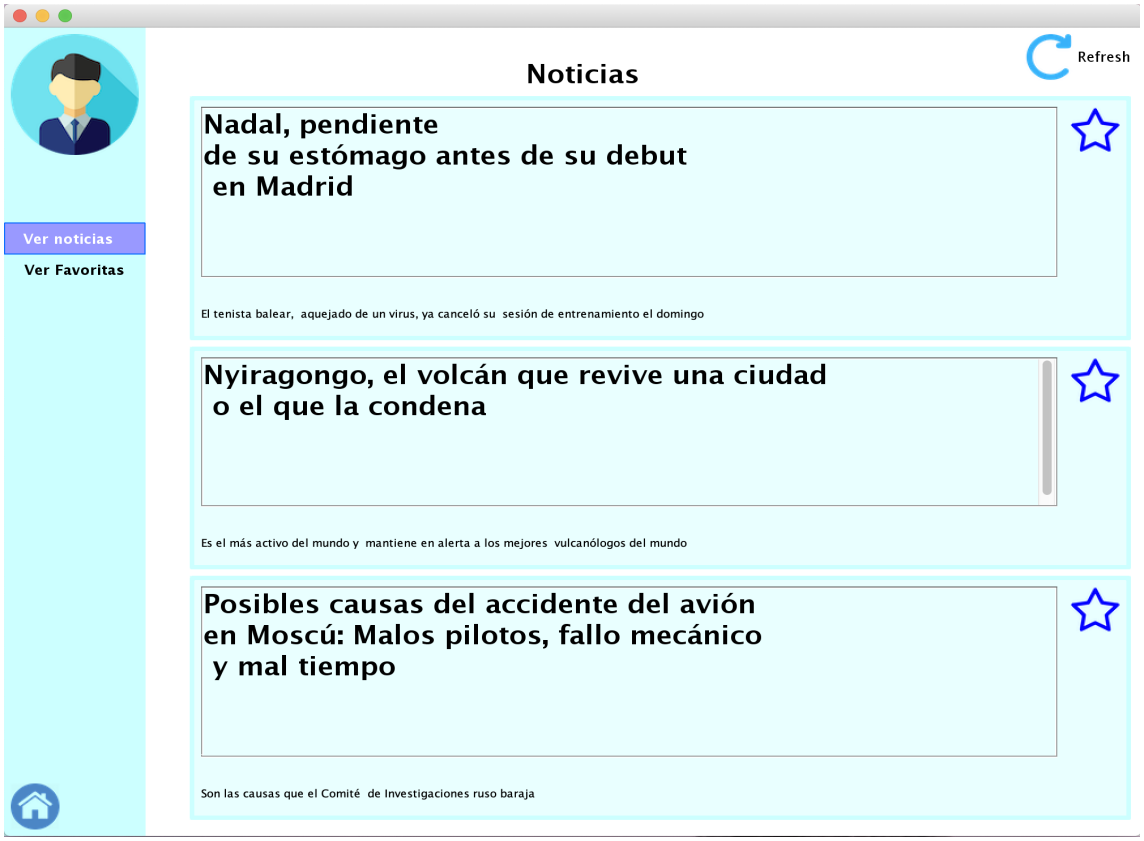

Figura 2: Pantalla de la aplicación **Noticias** de **Multi-App**

Una vez explicado el contenido por defecto de todas las pantallas (menos la principal), pasamos a mostrar todas y cada una de las pantallas de nuestra aplicación.

3.1. Pantallas de la aplicación Correo

Esta aplicación se encargará de mostrar los correos de nuestra bandeja de entrada y nos dará la posibilidad de entrar en un correo concreto junto con la de eliminarlo. Finalmente, en el apartado de bandeja de entrada se puede apreciar un botón para refrescar en el caso de haber añadido algún nuevo correo. Esta pantalla principal se puede apreciar en la Figura 3.

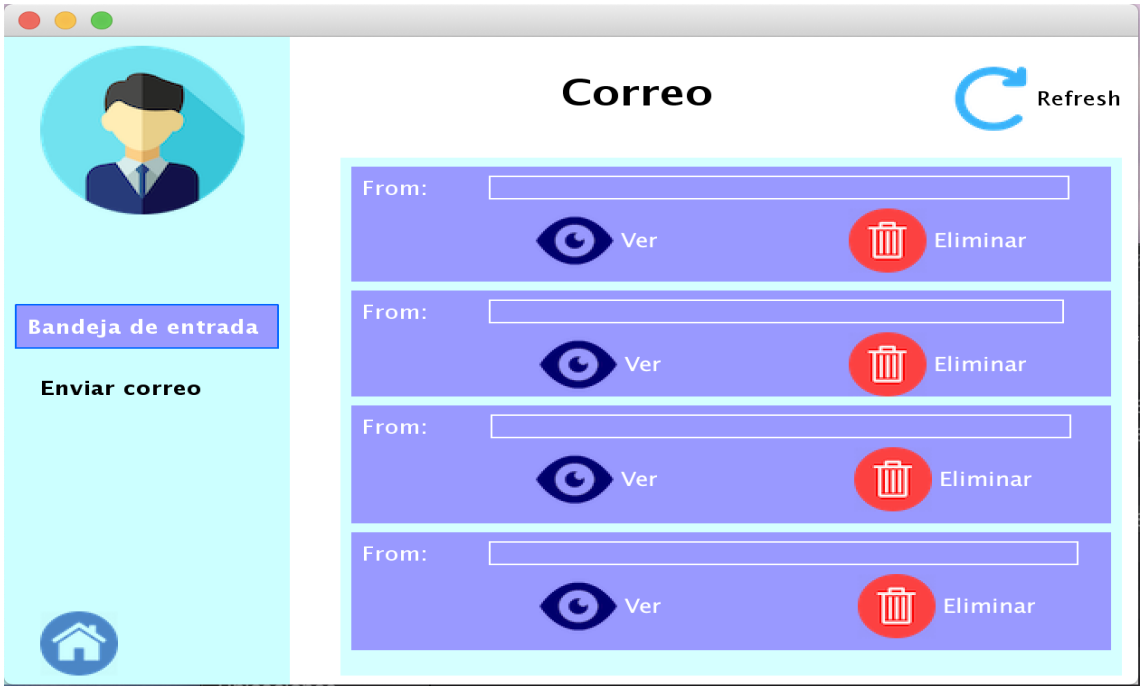

Figura 3: Pantalla principal de la aplicación **Correo** de **Multi-App**

Como vemos, hay tres botones principales, el botón de *Ver*, el botón de *Eliminar* y el botón de *Refrescar*. Una vez que ya hemos visto la bandeja de entrada, nos gustaría ver un correo en concreto, es decir, el usuario pinchará en el botón *Ver* y nos llevará a la pantalla de *Enviar Correo* (Figura 4).

Figura 4: Pantalla *Ver Correo* de la aplicación **Correo** de **Multi-App**

Como vemos, en esta pantalla seguimos teniendo la posibilidad de volver a la pantalla anterior, a la bandeja de entrada, o a la pantalla de *Enviar Correo*(Figura 5).

Figura 5: Pantalla *Enviar Correo* de la aplicación **Correo** de **Multi-App**

Como vemos en la Figura 5, el usuario tendrá que introducir un correo desde el cual se enviará el mensaje, el correo de la persona a la que se enviará el mensaje, un concepto para dicho mensaje y finalmente el contenido del mensaje. Una vez escrito el mensaje, el usuario tendrá que pulsar el botón de *Enviar* para poder enviar el mensaje. Esta acción nos redireccionará a la pantalla principal de Correo(Figura 3). Una vez en esa pantalla, para poder ver el mensaje enviado se tendrá que pulsar el botón de refrescar, y así se podrá comprobar que el mensaje ha sido enviado.

3.2. Pantalla de la aplicación To-do

En esta sección se explicará la pantalla de la aplicación *To-do* de la plataforma **Multi-App**. En la Figura 6 podemos apreciar el diseño de esta aplicación, como vemos, sigue el estándar explicado al principio, con la diferencia de que esta pantalla no tiene más acciones en la barra lateral.

Figura 6: Pantalla principal de la aplicación **To-do** de **Multi-App**

Como vemos, el usuario podrá añadir una nueva tarea introduciendo el concepto de dicha tarea y posteriormente presionando el botón +. La nueva tarea se añadirá a la primera sección de tareas que esté sin utilizar. En el caso de que todas estén en uso, la nueva tarea se añadirá a la primera sección. Una vez que el usuario ha realizado alguna de las tareas con éxito, podrá borrarlas haciendo click en el botón de eliminar(Botón rojo con la forma de una papelera).

3.3. Pantallas de la aplicación Noticias

En esta sección se explicará el diseño de las pantallas pertenecientes a la aplicación de *Noticias* de la plataforma **Multi-App**. Primero, explicaremos el diseño de la pantalla principal (Figura 7).

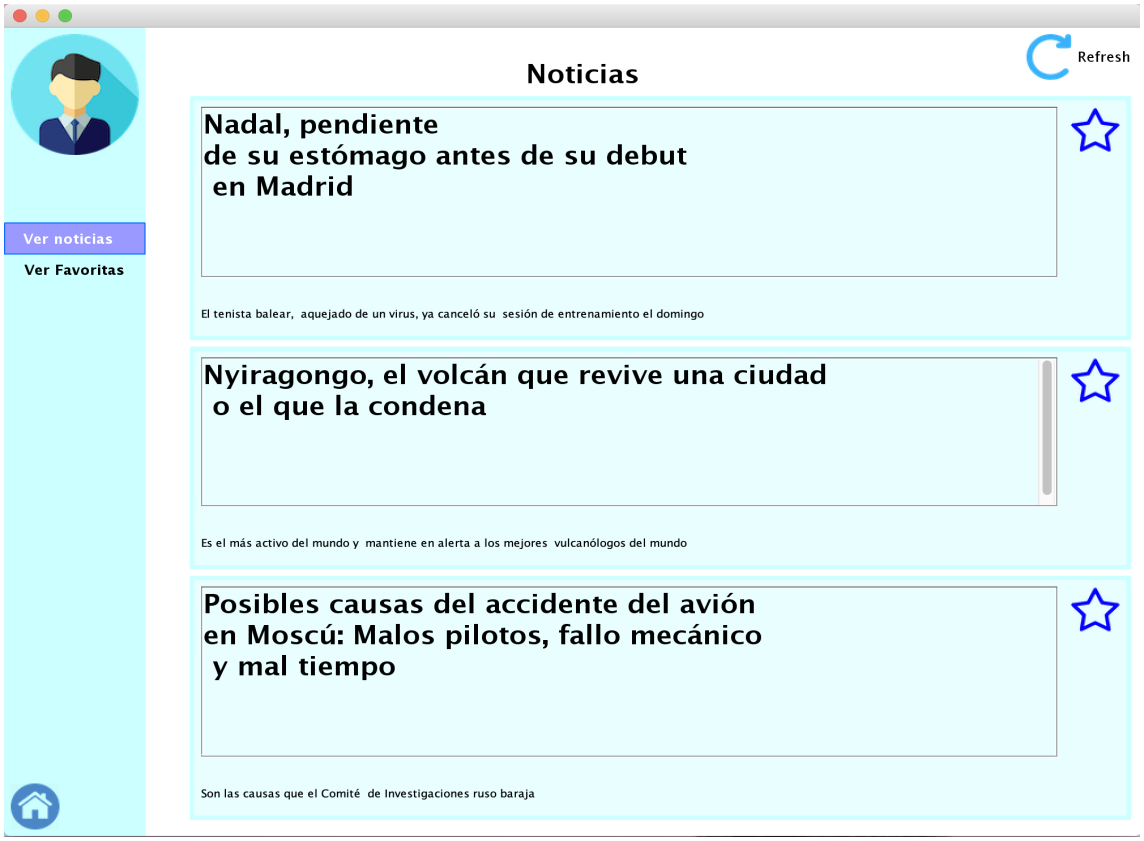

Figura 7: Pantalla principal de la aplicación **Noticias** de **Multi-App**

Como podemos observar, el usuario se encontrará con una serie de noticias en la que se mostrará su título y una pequeña descripción justo por debajo del título. El usuario podrá pulsar en el botón *refresh* para poder mostrar distintas noticias. En el caso de que el usuario tenga la intención de guardar una noticia porque le haya resultado muy llamativa, podrá añadirla a *Favoritos* con tan solo pulsar el botón de favoritos(Botón con forma de estrella). Una vez hecho esto, el usuario podrá ver sus noticias favoritas en la sección de *Ver Favoritas*. Esta pantalla, Figura 8, tiene el mismo diseño que las demás.

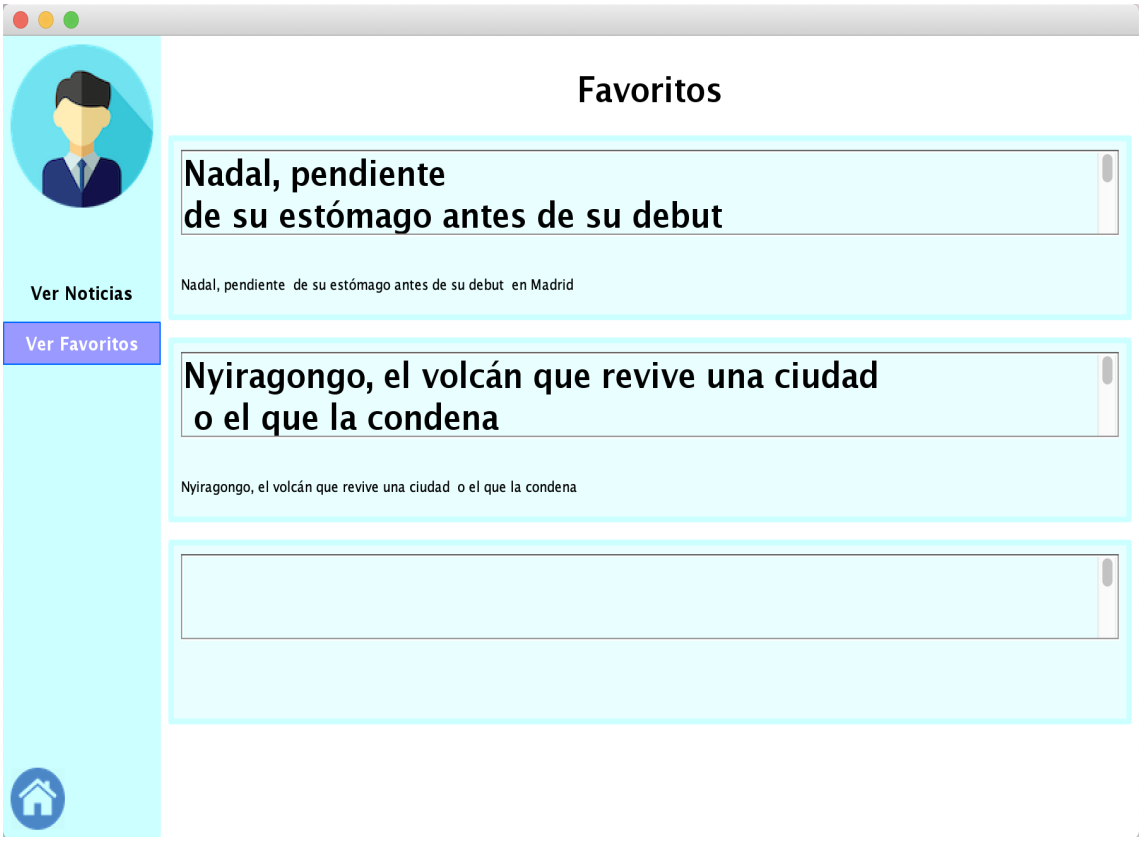

Figura 8: Pantalla *Favoritas* de la aplicación **Noticias** de **Multi-App**

Como vemos, el usuario podrá guardar en favoritos hasta tres noticias distintas. En el caso de que se añada una cuarta noticia, ésta sustituirá a la primera noticia guardada.

3.4. Pantallas de la aplicación Música

En esta sección se explicará el diseño tomado para la creación de las pantallas de la aplicación **Música** de la plataforma **Multi-App**. Como hasta ahora, comenzaremos mostrando la pantalla principal de la aplicación (Figura 9).

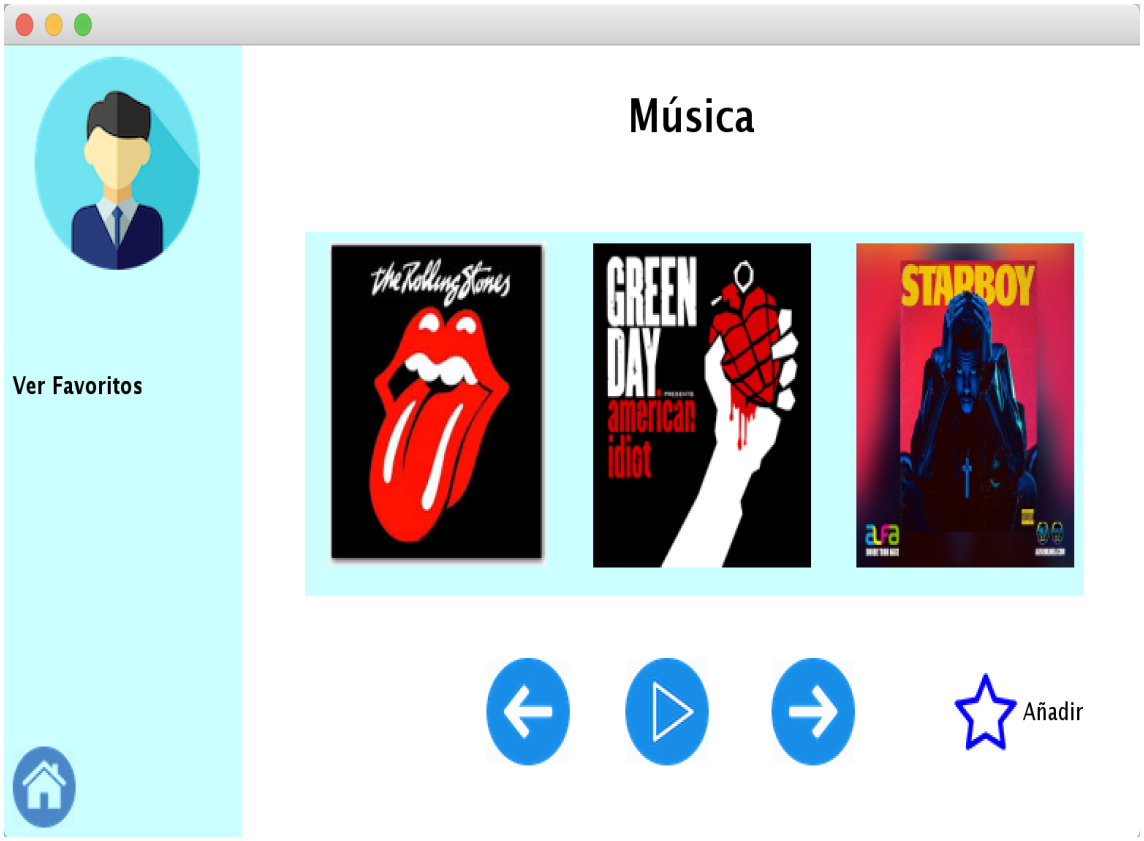

Figura 9: Pantalla principal de la aplicación **Música** de **Multi-App**

Como podemos observar, el usuario tendrá la posibilidad de ver la carátula de la canción que esté escuchando en el momento(carátula en el centro). En el caso de querer pasar a la siguiente canción, la canción actual pasará a la siguiente y la canción anterior pasar a ser la actual, quedando en una selección aleatoria la canción anterior. Lo mismo pero al contrario pasaría si el usuario quisiera reproducir la canción anterior. Para saber si se está reproduciendo una canción, basta con fijarse en el botón del centro. En el caso de que aparezca el botón de *Play*, el usuario sabrá que la canción está pausada. En el caso de que la canción se esté reproduciendo, el botón *Play* pasaría a tener la forma de un botón *Pause*. Finalmente, si el usuario quiere añadir a sus favoritos la canción que está escuchando actualmente(carátula del medio), bastaría con pulsar el botón de *Añadir*(botón con forma de estrella). Una vez que el usuario ha añadido alguna canción a sus favoritos, podrá irse a la sección de *Ver favoritos*. Esto le redireccionará a otra pantalla, la pantalla de favoritos(Figura 10).

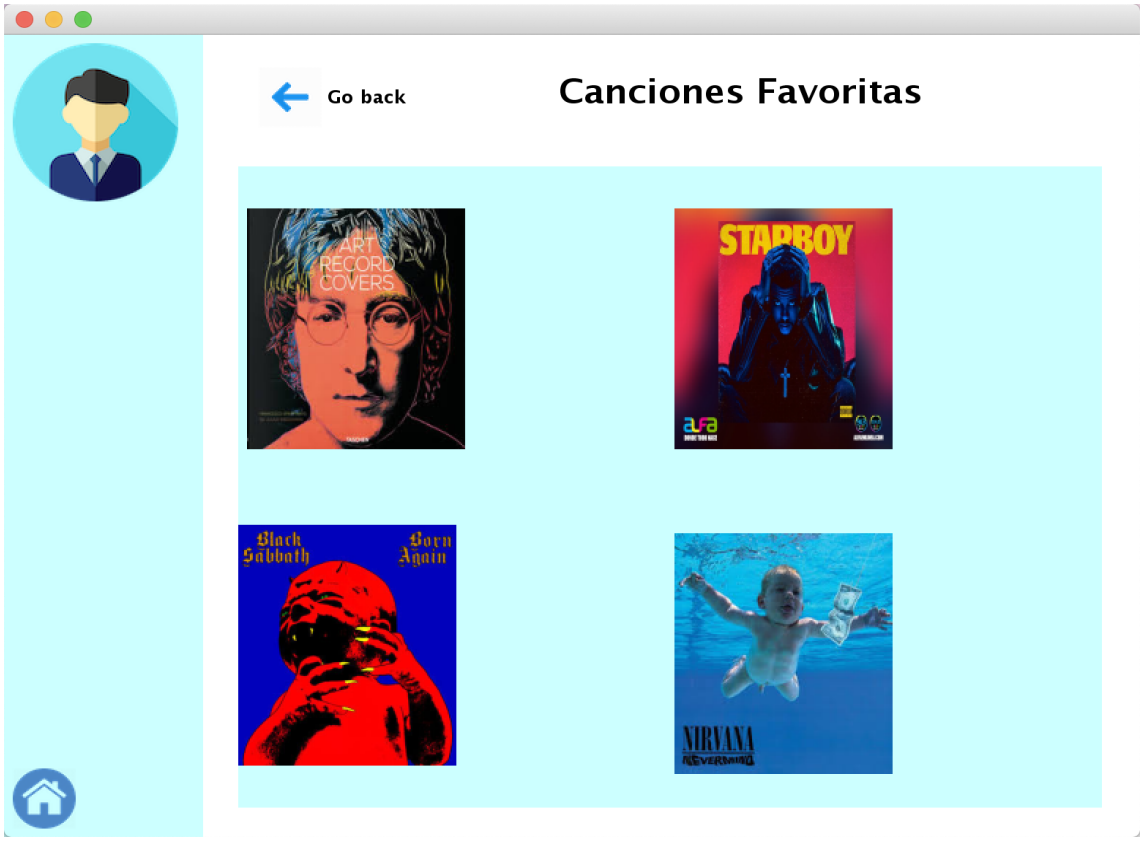

Figura 10: Pantalla *Favoritos* de la aplicación **Música** de **Multi-App**

Como vemos, esta pantalla tiene el mismo diseño que las anteriores, con la diferencia que no tiene ninguna acción en la barra lateral. Para volver a la pantalla principal de la aplicación, el usuario tendrá que pulsar en la flecha posicionada en la esquina superior izquierda.

4. Implementación

En esta sección explicaremos cómo hemos implementado las diferentes pantallas y sus acciones. Para una mejor comprensión, hemos decidido dividirlas según la pantalla.

4.1. Pantalla principal

Para esta pantalla, lo que hemos hecho ha sido crear cuatro *jLabel* y hacer que cuando el usuario pulse sobre alguna de ellas, haga visible la pantalla de esa aplicación y que establezca como no visible la pantalla principal. El Listing 1 muestra cómo hacer para mostrar la pantalla de la aplicación Correo.

```
1    private void jLabel2MouseReleased(java.awt.event.MouseEvent evt) {//GEN-FIRST:
     event_jLabel2MouseReleased
2        Mail correo = new Mail();
3        correo.setVisible(true);
4        this.setVisible(false);
5    }//GEN-LAST:event_jLabel2MouseReleased
```

Listing 1: Función jLabel2MouseRelease de la clase Home

Igualmente se hace para todas las demás aplicaciones, sólo bastaría con cambiar la clase de la variable *correo* por el nombre de la pantalla a la que queremos dirigirnos.

4.2. Aplicación Correo

Para esta aplicación hemos creado tres pantallas y una clase adicional para guardar las variables pertenecientes a un correo.

4.2.1. Clase MailVariables

Clase en la que hemos creado las variables pertenecientes a un mensaje. Junto con ello, hemos creado todos los observadores y modificadores de cada variable. El Listing 2 muestra las variables utilizadas.

```
1    public String fromMailText;
2    public String toMailText;
3    public String conceptText;
4    public String textText;
```

Listing 2: Variables públicas de la clase MailVariables

4.2.2. Pantalla principal

Para esta pantalla, como en la anterior, se ha creado una acción sobre el botón *Home* para que vuelva a la pantalla principal. Además, se añade otra función que nos haría visible la pantalla **EnviarCorreo** sin hacer desaparecer la pantalla principal. El Listing 3 muestra cómo lo hemos implementado.

```
1    private void enviarCorreoMouseReleased(java.awt.event.MouseEvent evt) {//GEN-
     FIRST:event_enviarCorreoMouseReleased
2        EnviarCorreo aux = new EnviarCorreo();
3        aux.setVisible(true);
4
5    }//GEN-LAST:event_enviarCorreoMouseReleased
```

Listing 3: Función enviarCorreoMouseReleased de la clase Correo

De igual manera, hemos creado otra acción sobre el botón *Ver* para que no haga visible la pantalla **VerCorreo** sin hacer desaparecer la pantalla principal.

Para simular la entrada de correos hemos creado una acción sobre el botón *Refresh* para que nos introduzca el correo leído de un fichero en el primer hueco libre que encuentre sobre los jLabels creados, en los Listings 4 y 5 se puede apreciar cómo hemos solucionado este problema.

```
1   private void refreshIconMouseReleased(java.awt.event.MouseEvent evt) {//GEN-
    FIRST:event_refreshIconMouseReleased
2       MailVariables mail = new MailVariables();
3       mail = readFile(mail);
4
5       String aux_from = from1.getText();
6
7       if(from1.getText() == ""){
8           from1.setText(mail.getFromText());
9       }else{
10          if(from2.getText() == ""){
11              from2.setText(mail.getFromText());
12          }else{
13              if(from3.getText() == ""){
14                  from3.setText(mail.getFromText());
15              }else{
16                  if(from4.getText() == ""){
17                      from4.setText(mail.getFromText());
18                  }else{
19                      from1.setText(mail.getFromText());
20                  }
21              }
22          }
23
24      }
```

Listing 4: Función refreshIconMouseRelease de la clase Correo

```
1   public MailVariables readFile(MailVariables correo){
2       String mail[] = new String[4];
3       try {
4               File file = new File("temp.txt");
5
6               int count = 0;
7
8               FileReader fileReader = new FileReader(file);
9               BufferedReader bufferedReader = new BufferedReader(fileReader);
10              StringBuffer stringBuffer = new StringBuffer();
11              String line;
12
13              while ((line = bufferedReader.readLine()) != null && count < 4) {
14                      stringBuffer.append(line);
15                      mail[count] = line;
16                      stringBuffer.append("\n");
17                      count++;
18              }
19              fileReader.close();
20          } catch (IOException e) {
21                  e.printStackTrace();
22          }
23
24      correo.setFromText(mail[0]);
25      correo.setToText(mail[1]);
26      correo.setConceptText(mail[2]);
27      correo.settextText(mail[3]);
28
29      return correo;
30  }
```

Listing 5: Función readFile de la clase Correo

Finalmente, hemos cuatro acciones más que se encargarán de resetear el texto de cada correo. Para ello, hemos utilizado la función que se muestra en el Listing 6.

```
1     private void deleteMouseReleased(java.awt.event.MouseEvent evt) {//GEN-FIRST:
      event_deleteMouseReleased
2         from1.setText("");
3     }//GEN-LAST:event_deleteMouseReleased
4
5     private void delete2MouseReleased(java.awt.event.MouseEvent evt) {//GEN-FIRST:
      event_delete2MouseReleased
6         from2.setText("");
7     }//GEN-LAST:event_delete2MouseReleased
8
9     private void delete3MouseReleased(java.awt.event.MouseEvent evt) {//GEN-FIRST:
      event_delete3MouseReleased
10        from3.setText("");
11    }//GEN-LAST:event_delete3MouseReleased
12
13    private void delete4MouseReleased(java.awt.event.MouseEvent evt) {//GEN-FIRST:
      event_delete4MouseReleased
14        from4.setText("");
15    }//GEN-LAST:event_delete4MouseReleased
```

Listing 6: Funciones delete de la clase Mail

4.2.3. Pantalla EnviarCorreo

Para esta pantalla hemos creado un objeto de la clase MailVariables que utilizaremos para guardar la información en un fichero. El Listing 7 nos muestra cómo hemos creado la variable y los Listing 8 y 9 nos muestran cómo almacenamos ese mensaje y lo escribimos en el fichero.

```
1     MailVariables correo = new MailVariables();
```

Listing 7: Variable MailVariable de la clase Enviar Correo

```
1     private void sendButtonMouseReleased(java.awt.event.MouseEvent evt) {//GEN-
      FIRST:event_sendButtonMouseReleased
2         correo.setFromText(fromMail.getText());
3         correo.setToText(toMail.getText()) ;
4         correo.setConceptText(conceptMail.getText());
5         correo.settextText(textMail.getText());
6
7         writeFile(correo);
8         this.setVisible(false);
9     }//GEN-LAST:event_sendButtonMouseReleased
10
11    /**
12     * Function that set Mail screen visible.
```

Listing 8: Función sendButtonMouseRelease de la clase EnviarCorreo

```
1     public void writeFile(MailVariables correo){
2         String fileName = "temp.txt";
3
4         try {
5             FileWriter fileWriter =
6                 new FileWriter(fileName);
7
8             BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);
9
10            bufferedWriter.write(correo.getFromText());
11            bufferedWriter.newLine();
12            bufferedWriter.write(correo.getToText());
13            bufferedWriter.newLine();
```

```
            bufferedWriter.write(correo.getConceptText());
            bufferedWriter.newLine();
            bufferedWriter.write(correo.gettextText());
            bufferedWriter.newLine();

            bufferedWriter.close();
        }
        catch(IOException ex) {
            System.out.println(
                "Error writing to file '"
                + fileName + "'");
        }
    }
```

Listing 9: Función writeFile de la clase EnviarCorreo

4.2.4. Pantalla VerCorreo

Esta pantalla lee el fichero con el correo y lo muestra por completo, no sólo la variable *fromMailText*. Para ello, hemos creado una función adicional para leer el fichero que es exactamente igual que la que se muestra en el Listing 5. A diferencia de la Pantalla EnviarCorreo, ahora leemos esas variables para mostrarlas por pantalla. El Listing 10 muestra cómo hemos abordado la solución.

```
    public void initializeMail(){
        MailVariables correo = new MailVariables();
        correo = readFile(correo);
        from.setText(correo.getFromText());
        concept.setText(correo.getConceptText());
        content.setText(correo.gettextText());

    }
```

Listing 10: Función initializeMail de la clase VerCorreo

4.3. Aplicación To-do

Esta aplicación sólo contiene una pantalla, con lo que todo ocurre en una misma clase. Primero, inicializamos todas las tareas a nulo, para ello, creamos una función como la que se muestra en el Listing 11.

```
    public void initializeList(){
        tarea1.setText("");
        tarea2.setText("");
        tarea3.setText("");
        tarea4.setText("");
        tarea5.setText("");
    }
```

Listing 11: Función initializeList de la clase ToDo

Seguidamente, para añadir las tareas hemos creado un componente jTextField en el cual recogeremos la tarea a añadir cuando pulsemos el botón de añadir. Para eso, hemos creado una función que introduce en el primer hueco libre que haya una nueva tarea, en el caso de no quedar ninguna tarea libre, se sobrescribe la primera tarea. Esta función se puede apreciar en el Listing 12.

```
    private void addButtonMouseReleased(java.awt.event.MouseEvent evt) {//GEN-FIRST
    :event_addButtonMouseReleased
        if(tarea1.getText() == ""){
            tarea1.setText(nuevaTarea.getText());
        }else{
            if(tarea2.getText() == ""){
                tarea2.setText(nuevaTarea.getText());
            }else{
```

```
                if(tarea3.getText() == ""){
                    tarea3.setText(nuevaTarea.getText());
                }else{
                    if(tarea4.getText() == ""){
                        tarea4.setText(nuevaTarea.getText());
                    }else{
                        if(tarea5.getText()==""){
                            tarea5.setText(nuevaTarea.getText());
                        }else{
                            tarea1.setText(nuevaTarea.getText());
                        }
                    }
                }
            }
        }
    }//GEN-LAST:event_addButtonMouseReleased
```

Listing 12: Función addButtonMouseReleased de la clase ToDo

Finalmente, creamos una serie de funciones para borrar alguna tarea. Para ello hemos creado tres jLabels para simular una papelera en cada una de las cinco tareas que se pueden mostrar. El Listing 13 muestra cómo lo hemos simulado.

```
    private void Borrar1MouseReleased(java.awt.event.MouseEvent evt) {//GEN-FIRST:
    event_Borrar1MouseReleased
        tarea1.setText("");
    }//GEN-LAST:event_Borrar1MouseReleased
```

Listing 13: Función Borrar1MouseReleased de la clase ToDo

4.4. Aplicación Noticias

Esta aplicación consta de dos pantallas, una principal y otra que muestra las noticias que el usuario ha marcado como favoritas. Primero, comenzaremos explicando cómo hemos implementado la pantalla principal, para así entender mejor la pantalla de canciones favoritas.

4.4.1. Pantalla principal

Para crear la clase de esta pantalla, primero hemos creado las variables necesarias tanto como para crear las noticias con títulos y comentarios, como para barajarlas aleatoriamente para sólo mostrar tres de ellas. En el Listing 14 se muestra cómo hemos inicializado estas variables.

```
    private String[] titles = new String[8];
    private String[] comments = new String[8];

    public static String[][] favorites = new String[3][2];
    public int favNumber = 0;
    int []cards = new int[4];
```

Listing 14: Variables privadas y públicas de la clase News

Seguidamente, para inicializar todas las noticias y mostrarlas en pantalla, hemos creado una función que crea una posición aleatoria para cada una de las tres noticias posibles. Al final de esta función, declaramos el texto de los tres jLabels que hemos creado para mostrar las tres noticias con título y comentario. Listing 15 muestra cómo hemos solucionado este problema.

```
    private void initializeNews(){
        titles[0] = "Camelia, una espa ola embarazada \n a la que Francia quiere
    expulsar por manifestarse\n con un escudo";
        titles[1] = "Posibles causas del accidente del avi n \nen Mosc : Malos
    pilotos, fallo mec nico\n y mal tiempo";
        titles[2] = "Nyiragongo, el volc n que revive una ciudad\n o el que la
    condena";
```

```
        titles[3] = "'Vengadores:\n Endgame' supera en dos\n semanas a 'Titanic' y amenaza a 'Avatar'.";
        titles[4] = "El Congreso cuenta desde hoy con un\n retrato oficial de Felipe VI que \nha costado 88.000 euros";
        titles[5] = "Nadal, pendiente \nde su est mago antes de su debut\n en Madrid";
        titles[6] = "La falta de confianza le\n est  pasando factura a Jorge\n Lorenzo";
        titles[7] = "Elecciones 2019: Ciudadanos\n no vetar  pactos con el PSOE en \nalgunas comunidades";

        comments[0] = "Esta espa ola de 34 a os\n que reside desde 2002 en Par s,\n fue detenida durante las manifestaciones.";
        comments[1] = "Son las causas que el Comit \n de Investigaciones ruso baraja";
        comments[2] = "Es el m s activo del mundo y\n mantiene en alerta a los mejores\n vulcan logos del mundo";
        comments[3] = "La pel cula de Marvel Studios ha\n recaudado 2.188 millones de d lares\n (1.955 millones de euros)";
        comments[4] = "La obra, de Hern n Cort s,\n preside la sala donde se re ne\n la Mesa de la C mara";
        comments[5] = "El tenista balear,\n aquejado de un virus, ya cancel  su \nsesi n de entrenamiento el domingo";
        comments[6] = "La victoria de Marc M rquez \nen Jerez, tras su ca da en la\n  ltima  carrera en Austin";
        comments[7] = "La formaci n 'naranja' podr a pactar\n gobiernos con los socialistas.";
        Random rand = new Random();
        int j = 0;
        for(int i = 0; i<4; i++){
            cards[i] = rand.nextInt(8);
            j = i;
        }

        TitleCard.setText(titles[cards[0]]);
        TitleCard1.setText(titles[cards[1]]);
        TitleCard2.setText(titles[cards[2]]);

        textCard.setText(comments[cards[0]]);
        textCard1.setText(comments[cards[1]]);
        textCard2.setText(comments[cards[2]]);
    }
```

Listing 15: Función initializeNews de la clase News

Además, hemos creado una función que devuelve la matriz de favoritos perteneciente a la clase Noticias. Listing 16 muestra cómo hemos creado esta función.

```
    public static String[][] getFavorites(){
        return favorites;
    }
```

Listing 16: Función getFavorites de la clase News

Finalmente, para que el usuario pueda guardar sus noticias favoritas, hemos creado un botón el cual guarda el título y el comentario de la noticia para después poder mostrarla en su lista de noticias favoritas. Listing 17 muestra cómo guardar esas variables.

```
    private void eyeIconMouseReleased(java.awt.event.MouseEvent evt) {//GEN-FIRST:event_eyeIconMouseReleased
        favorites[0][0] = TitleCard.getText();
        favorites[0][1] = TitleCard.getText();
        JOptionPane.showMessageDialog(null, "Noticia a adida a favoritos", "Mensaje de informaci n", JOptionPane.INFORMATION_MESSAGE);
    }//GEN-LAST:event_eyeIconMouseReleased
```

Listing 17: Función eyeIconMouseReleased de la clase News

4.4.2. Pantalla de favoritas

Esta pantalla se encarga de mostrar las tres noticias favoritas que el usuario haya guardado en la pantalla principal. Para ello, hemos creado una variable que guardará esas noticias, tal y como hemos hecho en la línea 4 de Listing 14. Además, hemos creado una función que inicializa esas noticias favoritas en cuando el usuario entra en esta pantalla. Para ello, hemos utilizado la función que muestra Listing 18.

```
1     public void initializeFavs(){
2         favorites = News.getFavorites();
3 
4         TitleCard.setText(favorites[0][0]);
5         TitleCard1.setText(favorites[1][0]);
6         TitleCard2.setText(favorites[2][0]);
7 
8         textCard.setText(favorites[0][1]);
9         textCard1.setText(favorites[1][1]);
10        textCard2.setText(favorites[2][1]);
11    }
```

Listing 18: Función initializeFavs de la clase FavoriteNews

Como vemos, utilizamos la función creada en Listing 16 para recoger las noticias que el usuario haya guardado en la pantalla principal. Estas noticias, como se puede apreciar, están en el orden que el usuario haya introducido.

4.5. Aplicación Música

Para la creación de esta aplicación, hemos utilizado dos pantallas, una principal y otra de favoritas. Esta aplicación simulará cómo funciona un reproductor de música como *Spotify*.

4.5.1. Pantalla principal

En esta pantalla hemos creado una clase llamada **Music**, la cual, contiene una serie de variables para el control de la reproducción de la música y el control de las canciones favoritas. Listing 19 muestra cuáles son estas variables.

```
1 public class Music extends javax.swing.JFrame {
2 
3     public boolean isplayed = false;
4     String canciones[] = new String[8];
5     Random rand = new Random();
6     int song = rand.nextInt(8);
7     static Icon favorites[] = new Icon[4];
```

Listing 19: Variables de la clase Music

Lo primero que haremos será como en el caso de Listing 16, pero en este caso devolveremos un vector de ocho posiciones de longitud. Seguidamente creamos una función para inicializar la carátula de las canciones para así mostrarlas en una de las tres posicione posibles. Listing 20 muestra esta función

```
1     public void initializeSongs(){
2 
3 
4         canciones[0] =  "/images/cancion1.png";
5         canciones[1] = "/images/cancion2.png";
6         canciones[2] = "/images/cancion3.png";
7         canciones[3] = "/images/cancion4.png";
8         canciones[4] =  "/images/cancion5.png";
9         canciones[5] = "/images/cancion6.png";
10        canciones[6] = "/images/cancion7.png";
11        canciones[7] = "/images/cancion8.png";
```

```
12
13          ImageIcon ii = new ImageIcon(getClass().getResource(canciones[song]));
14          Image image = ii.getImage().getScaledInstance(portada1.getWidth(), portada1
    .getHeight(), Image.SCALE_SMOOTH);
15          ii = new ImageIcon(image);
16          portada1.setIcon(ii);
17
18          song = rand.nextInt(8);
19          ImageIcon ii2 = new ImageIcon(getClass().getResource(canciones[song]));
20          Image image2 = ii2.getImage().getScaledInstance(portada2.getWidth(),
    portada2.getHeight(), Image.SCALE_SMOOTH);
21          ii2 = new ImageIcon(image2);
22          portada2.setIcon(ii2);
23
24          song = rand.nextInt(8);
25          ImageIcon ii3 = new ImageIcon(getClass().getResource(canciones[song]));
26          Image image3 = ii3.getImage().getScaledInstance(portada3.getWidth(),
    portada3.getHeight(), Image.SCALE_SMOOTH);
27          ii3 = new ImageIcon(image3);
28          portada3.setIcon(ii3);
29      }
```

Listing 20: Función initializeSongs de la clase Music

Como vemos, primero guardamos el path en el que están guardadas las carátulas de las canciones en un vector, seguidamente, crearemos una variable **ImageIcon** para leer las canciones de manera aleatoria. Después de eso, escalamos la imagen a las dimensiones que tienen los jLabels definidos como *portada*. Finalmente, sobrescribimos la variable *ii* transformando la variable *image* en **ImageIcon** para poder así modificar el icono de la portada y mostrar la carátula de la canción.

Para controlar si el usuario está escuchando música o está en modo pausa, hemos creado una acción sobre el botón *Play* para controlar esa situación. Listing 21 muestra cómo dependiendo de si está reproduciendo música o no, muestra un icono u otro.

```
1       private void playMouseReleased(java.awt.event.MouseEvent evt) {//GEN-FIRST:
    event_playMouseReleased
2           if(isplayed == false){
3               ImageIcon ii3 = new ImageIcon(getClass().getResource("/images/pause.png
    "));
4               Image image3 = ii3.getImage().getScaledInstance(play.getWidth(), play.
    getHeight(), Image.SCALE_SMOOTH);
5               ii3 = new ImageIcon(image3);
6               play.setIcon(ii3);
7               isplayed=true;
8           }else{
9               ImageIcon ii3 = new ImageIcon(getClass().getResource("/images/play.png"
    ));
10              Image image3 = ii3.getImage().getScaledInstance(play.getWidth(), play.
    getHeight(), Image.SCALE_SMOOTH);
11              ii3 = new ImageIcon(image3);
12              play.setIcon(ii3);
13              isplayed=false;
14          }
```

Listing 21: Función playMouseReleased de la clase Music

Si el usuario presiona el botón de siguiente canción, lo que haremos será mover todas las dos canciones de la derecha hacia la izquierda, y en la portada de la derecha mostrar otra carátula aleatoria. En el caso de que el usuario presione el botón de canción anterior, haremos los mismo pero en vez de mover las dos imágenes de la derecha, moveremos las dos imágenes de la izquierda a la derecha, mostrando una carátula aleatoria en la portada de la izquierda. Listing 22 y 23 muestran cómo hemos abordado la solución al problema anterior.

```
1     private void previousMouseReleased(java.awt.event.MouseEvent evt) {//GEN-FIRST:
      event_previousMouseReleased
2         portada1.setIcon(portada2.getIcon());
3         portada2.setIcon(portada3.getIcon());
4         song = rand.nextInt(8);
5         ImageIcon ii3 = new ImageIcon(getClass().getResource(canciones[song]));
6         Image image3 = ii3.getImage().getScaledInstance(portada3.getWidth(),
      portada3.getHeight(), Image.SCALE_SMOOTH);
7         ii3 = new ImageIcon(image3);
8         portada3.setIcon(ii3);
9     }//GEN-LAST:event_previousMouseReleased
```

Listing 22: Función previousMouseReleased de la clase Music

```
1     private void nextMouseReleased(java.awt.event.MouseEvent evt) {//GEN-FIRST:
      event_nextMouseReleased
2         portada3.setIcon(portada2.getIcon());
3         portada2.setIcon(portada1.getIcon());
4         song = rand.nextInt(8);
5         ImageIcon ii3 = new ImageIcon(getClass().getResource(canciones[song]));
6         Image image3 = ii3.getImage().getScaledInstance(portada1.getWidth(),
      portada1.getHeight(), Image.SCALE_SMOOTH);
7         ii3 = new ImageIcon(image3);
8         portada1.setIcon(ii3);
9     }//GEN-LAST:event_nextMouseReleased
```

Listing 23: Función nextMouseReleased de la clase Music

Finalmente, si el usuario quiere añadir la canción que se está reproduciendo, tendrá que presionar en el botón de *añadir* para por así ver sus canciones favoritas en la pantalla de favoritas. Listing 24 muestra cómo hemos realizado esto.

```
1     private void favoritosMouseReleased(java.awt.event.MouseEvent evt) {//GEN-FIRST
      :event_favoritosMouseReleased
2         if(numFav < 4){
3             favorites[numFav] = portada2.getIcon();
4             JOptionPane.showMessageDialog(null, "Imagen a adida a favoritos", "
      Mensaje de informaci n", JOptionPane.INFORMATION_MESSAGE);
5             numFav++;
6         }else{
7             JOptionPane.showMessageDialog(null, "No puedes guardar m s favoritos,
      borraremos el primero que a adiste", "Error!", JOptionPane.INFORMATION_MESSAGE
      );
8         }
9     }//GEN-LAST:event_favoritosMouseReleased
```

Listing 24: Función favoritosMouseReleased de la clase Music

4.5.2. Pantalla favoritas

Para esta pantalla hemos creado la clase **FavoriteMusic**, en la cual hemos creado variable que es un vector del tipo **Icon**, en el cual guardaremos las carátulas de las canciones favoritas del usuario. Seguidamente, hemos creado una función para inicializar los jLabels que hemos creado para mostrar las canciones favoritas. Listing 25 muestra cómo hemos realizado esta función.

```
1     public void initializeSongs(){
2         favorites = Music.getFavorites();
3
4         portada1.setIcon(favorites[0]);
5         portada2.setIcon(favorites[1]);
6         portada3.setIcon(favorites[2]);
7         portada4.setIcon(favorites[3]);
8     }
```

Listing 25: Función initializeSongs de la clase FavoriteMusic

5. Futuras mejoras

Con respecto a las futuras mejores que se realizarán sobre esta plataforma, hemos decidido ir explicandolas según cada aplicación.

1. **Correo**: Como futura mejora, podríamos crear una base de datos donde se guardaría cada uno de los correos del usuario, para así no tener que cambiar el nombre al correo que aparecerá en la bandeja de entrada. Además, como futura mejora hemos pensado en crear una bandeja de entrada y otra de salida.

2. **To-do**: Para esta aplicación queremos crear la posibilidad de poder crear carpetas tales como *Compras*, *Universidad*, etc. Para que así, el usuario pueda guardar las tareas según el tipo que sean.

3. **Noticias**: Lo ideal de esta aplicación sería contar con una base de datos real para poder así ir mostrando gran cantidad de noticias para que así el usuario nunca se quede sin conocer algo nuevo. Además, también crearíamos una variable más para la clase *News* a parte de los títulos y el comentario, que sería el contenido de la noticia, para que el usuario pueda leer más concretamente sobre alguna noticia que le haya interesado.

4. **Música**: Con respecto a una posible mejora para la aplicación de Música, hemos pensado igual que en la aplicación de Noticias, tener una base de datos más grande para que así el usuario tenga más variedad. Además, hemos pensado en crear una clase adicial en la cual guardaríamos el título y la carátula de la canción para que el usuario tenga más información sobre la canción que está escuchando.