

Documentación PDDI - Sebastián Silva

Índice

Guía de Arquitectura y Resumen Ejecutivo.....	5
Flujo de Datos Global.....	5
Paquete main y estructuración de archivos.....	5
Resumen.....	5
Main.....	5
Estructura de Archivos de Persistencia.....	6
1. users.txt.....	6
2. settings.txt.....	7
3. log.txt.....	7
Paquete windowContent (Ventanas).....	8
Resumen.....	8
ProgressBarWindow.....	8
LoginWindow.....	9
UserConfigWindow.....	10
MainWindow.....	10
AdminWindow.....	11
Paquete objects.....	12
Resumen.....	13
User.....	13
NewsContent.....	14
NewsItem.....	14
Paquete programLogic (funciones logicas).....	15
Resumen.....	15
WebReader.....	15
AutoEmailThread.....	17
EmailLogic.....	17
Paquete fileLogic (funciones rwx, I/O - txt).....	18
Resumen.....	18
WebLogic.....	19
ConfigLogic.....	20
UserLogic.....	21

Guía de Arquitectura y Resumen Ejecutivo

Esta documentación incluye resúmenes técnicos por paquete para facilitar una revisión ágil de la estructura del proyecto sin necesidad de examinar la totalidad del código fuente. Se detalla el propósito de cada componente, sus dependencias y su interacción con el sistema de persistencia.

Flujo de Datos Global

La aplicación sigue una arquitectura modular donde el paquete **windowContent** (Vista) solicita datos a **programLogic** y **fileLogic** (Controladores/Lógica). Estas capas procesan la información transformando los datos crudos de los archivos **.txt** y la web en objetos del paquete **objects** (**User**, **NewsItem**), que circulan por toda la aplicación.

Paquete main y estructuración de archivos

Resumen

La clase Main sirve únicamente como punto de entrada para arrancar la aplicación de forma segura. Se limita a activar el hilo de envío de correos en segundo plano y a lanzar la **ProgressBarWindow**, que funciona como la pantalla de carga inicial para el usuario.

La gestión de datos se basa en tres archivos de texto plano sin usar bases de datos. **users.txt** almacena las credenciales y el estado de los usuarios; **settings.txt** contiene toda la configuración crítica (URLs de noticias, datos del servidor SMTP y preferencias); y **log.txt** guarda el historial de actividad de cada consulta.

Main

Descripción

Clase principal y punto de entrada de la aplicación. Su única responsabilidad es iniciar el ciclo de vida de la aplicación de forma segura dentro del hilo de despacho de eventos de Swing (EDT).

Ubicación

`src/main/Main.java`

Método `main`

- `SwingUtilities.invokeLater(...)`: Asegura que la interfaz gráfica se cree en el hilo correcto, evitando problemas de concurrencia en Swing.
- Inicia el hilo en segundo plano `AutoEmailThread` para el envío programado de correos.
- Instancia `ProgressBarWindow`, que actúa como *Splash Screen* y carga inicial de la aplicación.

Dependencias

- `programLogic.AutoEmailThread`
- `windowContent.ProgressBarWindow`
- `javax.swing.SwingUtilities`

Estructura de Archivos de Persistencia

La aplicación utiliza archivos de texto plano (`.txt`) para almacenar toda la información, evitando el uso de bases de datos complejas. A continuación se detalla la estructura y formato de cada uno.

1. users.txt

Almacena la información de credenciales y estado de los usuarios.

Ubicación: `txtFiles/users.txt` **Formato:** Un usuario por línea. **Separador:** Punto y coma (;).

Campos

`Usuario;Contraseña;Email;Rol;Estado`

- **Usuario:** Nombre de usuario (Login).
- **Contraseña:** Contraseña en texto plano.
- **Email:** Dirección de correo electrónico.
- **Rol:** `ADMIN` o `USER`.
- **Estado:**
 - `0`: Usuario Nuevo (Debe pasar por configuración inicial).
 - `1`: Usuario Existente (Va directo al tablón de noticias).

Ejemplo

```
admin;1234;admin@test.com;ADMIN;1
pepe;abcd;pepe@gmail.com;USER;0
```

2. settings.txt

Archivo maestro de configuración. Contiene las URLs de las noticias, la configuración del servidor de correo y las preferencias de los usuarios. Se divide en **SECCIONES**.

Ubicación: `txtFiles/settings.txt`

Sección URLs (SECTION;URL)

Define las fuentes de noticias por categoría. **Formato:**

`CATEGORIA;URL1;SELECTOR1;URL2;SELECTOR2;URL3;SELECTOR3`

- **CATEGORIA:** Nombre de la categoría (ej: DEPORTES).
- **URL_N:** Dirección web de la fuente.
- **SELECTOR_N:** Selector CSS para extraer los titulares (`h2`, `.title`, etc.).

Sección Email (SECTION;FROM_EMAIL_CONFIGURATION)

Configuración del servidor SMTP para el envío de correos. **Formato:** `CLAVE;VALOR`

- `FROM_EMAIL_ADDRESS:` Email de origen.
- `FROM_EMAIL_PASSWORD:` Contraseña de aplicación o hash.
- `TIME:` Hora programada para el envío automático (HH:mm).

Sección Preferencias (SECTION;USERS_PREFERENCES)

Guarda qué categorías le interesan a cada usuario. **Formato:**

`Usuario;Categoria1;Categoria2;Categoria3...`

Ejemplo Estructural

SECTION;URL;

DEPORTES;<https://marca.com>;h2 a;<https://as.com>;h2 a...

...

SECTION;FROM_EMAIL_CONFIGURATION;

FROM_EMAIL_ADDRESS;bot@noticias.com;

FROM_EMAIL_PASSWORD;secreto123;

TIME;08:00

SECTION;USERS_PREFERENCES

pepe;DEPORTES;TECNOLOGIA

maria;ECONOMIA

3. log.txt

Historial de consultas de noticias guardadas por los usuarios.

Ubicación: `txtFiles/log.txt` **Formato:** Bloques de texto por consulta.

Estructura del Bloque

1. **Cabecera:** `LOG: [NombreUsuario] [Fecha Hora]`
2. **Cuerpo:** Lista de titulares guardados (uno por línea).
3. **Separador:** Línea en blanco entre bloques.

Ejemplo

LOG: pepe [2023-10-25 10:30:15]

DEPORTES: El equipo local gana el campeonato

TECNOLOGIA: Nueva IA revoluciona el mercado

LOG: admin [2023-10-25 11:00:00]

ECONOMIA: Sube el precio del aceite

Paquete windowContent (Ventanas)

Resumen

La aplicación arranca con la `ProgressBarWindow` cargando configuraciones antes de abrir el `LoginWindow`. El login actúa de filtro: si el usuario es nuevo, lo obliga a pasar primero por `UserConfigWindow` para elegir sus categorías de interés; si es un usuario normal, va directo a las noticias; y si es administrador, entra a su panel de gestión.

La `MainWindow` es el visor de noticias para el usuario. Lo más importante aquí es que descarga la información de internet en un hilo separado (`background thread`) para evitar que la ventana se congele o se ponga en blanco mientras carga. Solo cuando tiene los datos listos, actualiza la interfaz visualmente.

Por último, la `AdminWindow` sirve para crear o borrar usuarios y forzar el envío de emails manualmente. Usa un diseño de paneles superpuestos para no tener que abrir múltiples ventanas y, al igual que la principal, realiza el envío de correos en segundo plano para que la interfaz siga respondiendo mientras conecta con el servidor.

ProgressBarWindow

Descripción

Pantalla de carga inicial (*Splash Screen*). Se muestra mientras la aplicación lee los archivos de configuración y base de datos para dar feedback al usuario de que "algo está pasando".

Ubicación

`src/windowContent/ProgressBarWindow.java`

Características

- Extiende de `JWindow` en lugar de `JFrame`, por lo que **no tiene bordes ni botones de cerrar** (diseño limpio).
- Muestra el logo de la aplicación.
- Incluye una `JProgressBar`.

Lógica de Carga

Un hilo dedicado rellena la barra del 0 al 100%.

- Al llegar al **80%**, ejecuta la carga real de datos:
 - `UserLogic.readUsers()`
 - `UserLogic.readUserPreferences()`
 - `WebLogic.setNewsProperties()`
- Si la carga falla (archivos no encontrados), muestra un error fatal y cierra el programa.
- Al llegar al **100%**, se cierra a sí misma y abre la `LoginWindow`, cediendo el control al usuario.

LoginWindow

Descripción

La puerta de entrada a la aplicación. Gestiona la autenticación de usuarios contra la "base de datos" local (`UserLogic.usersList`).

Ubicación

`src/windowContent/LoginWindow.java`

Características de la Interfaz

- **Campos:** Usuario (`JTextField`) y Contraseña (`JPasswordField`).
- **Botón Ojo:** Un `JToggleButton` con lógica para alternar la visibilidad de la contraseña (`setEchoChar`).
- **Validaciones Visuales:** Etiquetas de error ocultas que aparecen si se intenta entrar con campos vacíos o credenciales incorrectas.

Lógica de Autenticación

El botón "Iniciar sesión" (o el Enter en los campos):

1. Recoge los textos.
2. Comprueba si están vacíos (`checkFieldsNullity`).
3. Itera sobre la lista cargada en memoria `UserLogic.usersList`.
4. Si encuentra coincidencia exacta de usuario y contraseña:

- Cierra el Login.
- **Enrutamiento:**
 - Si es Rol **ADMIN** -> Abre **AdminWindow**.
 - Si es Nuevo (**isNew()**) -> Abre **UserConfigWindow**.
 - Si es Usuario normal -> Abre **MainWindow**.
- 5. Si no encuentra coincidencia al terminar el bucle, muestra "Usuario o contraseña incorrectos".

Detalles Técnicos

- Usa coordenadas absolutas (Layout **null**) para posicionamiento exacto de elementos.
- Carga imágenes de iconos desde la carpeta **images/**.

UserConfigWindow

Descripción

Ventana de configuración inicial ("Onboarding") obligatoria para usuarios nuevos. Les obliga a elegir al menos una categoría de noticias antes de poder usar la aplicación.

Ubicación

src/windowContent/UserConfigWindow.java

Interfaz

- Presenta 6 **JCheckBox** correspondientes a las categorías disponibles (Economía, Deportes, etc.).
- Botón "Seleccionar" para confirmar.

Lógica de Negocio

1. Al pulsar confirmar, verifica que haya al menos 1 checkbox marcado.
2. Si no hay ninguno, muestra advertencia ("Seleccione al menos una categoría").
3. Si es válido:
 - Añade las cadenas de texto ("**DEPORTES**", "**TECNOLOGIA**", etc.) a la lista de preferencias del usuario.
 - Marca el usuario como **NO NUEVO** (**user.setNew(false)**).
 - Guarda los cambios en disco inmediatamente (**UserLogic.writeUserPreferences** y **rewriteUsersFile**).
 - Cierra esta ventana y abre **MainWindow**.

MainWindow

Descripción

La pantalla principal para el usuario final. Muestra el "Muro de Noticias" personalizado según los intereses del usuario.

Ubicación

`src/windowContent/MainWindow.java`

Carga Asíncrona de Noticias

Para evitar que la aplicación se congele ("pantalla blanca") al arrancar:

- El constructor inicia la interfaz vacía (Labels con "Cargando...").
- Lanza un `Thread` separado para llamar a `WebReader.getNews()`, que es una operación pesada de internet.
- Cuando las noticias llegan, usa `SwingUtilities.invokeLater` para actualizar los `JLabel` de la interfaz de forma segura.

Interfaz Gráfica

- **Grid de Noticias:** 18 `JLabel` preposicionados en 2 columnas.
- **Formato HTML:** Los textos de los labels usan `<html>...</html>` para permitir saltos de línea automáticos si el titular es muy largo.
- **Botón Guardar:** Permite volcar los titulares vistos al archivo `log.txt` usando `WebLogic.saveLog`.

Navegación

- Botón "Volver al login": Cierra sesión.
- Botón "Volver a Admin": Solo visible si el usuario logueado tiene rol `ADMIN` (para el modo de prueba).

AdminWindow

Descripción

Ventana de gestión para el rol de **ADMINISTRADOR**. Proporciona una interfaz gráfica ("Dashboard") desde donde se pueden controlar usuarios, probar funcionalidades y enviar correos masivos.

Ubicación

`src/windowContent/AdminWindow.java`

Estructura de la Interfaz

La ventana utiliza un sistema de **Paneles Superpuestos** (`JLayeredPane`) para simular navegación sin cambiar de ventana real.

- **Panel Menú (menuPane)**: El "Hub" central con 4 botones grandes.
- **Panel Crear Usuario (createUserPane)**: Formulario para registro de nuevos usuarios.
- **Panel Borrar Usuario (deleteUserPane)**: Interfaz simple para eliminar usuarios por nombre.
- **Panel Email (emailPane)**: Zona para enviar el boletín de noticias manualmente.

Funcionalidades Principales

Gestión de Usuarios

- **Crear**: Permite añadir usuarios validando:
 - Límite máximo de 10 usuarios.
 - Caracteres prohibidos (;) que romperían la persistencia.
 - El usuario se crea llamando a `UserLogic.createNewUser`.
- **Borrar**: Elimina usuarios validando:
 - Mínimo de 4 usuarios en el sistema para no quedar vacíos.
 - Llama a `UserLogic.deleteUser`.

Envío de Correos

- Botón "Enviar Noticias".
- Ejecuta el envío en un **Hilo Secundario (new Thread)** para no congelar la interfaz gráfica (EDT) mientras `JavaMail` conecta con el servidor SMTP.
- Muestra feedback visual en un `JLabel` de estado.
- Usa `JOptionPane` al finalizar para confirmar éxito o error.

Modo Prueba

- Botón "Probar noticias".
- Abre una instancia de `MainWindow` pero configurada con un usuario **Admin** temporal que tiene **TODAS** las categorías activadas.
- Sirve para que el admin verifique qué noticias se están extrayendo de cada fuente.

Eventos (Listeners)

- Los botones de navegación (`goTo...`) simplemente alternan la visibilidad de los paneles (`setVisible(true/false)`).
- Control estricto de cierre de ventana (`DO_NOTHING_ON_CLOSE`) con confirmación.

Paquete objects

Resumen

La clase User es la entidad principal que circula por toda la aplicación. Almacena los datos de identidad (usuario, contraseña, email) y el rol (Admin o User), pero lo más importante es que gestiona el estado del usuario mediante un flag (isNew) para controlar el acceso inicial y guarda una lista dinámica con sus categorías preferidas.

Por otro lado, NewsContent funciona puramente como un archivo de configuración para el scraping, sin lógica propia. Su trabajo es mapear cada categoría (ej: "Deportes") con hasta tres URLs distintas y sus correspondientes selectores CSS, sirviendo de guía para que el sistema sepa exactamente en qué webs entrar y qué elementos leer.

Finalmente, NewsItem es la unidad mínima de información, representando una noticia individual. Es un objeto simple que solo contiene la categoría, el titular y el enlace original; estos objetos se generan masivamente al leer las webs y son los que terminan mostrándose en la pantalla principal o enviándose en los correos.

User

Descripción

Clase principal para modelar la entidad Usuario. Almacena credenciales, rol, estado y preferencias personales. Es un objeto "vivo" que se pasa entre casi todas las ventanas y lógicas del programa.

Ubicación

`src/objects/User.java`

Atributos

- `String username`: Nombre de usuario (identificador único).
- `String password`: Contraseña (almacenada en texto plano en esta versión educativa).
- `String email`: Correo electrónico para recibir notificaciones.
- `String role`: Rol del usuario (`ADMIN` o `USER`).
- `boolean isNew`: Flag que indica si el usuario acaba de ser creado (para forzar configuración inicial).
- `ArrayList<String> preferencesList`: Lista de categorías de noticias que el usuario desea ver (ej: [`"DEPORTES"`, `"TECNOLOGIA"`]).

Métodos

- **Constructores**:
 - Completo: Incluyendo lista de preferencias.
 - Parcial: Sin preferencias iniciales (se inicializa vacía).
- **Getters y Setters**: Acceso estándar a todos los campos.

- **toString():** Representación textual del usuario y sus propiedades.

Notas de Diseño

La lista `preferencesList` es dinámica, permitiendo que un usuario tenga de 0 a N intereses.

NewsContent

Descripción

Clase de tipo *Bean* o *POJO* (Plain Old Java Object) que modela la configuración de una categoría de noticias. No contiene lógica de negocio, solo estructura de datos.

Diseñada para almacenar hasta 3 fuentes distintas para una misma categoría.

Ubicación

`src/objects/NewsContent.java`

Atributos

- `String category`: Nombre de la categoría (ej: "DEPORTES").
- `String url1, url2, url3`: URLs de las páginas web a scrapear.
- `String selector1, selector2, selector3`: Selectores CSS correspondientes a cada URL para extraer los titulares (`h2`, `h3` a, etc.).

Métodos

- **Constructor**: Inicializa todos los campos.
- **Getters y Setters**: Para acceso y modificación de los atributos.
- **toString()**: Representación en texto del objeto para depuración.

Uso

Es utilizada por `WebLogic` para cargar la configuración y por `WebReader` para saber dónde buscar las noticias.

NewsItem

Descripción

Clase *POJO* que representa una noticia individual. Es la unidad mínima de información que se muestra al usuario en la interfaz o se envía por correo.

Ubicación

`src/objects/NewsItem.java`

Atributos

- `private String category`: Categoría a la que pertenece la noticia (ej: "NACIONAL").
- `private String headline`: El titular de la noticia.
- `private String url`: Enlace directo a la noticia original.

Métodos

- **Constructor**: Inicializa categoría, titular y URL.
- **Getters y Setters**: Métodos estándar de acceso.
- **toString()**: Devuelve una representación en cadena del objeto, útil para logs y depuración.

Uso

Los objetos `NewsItem` son creados masivamente por `WebReader` y consumidos por `MainWindow` (para mostrarlos en etiquetas) y `AutoEmailThread` (para componer el cuerpo del correo).

Paquete programLogic (funciones logicas)

Resumen

El motor de extracción de datos es `WebReader`, que utiliza la librería `Jsoup` para conectarse a páginas web reales simulando un navegador Chrome (para evitar bloqueos) y descargar el HTML. Su función principal es iterar sobre las URLs configuradas, aplicar los selectores CSS para localizar titulares y filtrar el ruido mediante una "lista negra" de palabras prohibidas o textos demasiado cortos. El sistema es robusto ante fallos: si una URL específica da error, la captura individualmente y continúa con la siguiente para garantizar que el usuario siempre reciba contenido.

La automatización corre a cargo de `AutoEmailThread`, un hilo en segundo plano que mantiene un bucle infinito comprobando la hora del sistema cada 30 segundos. Cuando la hora actual coincide con la programada en la configuración, desencadena el proceso de envío masivo: recorre la lista de usuarios, descarga las noticias personalizadas de cada uno usando `WebReader` y construye el cuerpo del mensaje concatenando los titulares.

Finalmente, la infraestructura de comunicación reside en `EmailLogic`, una clase utilitaria que maneja el protocolo SMTP mediante la librería `JavaMail`. Se encarga de configurar las propiedades técnicas (SSL, timeouts de 5 segundos para evitar bloqueos), autenticar la sesión con las credenciales del sistema y despachar los correos en formato HTML. Está diseñada para capturar cualquier error de red sin detener la ejecución del programa principal.

WebReader

Descripción

El motor de *web scraping* de la aplicación. Utiliza la librería **Jsoup** para conectarse a páginas web reales, descargar su HTML y extraer los titulares basándose en selectores CSS configurados.

Ubicación

`src/programLogic/WebReader.java`

Métodos

```
public static ArrayList<NewsItem> getNews(ArrayList<String> prefs,
boolean isAdmin)
```

Función: Devuelve una lista de noticias filtradas por las preferencias del usuario.

Argumentos:

- **prefs:** Lista de categorías deseadas por el usuario.
- **isAdmin:** Booleano que modifica el límite de noticias (3 por categoría fija si es admin, dinámico si es usuario).

Lógica de Extracción:

1. Itera sobre las categorías definidas en `WebLogic.newsList`.
2. Filtra categorías no deseadas (si no es admin).
3. Itera sobre las 3 URLs configuradas para esa categoría.
4. **Conexión Jsoup:**
 - **User-Agent** real de Chrome para evitar bloqueos 403 (anti-bot).
 - **Timeout** de 5 segundos.
5. **Selección:** Usa `doc.select(selector)` para encontrar elementos HTML.
6. **Filtrado:**
 - Descarta textos cortos (< 10 caracteres).
 - Lista negra de palabras prohibidas ("Guerra de Ucrania", "Wordle", publicidad) para limpiar resultados.
7. Crea objetos **NewsItem** y los acumula en la lista de resultados.

Manejo de Errores

Captura excepciones genéricas (**Exception**) por cada URL individual. Si una web falla, imprime el error y continúa con la siguiente, asegurando que el usuario reciba al menos algunas noticias.

Dependencias

- `org.jsoup.Jsoup`
- `fileLogic.WebLogic`
- `objects.NewsItem`

AutoEmailThread

Descripción

Clase que extiende de `Thread` para ejecutarse en segundo plano (background). Su función es comprobar periódicamente si es la hora configurada para enviar el resumen de noticias por correo electrónico a todos los usuarios.

Ubicación

`src/programLogic/AutoEmailThread.java`

Ciclo de Vida (`run`)

El método `run()` ejecuta un bucle infinito (`while(true)`) que:

1. Lee la hora programada desde `ConfigLogic.get("TIME")`.
2. Si no hay hora, duerme 60 segundos.
3. Si hay hora, comprueba si la hora actual coincide con la programada Y si aún no se ha ejecutado hoy.
4. Si coincide, invoca `sendEmailsToAllUsers()`.
5. Duerme 30 segundos entre comprobaciones para no saturar la CPU.

Métodos Privados

`sendEmailsToAllUsers()`

Lógica de envío masivo:

- Itera sobre `UserLogic.usersList`.
- Para cada usuario con email válido:
 - Descarga sus noticias personalizadas con `WebReader.getNews`.
 - Construye el cuerpo del correo concatenando titulares (`String body += ...`).
 - Llama a `EmailLogic.sendEmail` para realizar el envío.
- Imprime logs en consola (`System.out/System.err`) sobre el progreso.

Dependencias

- `java.lang.Thread`
- `programLogic.EmailLogic`
- `programLogic.WebReader`
- `fileLogic.ConfigLogic`

EmailLogic

Descripción

Clase utilitaria estática encargada de la infraestructura de envío de correos electrónicos mediante el protocolo SMTP y la librería `javax.mail` (JavaMail).

Ubicación

`src/programLogic/EmailLogic.java`

Métodos

- `public static void sendEmail(String toEmail, String subject, String body)`

Función: Envía un correo HTML. **Parámetros:**

- `toEmail`: Destinatario.
- `subject`: Asunto del correo.
- `body`: Contenido del mensaje (acepta HTML). **Lógica:**
- Recupera configuración SMTP (Host, Puerto, Usuario, Pass) de `ConfigLogic`.
- Configura las `Properties` de JavaMail (autenticación, SSL, timeouts).
- Crea una sesión (`Session`) con un `Authenticator` anónimo.
- Compone el mensaje `MimeMessage`:
 - Cabeceras para UTF-8 y HTML.
 - Dirección de origen y destino.
- Realiza el envío con `Transport.send(msg)`.
- Captura errores y los imprime en `System.err` para no detener la ejecución del programa llamante.

Configuración Técnica

- **SSL**: Activado (`mail.smtp.ssl.enable`).
- **Timeouts**: 5 segundos para conexión y escritura, para evitar bloqueos largos.
- **Trust**: Confía en todos los certificados ().

Dependencias

- `javax.mail.*`
- `fileLogic.ConfigLogic`

Paquete fileLogic (funciones rwx, I/O - txt)

Resumen

WebLogic se encarga de la configuración del scraping y la auditoría. Lee del archivo `settings.txt` las URLs y selectores para saber de dónde sacar las noticias, y además registra

el historial de actividad de cada usuario guardando en log.txt los titulares que han consultado.

ConfigLogic funciona como una memoria caché para la configuración global. Carga datos fijos, como las credenciales del servidor de correo (SMTP) y la hora programada para envíos, y los mantiene en un mapa en memoria (HashMap) para evitar leer el disco constantemente y mejorar el rendimiento.

UserLogic es el gestor de la "base de datos". Mantiene la lista de usuarios sincronizada entre la memoria y los archivos de texto. Su trabajo es leer credenciales, guardar las preferencias de categorías de cada usuario y gestionar las altas y bajas, reescribiendo los archivos .txt cuando hay cambios para asegurar la persistencia.

WebLogic

Descripción

WebLogic gestiona la lógica relacionada con la configuración de las fuentes de noticias y el registro de actividad (logs). Es responsable de cargar qué URLs y selectores CSS se deben usar para cada categoría desde `settings.txt`.

Ubicación

- `src/fileLogic/WebLogic.java`

Atributos

- `public static ArrayList<NewsContent> newsList`: Lista que almacena objetos `NewsContent`, donde cada objeto representa una categoría y sus 3 fuentes asociadas.

Métodos

`public static void setNewsProperties()`

Función: Lee `settings.txt` para configurar el scraper de noticias. **Detalles:**

- Busca la sección `SECTION;URL`.
- Parsea líneas con formato complejo: `CATEGORIA;URL1;SELECTOR1;URL2;....`
- Crea objetos `NewsContent` y los añade a `newsList`.
- Esto permite que el scraping sea dinámico y configurable sin tocar código Java.

```
public static void saveLog(String username, ArrayList<String>
headlines, String timeStamp)
```

Función: Guarda un historial de consultas en `txtFiles/log.txt`. **Detalles:**

- Abre el archivo en modo *append* (`true`).
- Escribe una cabecera con el usuario y la hora.
- Lista todos los titulares consultados.
- Útil para auditoría o historial de usuario.
-

```
public static String checkNullity(...)
```

Función: Validación básica de campos (aunque su uso es limitado en la versión actual).

Dependencias

- `objects.NewsContent`
- `java.io`

ConfigLogic

Descripción

La clase `ConfigLogic` es la encargada de gestionar la configuración global de la aplicación. Actúa como una capa de abstracción sobre el archivo `settings.txt`, cargando y ofreciendo acceso a claves de configuración como los datos del servidor SMTP para el envío de correos o la hora programada para el envío automático.

Mantiene un mapa estático (`configData`) en memoria para evitar lecturas constantes al disco.

Ubicación

```
src/fileLogic/ConfigLogic.java
```

Atributos

- `private static final String CONFIG_FILE`: Ruta al archivo de configuración (obsoleto, ahora usa `settings.txt`).
- `private static Map<String, String> configData`: Estructura de datos clave-valor (`HashMap`) donde se almacena toda la configuración cargada en memoria.

Bloque Estático

Al cargar la clase, se ejecuta un bloque estático que:

1. Establece valores por defecto para SMTP (`smtp.gmail.com`, puerto `465`).
2. Llama al método `loadConfig()` para poblar el mapa con los datos del archivo.

Métodos

```
private static void loadConfig()
```

Función: Lee el archivo `txtFiles/settings.txt` y parsea su contenido. **Lógica:**

- Verifica si el archivo existe. Si no, imprime un error.
- Lee línea por línea ignorando líneas vacías o cabeceras de sección (`SECTION;`).
- Divide cada línea por punto y coma (;).
- Si detecta configuraciones de URL (más de 2 partes), las guarda con el prefijo `URL_`.
- Detecta y limpia la configuración de hora (`TIME`) si tiene puntos al final.
- Mapea claves específicas como `FROM_EMAIL_ADDRESS` a `SMTP_USER` y `FROM_EMAIL_PASSWORD` a `SMTP_PASS` para uso interno de JavaMail.
- Captura excepciones `IOException`.

```
public static String get(String key)
```

Función: Devuelve el valor de configuración asociado a una clave. **Parámetros:**

- `key` (String): La clave a buscar (ej: "SMTP_USER", "TIME"). **Retorno:**
- `String`: El valor configurado, o una cadena vacía "" si la clave no existe.

Dependencias

- `java.io` (Manejo de archivos)
- `java.util.HashMap / Map` (Estructura de datos)

UserLogic

Descripción

`UserLogic` es la clase central para la gestión de usuarios y persistencia de datos. Se encarga de leer, escribir, crear y eliminar usuarios del archivo `users.txt`, así como de gestionar sus preferencias en `settings.txt`. Mantiene una lista en memoria (`usersList`) que actúa como la fuente de verdad durante la ejecución del programa.

Ubicación

```
src/fileLogic/UserLogic.java
```

Atributos

- `static public ArrayList<User> usersList`: Lista estática que contiene todos los objetos `User` cargados en la aplicación.

Métodos Principales

```
public static void readUsers()
```

Función: Carga los usuarios desde `users.txt` a la lista en memoria. **Detalles:**

- Limpia `usersList` para evitar duplicados.
- Parsea cada línea separada por `;` (nombre, contraseña, email, rol, estado).
- Maneja la robustez de datos con `checkNullity` y `checkRole`.
- Detecta si un usuario es "nuevo" (estado `0`) o existente (`1`).

```
public static void readUserPreferences()
```

Función: Asocia las categorías de noticias preferidas a cada usuario leyendo `settings.txt`. **Detalles:**

- Busca la sección `SECTION;USERS_PREFERENCES`.
- Lee formato `usuario;CAT1;CAT2...`
- Actualiza la lista de preferencias de cada objeto `User` en memoria.
- Actualiza el estado `isNew` del usuario basándose en si tiene preferencias o no (si no es admin).

```
public static void writeUserPreferences(User user)
```

Función: Guarda las preferencias de todos los usuarios en `settings.txt`. **Detalles:**

- **Preservación:** Lee primero el archivo y guarda en memoria todo lo que NO sea la sección de usuarios (configuración del admin) para no perderla.
- **Limpieza:** Elimina líneas vacías al final para mantener el archivo limpio.
- **Escritura:** Reescribe la configuración preservada, añade la cabecera `SECTION;USERS_PREFERENCES` y vuelca la lista de usuarios con sus categorías.

```
public static void createNewUser(...)
```

Función: Registra un nuevo usuario en el sistema. **Detalles:**

- Verifica si el nombre de usuario ya existe.
- Crea el objeto `User`, lo añade a la lista, lo anexa al final de `users.txt` (`appendUserToFile`) y actualiza las preferencias.

```
public static void deleteUser(String username)
```

Función: Elimina un usuario del sistema. **Detalles:**

- Busca y elimina el usuario de la lista en memoria.
- Reescribe completamente `users.txt` (`rewriteUsersFile`) y `settings.txt` (`writeUserPreferences`) para reflejar los cambios.

```
public static void rewriteUsersFile()
```

Función: Sobrescribe `users.txt` con el estado actual de `usersList`. **Uso:** Se llama tras borrar usuarios o modificar estados masivamente.

Métodos Auxiliares

- `appendUserToFile(User u)`: Añade un usuario al final del archivo sin sobrescribir.
- `checkState(int)`: Convierte el entero del archivo a booleano (`isNew`).
- `checkRole(String)`: Valida y normaliza el rol (ADMIN/USER).
- `checkNullity(String, int)`: Repara campos vacíos o nulos asignando valores por defecto.

Dependencias

- `objects.User`
- `javax.swing.JOptionPane` (para feedback visual)
- `java.io` (Lectura/Escritura)