UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

PROYECTOS DE PROGRAMACIÓN

-APLICACIÓN DE GESTIÓN DE TECLADOS-

Santiago Cervera – santiago.cervera.perez

Nicolas Longueira – nicolas.longueira

Enric Teixidó – enric.teixido

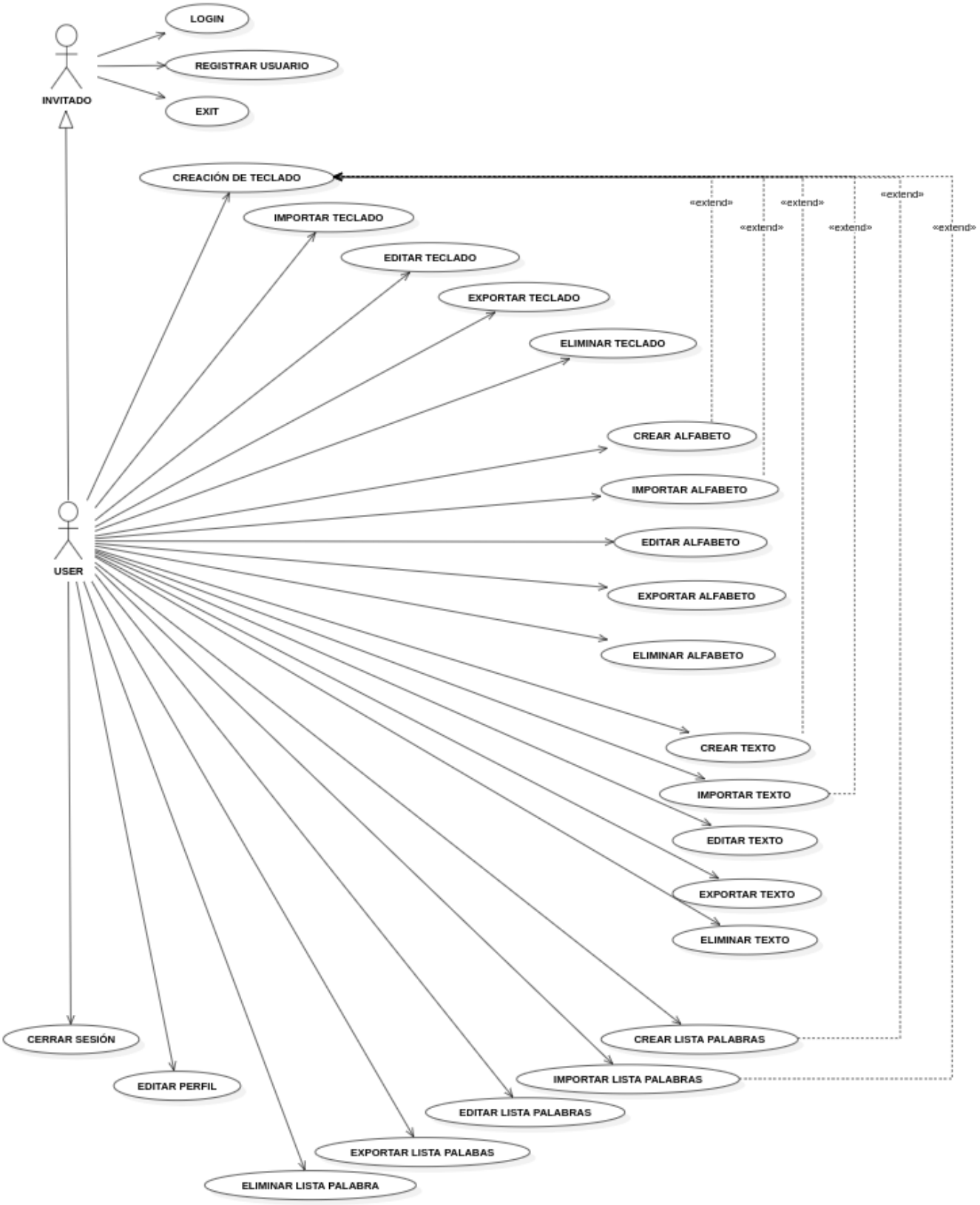
Alejandra Traveria – alejandra.traveria.marti

Grupo 41.1 : 1ª Entrega – Versión 3.0.

ÍNDICE

1. CASOS DE USO

1.1. Diagrama de casos de uso



1.2. Documentación casos de uso

1.2.1. Casos de uso invitado

Título: Login

Actor: Usuario Invitado

Descripción: Un usuario no autorizado inicia sesión y se convierte en un usuario autorizado.

Requisitos:

1. El usuario no autorizado debe estar registrado.

Comportamiento:

1. El usuario no autorizado ingresa su username.
    a. El nombre de usuario no debe contener más de 20 caracteres ni menos de 3.
    b. Los caracteres deben de estar comprendidos entre el 33-126 de la tabla ASCII. Excluyendo el 34 y 44.
2. El usuario no autorizado introduce su contraseña de usuario autorizado.
    a. La contraseña no debe contener más de 20 caracteres ni menos de 3.
    b. Los caracteres deben estar comprendidos entre el 32-126 de la tabla ASCII. Excluyendo el 34 y 44.
3. El sistema valida los datos ingresados e inicia sesión convirtiendo al usuario no autorizado en autorizado.

Errores y cursos alternativos:

1. Nombre de usuario incorrecto: El nombre de usuario proporcionado no cumple con el formato especificado de longitud o contiene algún carácter invalido El sistema lanza un mensaje de error y solicita un nuevo nombre de usuario o da la opción de cancelar el proceso de registro.
2. Contraseña incorrecta: La contraseña proporcionada no cumple con el formato especificado de longitud o contiene algún carácter invalido. El sistema lanza un mensaje de error y solicita una nueva contraseña o da la opción de cancelar el proceso de registro.
3. Usuario no existe: El nombre de usuario no coincide con ningún usuario registrado. El sistema notificará el error.
4. Contraseña no coincide: La contraseña dada no coincide con la registrada para el nombre de usuario dado. El sistema notificará al usuario del error.

Título: Registrar usuario

Actor: Usuario Invitado

Descripción: El usuario invitado debe poder registrarse como usuario para poder guardar y gestionar los diversos teclados generados, alfabetos, textos y listas de palabras.

Requisitos:

1. El usuario debe proporcionar una cuenta de correo electrónico válida
2. El usuario debe proporcionar un nombre de usuario
    a. El nombre de usuario no debe contener más de 20 caracteres ni menos de 3.
    b. Los caracteres deben de estar comprendidos entre el 33-126 de la tabla ASCII. Excluyendo el 34 y 44.
    c. El nombre de usuario tiene que ser único

3. El usuario debe proporcionar una contraseña
    a. La contraseña no debe contener más de 20 caracteres ni menos de 3.
    b. Los caracteres deben estar comprendidos entre el 32-126 de la tabla ASCII. Excluyendo el 34 y 44.

Comportamiento:

1. El sistema solicita al usuario que le facilite los requisitos anteriormente mencionados
2. Una vez proporcionados los datos el sistema valida que los datos sean correctos
3. El sistema crea un nuevo usuario

Errores y cursos alternativos:

5. Dirección de correo electrónico ya registrada: La dirección de correo electrónico ya está registrada con otro nombre de usuario. El sistema cancela el proceso de registro.
6. Dirección de correo electrónico inválida: La dirección de correo electrónico no ha sido validada o no existe. El sistema lanza un mensaje de error y solicita un correo electrónico alternativo o da la opción de cancelar el proceso de registro.
7. Nombre de usuario ya existente: El nombre de usuario proporcionado ya está registrado. El sistema lanza un mensaje de error y solicita un nuevo nombre de usuario o da la opción de cancelar el proceso.
8. Nombre de usuario incorrecto: El nombre de usuario proporcionado no cumple con el formato especificado de longitud o contiene algún carácter invalido El sistema lanza un mensaje de error y solicita un nuevo nombre de usuario o da la opción de cancelar el proceso de registro.
9. Contraseña incorrecta: La contraseña proporcionada no cumple con el formato especificado de longitud o contiene algún carácter invalido. El sistema lanza un mensaje de error y solicita una nueva contraseña o da la opción de cancelar el proceso de registro.

Título: Exit

Actor: Usuario Invitado

Descripción: El usuario no registrado que no desee registrarse puede cerrar la aplicación en cualquier momento.

Requisitos:

1. La aplicación tiene que estar operativa.

Comportamiento:

1. El usuario decide cerrar la aplicación como usuario invitado.
2. La aplicación se cierra y el usuario es devuelto a la pantalla de inicio o al sistema operativo.

Errores y Cursos Alternativos:

1. Cambios no guardados: El usuario ha realizado alguna acción que requiere guardar datos o progreso antes de cerrar la aplicación. El sistema lanza un mensaje de confirmación para asegurarse de que desea cerrar sin guardar.
2. Error en el cierre de sesión: En caso de un mal funcionamiento de la aplicación o un error crítico. El sistema lanza un mensaje de error al usuario antes de cerrar la aplicación de manera inesperada.

1.2.2. Casos de uso usuario

Título: Crear teclado

Actor: Usuario

Descripción: El usuario puede utilizar el sistema para diseñar una layout de teclado optimizado.

Requisitos:

1. El usuario tiene que crear/seleccionar/importar un alfabeto.
    a. El alfabeto no debe contener más de 25 caracteres.
    b. El alfabeto importado debe corresponder con el formato exigido (.csv).
2. El usuario tiene que crear/seleccionar/importar un texto o una lista de palabras con frecuencias.
    a. El texto no debe contener más de 1024 palabras ni más de 10240 carácteres.
    b. La suma de las frecuencias de la lista de palabras con frecuencia no puede ser superior a 1024.
    c. El texto o la lista de palabras con frecuencia importada deben corresponder con el formato exigido (.csv).
    d. El texto o la lista de palabras con frecuencia debe estar compuesto por caracteres del alfabeto.
3. El usuario debe proporcionar un nombre de teclado.
    a. El nombre del teclado no debe contener más de 20 caracteres ni menos de 4.
    b. Los caracteres deben de estar comprendidos entre el 33-126 de la tabla ASCII. Excluyendo el 34 y 44..
4. El usuario debe tener menos de 10 teclados guardados.

Comportamiento:

1. El sistema solicita al usuario que le facilite los requisitos anteriormente mencionados.
2. Una vez proporcionados los parámetros crea una distribución de teclado mediante el algoritmo QAP o (algoritmo que decidamos nosotros).
3. El sistema solicitará un nombre para guardar el teclado.
4. El sistema validará los datos y guardará el teclado.

Errores y cursos alternativos:

1. Alfabeto con exceso de caracteres: El número de carácteres introducido/importado por el usuario es mayor que el máximo permitido. No se permite la introducción de más carácteres una vez alcanzado el límite / se cancela la creación.
2. Alfabeto con formato incorrecto: El alfabeto importado no corresponde al formato exigido. Se cancela la creación.
3. Texto o lista de palabras con frecuencias con formato incorrecto): El texto o la lista de palabras con frecuencia importada no corresponde al formato exigido. Se cancela la creación.
4. Alfabeto y, texto o lista de palabras con frecuencias incompatibles: El texto o la lista de palabras con frecuencias insertada contiene símbolos no incluidos dentro del alfabeto usado. El sistema notificará y le dará la opción de incluir dichos símbolos del alfabeto, cambiar el alfabeto, cambiar el texto o la lista de palabras con frecuencias o cancelar la acción.
5. Teclado con nombre repetido: El sistema ya tiene registrado un teclado con el nombre de teclado proporcionado. El sistema lanza un mensaje de error y solicita un nuevo nombre de teclado o da la opción de cancelar el proceso de creación.
6. Nombre del teclado invalido: El nombre de teclado proporcionado no cumple con el formato especificado de longitud o contiene algún carácter invalido El sistema lanza un mensaje de error y solicita un nuevo nombre de teclado o da la opción de cancelar el proceso de creación.
7. Exceso de teclados: El usuario está intentando crear más teclados que el límite establecido. El sistema notificará y le dará la opción de eliminar otro teclado o cancelar la acción.

Título: Eliminar teclado

Actor: Usuario

Descripción: El usuario elimina un teclado almacenado en el sistema.

Requisitos:

1. El usuario tiene que tener teclados guardados.

Comportamiento:

1. El usuario ingresara el nombre del teclado que desea eliminar.
2. El sistema valida que el teclado existe.
3. El sistema eliminará el teclado con el nombre ingresado.

Errores y cursos alternativos:

1. Teclado no existente: El teclado identificado por el nombre ingresado no existe. El sistema notificará al usuario, este le volverá a exigir un nombre de teclado existente o la opción de cancelar la acción.

Título: Exportar teclado

Actor: Usuario

Descripción: El usuario exporta un teclado almacenado en el sistema.

Requisitos:

1. El usuario tiene que tener teclado guardados.
2. (Memoria del disco duro / ubicación / permisos /)

Comportamiento:

1. El usuario ingresa el nombre del teclado que desea exportar.
2. El usuario ingresa el nombre que tendrá el archivo exportado.
3. El usuario establece la ubicación en la desea guardar el teclado.
4. El sistema crea un archivo con el formato compatible en la ubicación anteriormente establecida.

Errores y cursos alternativos:

1. Almacenamiento (No hay espacio): El archivo no puede generarse porque no existe espacio de almacenamiento suficiente. El sistema notifica y cancela la operación.
2. Ubicación (No privilegios): El archivos no puede generarse en la ubicación establecida por falta de permisos. El sistema notifica, da la opción de cambiar de ubicación o cancelar la acción.
3. Archivo (Archivo ya existente): Ya existe un archivo con el mismo nombre, en la misma ubicación establecida. El sistema lo notificará y dará la opción de sobreescribir el archivo, cambiar el nombre, cambiar la ubicación o cancelar la acción.

Título: Importar teclado

Actor: Usuario

Descripción: El usuario importa un teclado con un formato compatible

Requisitos:

1. El formato del archivo tiene que ser compatible con el exigido.
2. El nombre del nuevo teclado debe ser distinto a los ya existentes.
3. El usuario debe tener menos de 10 teclados guardados.

Comportamiento:

1. El usuario ingresa la ubicación del archivo que desea importar.
2. El usuario ingresa el nombre con el que desea guardar el nuevo teclado en el sistema.
3. El sistema valida que el archivo sea del formato exigido y sea compatible.
4. El sistema agrega el teclado a la lista de teclados del usuario correspondiente.

Errores y cursos alternativos:

1. Teclado (Ya existe): Ya existe en el sistema un teclado con el nombre asignado al nuevo teclado. El sistema lo notificará y dará la opción de cambiar el nombre del teclado nuevo, de cambiar el nombre del teclado que ya existía en el sistema o cancelar la acción.
2. Teclado (Archivo no compatible): El archivo importado no cumple con el formato exigido o es incompatible con el sistema. El sistema lo notificará y dará la opción de cambiar de ubicación o cancelar la acción.
3. Ubicación (No privilegios): El sistema no puede acceder a la ubicación determinada. El sistema notificará y dará la opción de cambiar de ubicación o cancelar la acción.
4. Teclados (Exceso de teclados): El usuario está intentando importar más teclados que el límite establecido. El sistema notificará y le dará la opción de eliminar otro teclado o cancelar la acción.

Título: Editar teclado

Actor: Usuario

Descripción: El usuario edita un teclado almacenado en el sistema.

Requisitos:

1. El usuario tiene que tener teclado guardados.

Comportamiento:

1. El usuario selecciona el nombre del teclado que desea editar.
2. El usuario selecciona qué características del teclado desea modificar.
3. El sistema validará los datos y guardará el teclado con las modificaciones que haya realizado el usuario.

Errores y cursos alternativos:

1. Alfabeto con exceso de caracteres: El número de carácteres posterior a la modificación del usuario es mayor que el máximo permitido. No se permite la introducción de más carácteres una vez alcanzado el límite / se cancela la modificación.
2. Teclado con nombre repetido: El sistema ya tiene registrado un teclado con el nombre de teclado modificado. El sistema lanza un mensaje de error y solicita un nuevo nombre de teclado o da la opción de cancelar el proceso de modificación.
3. Nombre del teclado invalido: El nombre de teclado modificado no cumple con el formato especificado de longitud o contiene algún carácter invalido El sistema lanza un mensaje de error y solicita un nuevo nombre de teclado o da la opción de cancelar el proceso de modificación.

4. Alfabeto y, texto o lista de palabras con frecuencias incompatibles: El texto o la lista de palabras con frecuencias insertada contiene símbolos no incluidos dentro del alfabeto usado. El sistema notificará y le dará la opción de incluir dichos símbolos del alfabeto, cambiar el alfabeto, cambiar el texto o la lista de palabras con frecuencias o cancelar la acción.

Título: Crear alfabeto

Actor: Usuario

Descripción: Los usuarios pueden agregar nuevos alfabetos en el sistema

Requisitos:

1. El alfabeto no debe contener más de 25 caracteres ni menos de 2.
2. El usuario debe proporcionar un nombre al alfabeto.
    a. El nombre del alfabeto no debe contener más de 20 caracteres ni menos de 4.
    b. Los caracteres deben de estar comprendidos entre el 33-126 de la tabla ASCII. Excluyendo el 34 y 44.
3. El usuario no tiene más de 10 alfabetos guardados.

Comportamiento:

1. El sistema solicita al usuario que inserte un alfabeto.
2. El sistema solicitará un nombre para guardar el teclado.
3. El sistema validará los datos y guardará el teclado.

Errores y cursos alternativos:

1. Alfabeto con exceso de caracteres: el alfabeto proporcionado tiene más caracteres de los permitidos. El sistema lanza un mensaje de error y solicita que modifiques el alfabeto o da la opción de cancelar el proceso de creación.
2. Alfabeto ya existente: El nombre proporcionado ya existe en el sistema. El sistema lanza un mensaje de error y solicita un nuevo nombre de alfabeto o da la opción de cancelar el proceso de creación.
3. Nombre incorrecto: El nombre proporcionado no cumple con el formato especificado de longitud o contiene algún carácter invalido El sistema lanza un mensaje de error y solicita un nuevo nombre de alfabeto o da la opción de cancelar el proceso de creación.

Título: Eliminar alfabeto

Actor: Usuario

Descripción: Los usuarios pueden eliminar alfabetos del sistema.

Requisitos:

1. El usuario debe proporcionar el nombre del alfabeto que desea eliminar.
2. El alfabeto con el nombre proporcionado debe existir en el sistema.

Comportamiento:

1. El sistema solicita el nombre del alfabeto que el usuario desea eliminar.
2. El sistema valida que el alfabeto exista.

3. El sistema elimina el alfabeto.

Errores y cursos alternativos:

1. Alfabeto inexistente: El alfabeto identificado por el nombre proporcionado no existe. El sistema lanza un mensaje de error y solicita al usuario el nombre de un alfabeto existente o la opción de cancelar la acción.

Título: Exportar alfabeto

Actor: Usuario

Descripción: Los usuarios pueden exportar en un archivo .csv los alfabetos guardados.

Requisitos:

1. El usuario debe tener alfabetos guardados.
2. El usuario debe proporcionar un nombre al archivo que se va a exportar.
    a. El nombre del alfabeto no debe contener más de 20 caracteres ni menos de 4.
    b. Los caracteres deben de estar comprendidos entre el 33-126 de la tabla ASCII. Excluyendo el 34 y 44.
3. El usuario tiene memoria disponible para exportar el documento.
4. (ubi/privilegios)

Comportamiento:

1. El sistema solicita al usuario que ingrese el nombre del teclado que desea exportar.
2. El sistema solicita al usuario que ingrese el nombre que tendrá el archivo exportado.
3. El usuario establece la ubicación en la que desea guardar el alfabeto exportado.
4. El sistema crea un archivo .csv en la ubicación establecida.

Errores y cursos alternativos:

1. Alfabeto inexistente: el nombre proporcionado no pertenece a ningún alfabeto del sistema. El sistema lanza un mensaje de error y solicita que le proporciones otro nombre o da la opción de cancelar el proceso de exportación.
2. Nombre del archivo inválido: el nombre proporcionado para el archivo a exportar no cumple el formato especificado de longitud o contiene un caracter invalido. El sistema lanza un mensaje de error y solicita que le proporciones otro nombre o da la opción de cancelar el proceso de exportación.
3. Memoria insuficiente: el usuario no dispone de suficiente memoria en la ubicación seleccionada para exportar el archivo. El sistema lanza un mensaje de error y solicita que le proporciones otra ubicación o da la opción de cancelar el proceso de exportación.

Título: Importar alfabeto

Actor: Usuario

Descripción: Los usuarios pueden cargar un archivo .csv que contiene un alfabeto.

Requisitos:

1. El formato del archivo a importar tiene que ser .csv.

2. El usuario debe proporcionar un nombre al alfabeto que se va a importar.
    a. El nombre del alfabeto no debe contener más de 20 caracteres ni menos de 4.
    b. Los caracteres deben de estar comprendidos entre el 33-126 de la tabla ASCII. Excluyendo el 34 y 44.
3. El usuario no debe tener más de 10 alfabetos en el sistema.

Comportamiento:

1. El sistema le indica al usuario que le proporcione la ubicación del archivo a exportar.
2. El sistema valida que el archivo sea del formato correspondiente.
3. El sistema solicita al usuario que introduzca un nombre al alfabeto importado.
4. El sistema guarda el nuevo alfabeto.

Errores y cursos alternativos:

1. Formato incompatible: el formato del archivo seleccionado para importar no es un archivo .csv. El sistema lanza un mensaje de error solicitando al usuario que seleccione otro archivo para importar o da la opción de cancelar el proceso de importación.
2. Nombre del archivo inválido: el nombre proporcionado para el nuevo alfabeto no cumple el formato especificado de longitud o contiene un caracter invalido. El sistema lanza un mensaje de error y solicita que le proporciones otro nombre o da la opción de cancelar el proceso de importación.
3. Alfabeto ya existente: el nombre del alfabeto a importar coincide con el nombre de un alfabeto existente. El sistema lanza un mensaje de error y solicita que le proporciones otro nombre o da la opción de cancelar el proceso de importación.

Título: Editar alfabeto

Actor: Usuario

Descripción: Los usuarios pueden realizar cambios en un alfabeto existente, como agregar o eliminar letras, o modificar sus atributos.

Requisitos:

1. El usuario debe tener alfabetos guardados.
2. El alfabeto no debe contener más de 25 caracteres.

Comportamiento:

1. El sistema solicita al usuario que seleccione el alfabeto a editar.
2. El usuario modifica el alfabeto.
    2.1. Si el alfabeto estaba asociado a un teclado solo se puede cambiar su nombre.T
3. El sistema valida que los cambios realizados están dentro del formato exigido.
4. El sistema guarda los cambios realizados.

Errores y cursos alternativos:

1. Formato incorrecto: el alfabeto modificado no cumple los requisitos de formato de longitud especificados.

Título: Crear texto

Actor: Usuario

Descripción: Los usuarios pueden escribir o añadir textos nuevos.

Requisitos:

1. El texto no debe contener más de 1024 palabras ni más de 10240 carácteres.
2. El usuario debe proporcionar un nombre al texto.
    a. El nombre del texto no debe contener más de 20 caracteres ni menos de 1.
    b. Los caracteres del nombre deben de estar comprendidos entre el 33-126 de la tabla ASCII. Excluyendo el 34 y 44 (espacio y coma).
3. El usuario no tiene más de diez textos guardados.

Comportamiento:

1. El sistema solicita al usuario que inserte un texto.
2. El sistema solicita un nombre para el texto.
3. El sistema comprueba y valida el texto y lo guarda.

Errores y cursos alternativos:

1. Texto con exceso de palabras: El texto supera el límite de palabras máximo. El sistema lo notificará y dará la opción de introducir nuevamente el texto o de cancelar la acción.
2. Texto ya existente: El texto con el nombre proporcionado por el usuario ya existe. El sistema notificará del error y dará la opción de cambiar el nombre, eliminar el texto ya existente o cancelar la acción.
3. Exceso de textos: El número de textos guardados supera el máximo permitido al crear uno nuevo. El sistema notificará el problema y dará la opción de eliminar uno ya existente o cancelar la opción.

Título: Eliminar texto

Actor: Usuario

Descripción: Los usuarios pueden eliminar textos que ya no sean relevantes para su diseño de teclado.

Requisitos:

1. El usuario debe seleccionar el texto que quiere eliminar.
2. El texto con el nombre proporcionado debe existir en el sistema.

Comportamiento:

1. El sistema solicitará el texto a eliminar.
2. El sistema comprueba la existencia del texto.
3. El sistema elimina el texto.

Errores y cursos alternativos:

1. Texto inexistente: El texto con nombre proporcionado por el usuario no existe en el sistema. El sistema informará del error y dará la opción de seleccionar otro texto o de cancelar la opción.

Título: Exportar texto

Actor: Usuario

Descripción :Los usuarios pueden exportar en un archivo de texto los textos almacenados en el sistema.

Requisitos:

1. El texto debe existir en el sistema.
2. El usuario debe proporcionar una ruta válida para guardar el archivo.
3. El usuario debe tener permisos para acceder y editar en la ruta deseada.
4. El archivo no debe ocupar más memoria de la disponible en el sistema.
5. El usuario debe proporcionar un nombre para el archivo.
    a. El nombre del archivo no debe contener más de 20 caracteres ni menos de 1.
    b. Los caracteres del nombre deben de estar comprendidos entre el 33-126 de la tabla ASCII. Excluyendo el 34 y 44 (espacio y coma).

Comportamiento:

1. El sistema solicita al usuario que seleccione el texto que se desea exportar.
2. El sistema solicita al usuario que proporcione un nombre y la ruta donde se quiere guardar el archivo con el texto.
3. El sistema comprueba que la ruta y los permisos sean correctos.
4. El sistema comprueba que el archivo no ocupa más de la memoria permitida.
5. El sistema crea un archivo en la ruta proporcionada con el formato y el texto escogido.

Errores y cursos alternativos:

1. No hay espacio de almacenamiento suficiente: El archivo que se quiere exportar ocupa más memoria de la que hay disponible. El sistema notificará al usuario y le permitirá cambiar la ruta proporcionada o cancelar la opción.
2. No tiene permisos suficientes: El usuario no cuenta con los permisos necesarios para escribir en la memoria de la ruta seleccionada. El sistema notificará del suceso y permitirá cambiar la ruta o cancelar la opción.
3. El archivo ya existe: El archivo con el nombre proporcionado ya existe. El sistema notifica al usuario y le permite cambiar el nombre del archivo, eliminar el ya existente o cancelar la acción.

Título: Importar texto

Actor: Usuario

Descripción: Los usuarios pueden cargar un archivo de texto que contiene un texto específico para su análisis de frecuencia de palabras.

Requisitos:

1. El formato del archivo importado debe ser ".txt".
2. El usuario debe proporcionar un nombre al texto.
    a. El nombre del archivo no debe contener más de 20 caracteres ni menos de 1.
    b. Los caracteres del nombre deben de estar comprendidos entre el 33-126 de la tabla ASCII. Excluyendo el 34 y 44 (espacio y coma).
3. El sistema debe contener menos de 10 textos.

Comportamiento:

1. El sistema solicita al usuario la ruta del archivo que quiere exportar.
2. El sistema comprueba que el formato sea el correspondiente.
3. El sistema comprueba que el sistema contenga menos de 10 textos.
4. El sistema guarda el texto.

Errores y cursos alternativos:

1. Formato incompatible: El formato del archivo seleccionado no corresponde al formato solicitado. El sistema notificará del problema y dará la opción de seleccionar otro archivo o de cancelar la acción.
2. Nombre invalido: El nombre proporcionado para guardar el texto no cumple con los requisitos solicitados. El sistema notificará sobre el problema y dará la opción de cambiar el nombre o de cancelar la acción.

Título: Editar texto

Actor: Usuario

Descripción: Los usuarios pueden realizar cambios en el contenido de un texto cargado, como agregar, eliminar o modificar palabras.

Requisitos:

1. El texto debe existir.
2. El texto editado no puede contener más de 1024 palabras.

Comportamiento:

1. El sistema solicitará al usuario que seleccione el texto a editar.
2. El usuario modificará el texto.
3. El sistema comprobará que el texto cumpla con los requisitos de formato.
4. El sistema guardará el texto con las modificaciones.

Errores y cursos alternativos:

1. Formato incorrecto: El formato del texto modificado no concuerda con el formato requerido. El sistema permitirá volver a modificar el texto o cancelar la acción.

Título: Crear lista de palabras con frecuencias

Actor: Usuario

Descripción: El usuario puede crear listas de palabras con frecuencias.

Requisitos:

1. El nombre de la lista es válido.
2. El usuario no tiene más de 10 listas.
3. La suma de las frecuencias de todas las palabras tiene que ser menor a 1024 y la suma de las frecuencias de todos los carácteres tiene que ser menor a 10240.

Comportamiento:

1. El usuario introduce el nombre de la lista.
2. El sistema crea una lista vacía con dicho nombre.

Errores y cursos alternativos:

1. Lista ya existente: Ya existe una lista con ese nombre. Se cancela la creación de la lista.
2. Máximo listas alcanzado: El usuario tiene 10 listas. Se cancela la creación de la lista.
3. Máximo frecuencias alcanzado: Las frecuencias de todas las palabras es mayor a 1024. El sistema notificará el problema y dará la opción de eliminar una ya existente o cancelar la opción.

Título: Eliminar lista de palabras con frecuencias

Actor: Usuario

Descripción: El usuario puede crear listas de palabras con frecuencias.

Requisitos:

1. La lista ya existe en el sistema.
2. El usuario debe proporcionar el nombre del texto que quiere eliminar.

Comportamiento:

1. El sistema solicitará el nombre de la lista a eliminar.
2. El sistema comprueba la existencia de la lista.
3. El sistema elimina la lista.

Errores y cursos alternativos:

1. Lista inexistente: La lista con nombre proporcionado por el usuario no existe en el sistema. El sistema informará del error y dará la opción de seleccionar otra lista o de cancelar la opción.

Título: Exportar lista de palabras con frecuencias

Actor: Usuario

Descripción: Los usuarios pueden exportar una lista de palabras almacenada en el sistema.

Requisitos:

1. La lista debe existir en el sistema.
2. El usuario debe proporcionar una ruta válida para guardar el archivo.
3. El usuario debe tener permisos para acceder y editar en la ruta deseada.
4. El archivo no debe ocupar más memoria de la disponible en el sistema.
5. El usuario debe proporcionar un nombre para el archivo.

Comportamiento:

1. El sistema solicita al usuario que ingrese el nombre de la lista que desea exportar.
2. El sistema solicita al usuario que ingrese el nombre que tendrá el archivo exportado.
3. El usuario establece la ubicación en la que desea guardar la lista exportada.
4. El sistema crea un archivo .csv en la ubicación establecida.

Errores y cursos alternativos:

1. Lita inexistente: el nombre proporcionado no pertenece a ningúna lista del sistema. El sistema lanza un mensaje de error y solicita que le proporciones otro nombre o da la opción de cancelar el proceso de exportación.
2. Nombre del archivo inválido: el nombre proporcionado para el archivo a exportar no cumple el formato especificado de longitud o contiene un caracter invalido. El sistema lanza un mensaje de error y solicita que le proporciones otro nombre o da la opción de cancelar el proceso de exportación.
3. Memoria insuficiente: el usuario no dispone de suficiente memoria en la ubicación seleccionada para exportar el archivo. El sistema lanza un mensaje de error y solicita que le proporciones otra ubicación o da la opción de cancelar el proceso de exportación.

Título: Importar lista de palabras con frecuencias

Actor: Usuario

Descripción: Los usuarios pueden cargar un archivo .csv que contiene una lista.

Requisitos:

1. El formato del archivo a importar tiene que ser .csv.
2. El usuario debe proporcionar un nombre a la lista que se va a importar.
3. El usuario no debe tener más de 10 listas en el sistema.

Comportamiento:

1. El sistema le indica al usuario que le proporcione la ubicación del archivo a exportar.
2. El sistema valida que el archivo sea del formato correspondiente.
3. El sistema solicita al usuario que introduzca un nombre a la lista importada.
4. El sistema guarda la nueva lista.

Errores y cursos alternativos:

1. Formato incompatible: el formato del archivo seleccionado para importar no es un archivo .csv. El sistema lanza un mensaje de error solicitando al usuario que seleccione otro archivo para importar o da la opción de cancelar el proceso de importación.
2. Nombre del archivo inválido: el nombre proporcionado para la nueva lista no cumple el formato especificado de longitud o contiene un caracter invalido. El sistema lanza un mensaje de error y solicita que le proporciones otro nombre o da la opción de cancelar el proceso de importación.
3. Alfabeto ya existente: el nombre de la lista a importar coincide con el nombre de una lista existente. El sistema lanza un mensaje de error y solicita que le proporciones otro nombre o da la opción de cancelar el proceso de importación.

Título: Editar lista de palabras con frecuencias

Actor: Usuario

Descripción: Los usuarios pueden realizar cambios en una lista existente, como agregar o eliminar palabras.

Requisitos:

1. El usuario debe tener listas guardadas.

Comportamiento:

1. El sistema solicita al usuario que seleccione la lista a editar.
2. El usuario modifica la lista.
3. El sistema valida que los cambios realizados están dentro del formato exigido.
4. El sistema guarda los cambios realizados.

Errores y cursos alternativos:

1. Formato incorrecto: la lista modificada no cumple los requisitos de formato especificados.

Título: Editar perfil

Actor: Usuario

Descripción: El usuario puede modificar dentro del sistema su nombre de usuario, su contraseña y su correo electrónico cuando sea necesario.

Requisitos:

1. El usuario debe proporcionar una cuenta de correo electrónico válida
2. El usuario debe proporcionar un nombre de usuario
    a. El nombre de usuario no debe contener más de 20 caracteres ni menos de 3.
    b. Los caracteres deben de estar comprendidos entre el 33-126 de la tabla ASCII. Excluyendo el 34 y 44.
    c. El nombre de usuario tiene que ser único
3. El usuario debe proporcionar una contraseña
    a. La contraseña no debe contener más de 20 caracteres ni menos de 3.
    b. Los caracteres deben estar comprendidos entre el 32-126 de la tabla ASCII. Excluyendo el 34 y 44.

Comportamiento:

1. El usuario modifica los campos que considere necesarios.
2. El sistema valida que todos los cambios realizados sean correctos y válidos.
3. El sistema guarda los cambios realizados.

Errores y cursos alternativos:

1. Dirección de correo electrónico ya registrada: La dirección de correo electrónico ya está registrada con otro nombre de usuario. El sistema cancela el proceso de registro.
2. Dirección de correo electrónico inválida: La dirección de correo electrónico no ha sido validada o no existe. El sistema lanza un mensaje de error y solicita un correo electrónico alternativo o da la opción de cancelar el proceso de registro.
3. Nombre de usuario ya existente: El nombre de usuario proporcionado ya está registrado. El sistema lanza un mensaje de error y solicita un nuevo nombre de usuario o da la opción de cancelar el proceso.
4. Nombre de usuario incorrecto: El nombre de usuario proporcionado no cumple con el formato especificado de longitud o contiene algún carácter invalido El sistema lanza un mensaje de error y solicita un nuevo nombre de usuario o da la opción de cancelar el proceso de registro.

5. Contraseña incorrecta: La contraseña proporcionada no cumple con el formato especificado de longitud o contiene algún carácter invalido. El sistema lanza un mensaje de error y solicita una nueva contraseña o da la opción de cancelar el proceso de registro.

Título: Cerrar sesión

Actor: Usuario

Descripción: El usuario autorizado cierra sesión y pasa a ser un usuario no autorizado.

Requisitos:

1. El usuario tiene que estar autorizado, es decir, en una sesión de usuario activa.
2. El sistema no tiene que estar procesando información relacionada a este usuario..

Comportamiento:

1. El usuario ingresa el comando “logOut”.
2. El sistema comprueba que no se esté procesando información y cierra la sesión del usuario.

Errores y cursos alternativos:

1. Procesando información: El usuario estaba realizando una acción que requiere procesar información. El sistema lo notificará y le dará la opción de esperar el tiempo necesario, cancelar la acción o terminar la sesión con el riesgo de perder información.

2. MODELO CONCEPTUAL DOMINIO

2.1. Diagrama del modelo conceptual de dominio

2.2. Descripción de las clases de dominio

Clase Usuario

Descripción: Representa un usuario en el sistema, donde cada usuario tiene un nombre de usuario único (username), una contraseña (password), y una dirección de correo electrónico (mail). Además, el usuario puede gestionar distintos elementos como teclados, alfabetos, listas de palabras y textos.

Atributos:

- String username: Identificador del usuario, nombre del usuario.
- String Password: Contraseña asociada al usuario.
- String Mail: Correo electrónico asociado al usuario.
- HashMap<String,Integer> mapaTeclados: Mapa que guarda la asociación nombre-id de los teclados en propiedad del usuario.
- HashMap<String,Integer> mapaAlfabetos: Mapa que guarda la asociación nombre-id de los alfabetos en propiedad del usuario.
- HashMap<String,Integer> mapaTextos: Mapa que guarda la asociación nombre-id de los textos en propiedad del usuario.
- HashMap<String,Integer> mapaListasPalabras: Mapa que guarda la asociación nombre-id de las listas de palabras en propiedad del usuario.

Métodos:

Métodos de Acceso (Getters)

- getUsername(): Obtiene el nombre de usuario.
- getPassword(): Obtiene la contraseña del usuario.
- getMail(): Obtiene la dirección de correo electrónico del usuario.
- getTeclados(): Obtiene una lista de identificadores de teclados.
- getAlfabetos(): Obtiene una lista de identificadores de alfabetos.
- getListasPalabras(): Obtiene una lista de identificadores de listas de palabras.
- getTextos(): Obtiene una lista de identificadores de textos.

Métodos de Comprobación (Checkers)

- checkTeclado(nombreTeclado): Verifica si el usuario tiene un teclado dado su nombre.
- checkIdTeclado(idTeclado): Verifica si el usuario tiene un teclado dado su identificador.
- checkAlfabeto(nombreAlfabeto): Verifica si el usuario tiene un alfabeto dado su nombre.
- checkIdAlfabeto(idAlfabeto): Verifica si el usuario tiene un alfabeto dado su identificador.
- checkLista(nombreLista): Verifica si el usuario tiene una lista de palabras dada su nombre.
- checkIdLista(idLista): Verifica si el usuario tiene una lista de palabras dada su identificador.
- checkTexto(nombreTexto): Verifica si el usuario tiene un texto dado su nombre.
- checkIdTexto(idTexto): Verifica si el usuario tiene un texto dado su identificador.
- checkSizeTeclados(): Verifica si el usuario tiene espacio para más teclados.
- checkSizeAlfabetos(): Verifica si el usuario tiene espacio para más alfabetos.
- checkSizeListas(): Verifica si el usuario tiene espacio para más listas de palabras.
- checkSizeTextos(): Verifica si el usuario tiene espacio para más textos.
- emptyTeclados(): Verifica si el usuario no tiene teclados.
- emptyAlfabetos(): Verifica si el usuario no tiene alfabetos.
- emptyListas(): Verifica si el usuario no tiene listas de palabras.
- emptyTextos(): Verifica si el usuario no tiene textos.

Métodos de Modificación (Setters)

- setUsername(newUsername): Modifica el nombre de usuario.
- setPassword(newPassword): Modifica la contraseña del usuario.
- setMail(newMail): Modifica la dirección de correo electrónico del usuario.

Métodos de Manipulación de Datos

- addTeclado(idTeclado, nombreTeclado): Añade un nuevo teclado al usuario.
- addAlfabeto(idAlfabeto, nombreAlfabeto): Añade un nuevo alfabeto al usuario.
- addListaPalabras(idLista, nombreLista): Añade una nueva lista de palabras al usuario.
- addTexto(idTexto, nombreTexto): Añade un nuevo texto al usuario.
- deleteTeclado(nombreTeclado): Elimina un teclado del usuario.
- deleteAlfabeto(nombreAlfabeto): Elimina un alfabeto del usuario.
- deleteListaPalabras(nombreLista): Elimina una lista de palabras del usuario.
- deleteTexto(nombreTexto): Elimina un texto del usuario.
- editarTeclado(nombreTeclado, newNombreTeclado): Edita el nombre de un teclado.
- editarAlfabeto(nombreAlfabeto, newNombreAlfabeto): Edita el nombre de un alfabeto.
- editarLista(nombreLista, newNombreLista): Edita el nombre de una lista de palabras.
- editarTexto(nombreTexto, newNombreTexto): Edita el nombre de un texto.

Clase Alfabeto

Descripción: Representa un conjunto de símbolos utilizados en el sistema, y puede estar asociado a uno o varios teclados. Este alfabeto se utiliza para definir las letras o caracteres que conforman los teclados asociados.

Atributos:

- String alfabeto: Conjunto de símbolos que conforman el alfabeto.
- ArrayList<Integer> idsTecladosAsociados: ArrayList que almacena los identificadores de los teclados asociados al alfabeto.

Métodos:

Métodos de Acceso (Getters)

- getAlfabeto(): Obtiene el conjunto de símbolos del alfabeto.
- getIdsTecladosAsociados(): Obtiene los identificadores de los teclados asociados al alfabeto.

Métodos de Modificación (Setters)

- setAlfabeto(alfabeto): Modifica el conjunto de símbolos del alfabeto.

Métodos de Comprobación

- hasTecladoAsociado(): Verifica si el alfabeto tiene al menos un teclado asociado.
- checkTecladoAsociado(idTeclado): Verifica si un teclado con un identificador dado está asociado al alfabeto.

Métodos de Manipulación de Datos

- addTecladoAsociado(idTeclado): Añade un teclado al conjunto de teclados asociados al alfabeto.

- deleteTecladoAsociado(idTeclado): Elimina un teclado del conjunto de teclados asociados al alfabeto.

Clase Conjunto De Alfabetos

Descripción: Representa un conjunto de alfabetos, donde cada alfabeto está identificado por un identificador único. Este conjunto de alfabetos permite gestionar y manipular múltiples alfabetos asociados al usuario activo.

Atributos:

- HashMap<Integer,Alfabeto> alfabetos: HashMap que almacena los alfabetos del usuario activo en el sistema, utilizando el identificador como clave y la instancia de la clase Alfabeto como valor.

Métodos:

Métodos de Acceso (Getters)

- getAlfabeto(id): Obtiene el conjunto de símbolos de un alfabeto específico.
- getIdsTecladosAsociados(idAlfabeto): Obtiene los identificadores de los teclados asociados a un alfabeto específico.

Métodos de Comprobación

- hasTecladoAsociado(idAlfabeto): Verifica si un alfabeto específico está asociado a algún teclado.
- checkTecladoAsociado(idAlfabeto, idTeclado): Verifica si un teclado con un identificador dado está asociado a un alfabeto específico.

Métodos de Manipulación de Datos

- crearAlfabeto(id, alfabeto): Añade un nuevo alfabeto al conjunto con un identificador dado y un conjunto específico de símbolos.
- editarAlfabeto(idAlfabeto, newAlfabeto): Modifica el conjunto de símbolos de un alfabeto específico.
- eliminarAlfabeto(id): Elimina un alfabeto del conjunto.
- addTecladoAsociado(idAlfabeto, idTeclado): Asocia un teclado al alfabeto especificado.
- deleteTecladoAsociado(idAlfabeto, idTeclado): Desasocia un teclado del alfabeto especificado.

Clase Texto

Descripción: Representa un bloque de texto que puede estar asociado a uno o más teclados. Cada instancia de esta clase contiene un conjunto de símbolos (caracteres) y mantiene un seguimiento de los teclados a los que está asociado.

Atributos:

- String texto: String que almacena el conjunto de caracteres que conforman el texto.
- ArrayList<Integer> idsTecladosAsociados: ArrayList que guarda los identificadores de los teclados asociados al texto.

Métodos:

Métodos de Acceso (Getters)

- getTexto(): Obtiene el conjunto de símbolos del texto.

- getIdsTecladosAsociados(): Obtiene los identificadores de los teclados asociados al texto.

Métodos de Modificación(Setters)

- setTexto(Texto): Modifica el conjunto de símbolos del texto.

Métodos de Comprobación

- hasTecladoAsociado(): Verifica si el texto está asociado a algún teclado.
- checkTecladoAsociado(idTeclado): Verifica si un teclado específico está asociado al texto.

Métodos de Manipulación de Datos

- addTecladoAsociado(idTeclado): Asocia un teclado al texto.
- deleteTecladoAsociado(idTeclado): Desasocia un teclado del texto.

Clase Conjunto De Textos

Descripción: Representa un conjunto de instancias de la clase Texto. Permite crear, editar, eliminar y gestionar textos en propiedad del usuario activo. Cada texto dentro del conjunto puede estar asociado a uno o más teclados.

Atributos:

- HashMap<Integer,Texto> textos: HashMap que almacena instancias de la clase Texto, donde cada texto está identificado por un entero (id).

Métodos:

Métodos de Acceso(Getters)

- getTexto(id): Obtiene el contenido de un texto específico en el conjunto.
- getIdsTecladosAsociados(idTexto): Obtiene los identificadores de los teclados asociados a un texto.

Métodos de Comprobación

- hasTecladoAsociado(idTexto): Verifica si un texto está asociado a algún teclado.
- checkTecladoAsociado(idTexto, idTeclado): Verifica si un teclado específico está asociado a un texto.

Métodos de Manipulación de Datos

- crearTexto(id, texto): Añade un nuevo texto al conjunto con un identificador único.
- editarTexto(idTexto, newTexto): Modifica el contenido de un texto existente en el conjunto.
- eliminarTexto(id): Elimina un texto del conjunto
- addTecladoAsociado(idTexto, idTeclado): Asocia un teclado a un texto, devolviendo true si se realiza correctamente, false si ya estaba asociado.
- deleteTecladoAsociado(idTexto, idTeclado): Desasocia un teclado de un texto, devolviendo true si se realiza correctamente, false si no estaba asociado.

Clase Lista De Palabras

Descripción: Representa un conjunto de palabras junto con sus frecuencias. Cada palabra en la lista tiene una frecuencia asociada. Además, la lista puede estar vinculada a uno o más teclados.

Atributos:

- HashMap<String,Integer> listaPalabras: HashMap que almacena palabras como claves y sus correspondientes frecuencias como valores.
- ArrayList<Integer> idsTecladosAsociados: ArrayList de identificadores de teclados asociados a la lista de palabras.

Métodos:

Métodos de Acceso(Getters)

- getListaPalabras(): Obtiene el HashMap de palabras y sus frecuencias.
- getIdsTecladosAsociados(): Obtiene los identificadores de los teclados asociados a la lista de palabras.
- getFrec(Palabra): Obtiene la frecuencia de una palabra específica en la lista.

Métodos de Modificación(Setters)

- setListaPalabras(listaPalabras): Establece un nuevo HashMap de palabras y frecuencias.

Métodos de Comprobación

- hasTecladoAsociado(): Verifica si la lista de palabras está asociada a algún teclado.
- checkTecladoAsociado(idTeclado): Verifica si un teclado específico está asociado a la lista de palabras.

Métodos de Manipulación de Datos

- addTecladoAsociado(idTeclado): Asocia un teclado a la lista de palabras, devolviendo true si se realiza correctamente, false si ya estaba asociado.
- deleteTecladoAsociado(idTeclado): Desasocia un teclado de la lista de palabras, devolviendo true si se realiza correctamente, false si no estaba asociado.

Clase Conjunto De Listas

Descripción: Representa el conjunto de listas de palabras en propiedad del usuario activo, cada una identificada por un ID único. Cada lista de palabras está compuesta por un HashMap que asocia palabras con sus respectivas frecuencias. Además, cada lista puede estar vinculada a uno o más teclados.

Atributos:

- HashMap<Integer,String> listasDePalabras: HashMap que almacena listas de palabras, donde cada lista está identificada por un ID único.

Métodos:

Métodos de Acceso(Getters)

- getListaDePalabras(id): Obtiene el HashMap de palabras y frecuencias de una lista de palabras específica, identificada por su ID.

- getIdsTecladosAsociados(idLista): Obtiene los identificadores de los teclados asociados a una lista de palabras específica.

Métodos de Comprobación

- hasTecladoAsociado(idLista): Verifica si una lista de palabras está asociada a algún teclado.
- checkTecladoAsociado(idLista, idTeclado): Verifica si un teclado específico está asociado a una lista de palabras.

Métodos de Manipulación de Datos

- crearListaDePalabras(id, listaPalabras): Añade una nueva lista de palabras al conjunto, identificada por un ID único y con un HashMap dado de palabras y frecuencias.
- editarListaDePalabras(idLista, newLista): Edita una lista de palabras existente, identificada por su ID, estableciendo un nuevo HashMap de palabras y frecuencias.
- eliminarListaDePalabras(id): Elimina una lista de palabras del conjunto, identificada por su ID.
- addTecladoAsociado(idLista, idTeclado): Asocia un teclado a una lista de palabras, devolviendo true si se realiza correctamente, false si ya estaba asociado.
- deleteTecladoAsociado(idLista, idTeclado): Desasocia un teclado de una lista de palabras, devolviendo true si se realiza correctamente, false si no estaba asociado.

Clase Teclado

Descripción: Representa un teclado físico o virtual, con su disposición de teclas y su alfabeto, texto o lista de palabras asociado. Los atributos principales incluyen la disposición de teclas, así como identificadores para indicar su asociación con un alfabeto, un texto o una lista.

Atributos:

- char[][] layout: Matriz bidimensional que representa la distribución de las teclas en el teclado.
- Integer idAlfabeto: Identificador del alfabeto asociado al teclado.
- Integer idTexto: Identificador del texto asociado al teclado (-1 si no está asociado a ningún texto).
- Integer idLista: Identificador de la lista asociada al teclado (-1 si no está asociado a ninguna lista).

Métodos:

Métodos de Acceso(Getters)

- getLayout(): Obtiene la disposición de las teclas del teclado.
- getIdAlfabeto(): Obtiene el identificador del alfabeto asociado al teclado.
- getIdTexto(): Obtiene el identificador del texto asociado al teclado.
- getIdLista(): Obtiene el identificador de la lista asociada al teclado.

Métodos de Modificación

- setLayout(layout): Establece una nueva disposición de teclas para el teclado.

Clase Conjunto De Teclados

Descripción: Representa un conjunto de teclados en propiedad del usuario activo, cada uno identificado por un identificador numérico único. Permite gestionar y acceder a múltiples teclados, cada uno con su propia disposición de teclas, asociado a un alfabeto y asociado opcionalmente a un texto o una lista de palabras.

Atributos:

- HashMap<Integer,Teclado> teclados: HashMap que almacena los teclados del conjunto, donde la clave es el identificador del teclado y el valor es una instancia de la clase Teclado.

Métodos:

Métodos de Acceso(Getters)

- getIdAlfabeto(idTeclado): Obtiene el identificador del alfabeto asociado al teclado especificado.
- getIdTexto(idTeclado): Obtiene el identificador del texto asociado al teclado especificado.
- getIdLista(idTeclado): Obtiene el identificador de la lista asociada al teclado especificado.
- getLayoutTeclado(idTeclado): Obtiene la disposición de teclas del teclado especificado.

Métodos de Manipulación de Datos

- crearTeclado(id, idAlfabeto, idTexto, idLista, layout): Añade un teclado al conjunto de teclados con el identificador especificado, asociándolo a un alfabeto y opcionalmente a un texto o una lista de palabras, y proporcionando su distribución de teclas.
- eliminarTeclado(id): Elimina un teclado del conjunto por su identificador.

Clase Error

Descripción: La clase Error representa un conjunto de posibles errores que pueden ocurrir en el sistema. Cada error tiene un identificador único asociado y un mensaje descriptivo que se puede imprimir en la salida estándar para informar al usuario sobre la naturaleza del error.

Atributos:

- Enumeración typeError: que contiene una lista exhaustiva de tipos de error posibles. Cada tipo de error tiene un mensaje asociado que se mostrará al usuario cuando ocurra.
- typeError te: Variable que almacena el tipo de error actual, utilizando la enumeración typeError.

Métodos:

- printError(int i): Imprime en la salida estándar el mensaje asociado al tipo de error indicado por el índice i. Este método se utiliza para mostrar mensajes de error específicos.

Clase Utilidad

Descripción: La clase Utilidad proporciona métodos de utilidad para realizar comprobaciones y conversiones relacionadas con la aplicación. Estas funciones abarcan la validación de nombres, correos electrónicos, alfabetos, textos, listas de palabras, así como la compatibilidad entre alfabetos y textos o listas.

Atributos:

- No contiene atributos específicos.

Métodos:

Métodos principales

- comprobarPasswordONombre(String nombre) → Boolean: Comprueba si el nombre cumple con los requisitos establecidos, como la longitud y la ausencia de caracteres especiales.
- comprobarAlfabeto(String alfabeto) → Boolean: Verifica si el alfabeto cumple con los requisitos de longitud.
- comprobarTexto(String texto) → Boolean: Evalúa si el texto cumple con las restricciones de longitud y conteo de palabras.
- comprobarListaPalabras(HashMap<String, Integer> listaPalabras) → Boolean: Examina si la lista de palabras cumple con los requisitos de longitud y conteo total.
- comprobarCorreo(String correo) → Boolean: Comprueba la validez del formato de correo electrónico.
- conversorTextoLista(String texto) → HashMap<String, Integer>: Convierte un texto en una lista de palabras, asignando frecuencias a cada palabra.
- comprobarCompatibilidadTextoAlfabeto(String alfabeto, String texto) → Boolean: Verifica si un texto es compatible con un alfabeto dado.
- comprobarCompatibilidadListaAlfabeto(String alfabeto, HashMap<String, Integer> lista) → Boolean: Comprueba si una lista de palabras es compatible con un alfabeto dado.

Clase Controlador 1

Descripción: Controlador 1 actúa como un controlador central que gestiona conjuntos de alfabetos, teclados, listas de palabras y textos en la aplicación. Proporciona métodos para crear, editar y eliminar elementos de estos conjuntos, así como funciones para obtener información relacionada con estos elementos.

Atributos:

- CjtAlfabeto alfabetos: Conjunto de alfabetos gestionado por la clase CjtAlfabeto.
- CjtTeclado teclados: Conjunto de teclados gestionado por la clase CjtTeclado.
- CjtListaPalabras listasDePalabras: Conjunto de listas de palabras gestionado por la clase CjtListaPalabras.
- CjtTextos textos: Conjunto de textos gestionado por la clase CjtTextos.

Métodos:

Métodos de Acceso(Getters)

- getTexto(Integer id) → String: Obtiene el texto con el identificador proporcionado.
- getAlfabeto(Integer id) → String: Obtiene el alfabeto con el identificador proporcionado.
- getListaPalabras(Integer id) → HashMap<String, Integer>: Obtiene la lista de palabras con el identificador proporcionado.
- getTeclado(Integer idTeclado) → char[][]: Obtiene la distribución de teclas de un teclado específico.
- getTecladosAsociadosAlfabeto(Integer idAlfabeto) → ArrayList<Integer>: Obtiene los identificadores de los teclados asociados a un alfabeto.
- getTecladosAsociadosTexto(Integer idTexto) → ArrayList<Integer>: Obtiene los identificadores de los teclados asociados a un texto.
- getTecladosAsociadosLista(Integer idLista) → ArrayList<Integer>: Obtiene los identificadores de los teclados asociados a una lista.

Métodos de Manipulación de Datos

- crearAlfabeto(Integer id, String alfabeto): Añade un nuevo alfabeto al conjunto de alfabetos.

- crearTeclado(Integer id, Integer idAlfabeto, Integer idTexto, Integer idLista, char[][] layout): Añade un nuevo teclado al conjunto de teclados y establece asociaciones con alfabetos, textos o listas de palabras.
- crearListaDePalabras(Integer id, HashMap<String, Integer> listaPalabras): Añade una nueva lista de palabras al conjunto.
- crearTexto(Integer id, String texto): Añade un nuevo texto al conjunto.
- editarAlfabeto(Integer idAlfabeto, String newAlfabeto) → Boolean: Edita un alfabeto existente, devolviendo true si la operación fue exitosa y false si el alfabeto está asociado a un teclado.
- editarTexto(Integer idTexto, String newTexto) → Boolean: Edita un texto existente, devolviendo true si la operación fue exitosa y false si el texto está asociado a un teclado.
- editarListaPalabras(Integer idLista, HashMap<String, Integer> newLista) → Boolean: Edita una lista de palabras existente, devolviendo true si la operación fue exitosa y false si la lista está asociada a un teclado.
- eliminarTeclado(Integer id): Elimina un teclado del conjunto de teclados y las asociaciones correspondientes con alfabetos, textos o listas de palabras.
- eliminarAlfabeto(Integer id) → Boolean: Elimina un alfabeto del conjunto, devolviendo true si la operación fue exitosa y false si el alfabeto está asociado a un teclado.
- eliminarListaDePalabras(Integer id) → Boolean: Elimina una lista de palabras del conjunto, devolviendo true si la operación fue exitosa y false si la lista está asociada a un teclado.
- eliminarTexto(Integer id) → Boolean: Elimina un texto del conjunto, devolviendo true si la operación fue exitosa y false si el texto está asociado a un teclado.

Clase Controlador De Dominio

Descripción: Controlador de Dominio es un componente fundamental del sistema, encargado de gestionar y coordinar diversas operaciones relacionadas con la manipulación y control de elementos como teclados, textos, alfabetos y listas. Actúa como intermediario entre la interfaz de usuario y la lógica subyacente del sistema, asegurando la coherencia y la integridad de los datos.

Atributos:

- Usuario usuario: Representa la instancia de la clase Usuario, que contiene la información y operaciones relacionadas con el usuario actual del sistema.
- Controlador1 controlador: Instancia de la clase Controlador 1, que parece ser una clase externa que maneja operaciones específicas del sistema.
- HashMap<String,Integer> usernameId: Un HashMap que almacena la asociación del nombre de usuario con su identificador (id).
- Utilidad util: Instancia de la clase Utilidad, que proporciona funciones de utilidad para diversas comprobaciones.
- persistenciaFake persistencia: Instancia de la clase persistenciaFake, que simula operaciones de persistencia para pruebas o prototipos.
- Integer idUser, idTeclado, idTexto, idLista, idAlfabeto: Variables que almacenan los índices actuales para la creación de nuevos usuarios, teclados, textos, listas y alfabetos, respectivamente.
- Boolean activo: Un booleano que indica si hay un usuario activo en la sesión actual.
- idActivo: Almacena el identificador del usuario activo.

Métodos:

Métodos principales

- cargarReferencias() - Carga en el programa todos los usernames con sus ids.
- cargarIndices() - Carga en el programa los índices para usar como identificadores.

- loginUsuario(String username, String password) - Carga un usuario en el sistema.
- registroUsuario(String username, String password, String mail) - Registra un nuevo usuario en el sistema.
- logoutUsuario() - Cierra la sesión del usuario activo.
- crearTecladoConLista(String nombre, String nombreAlfabeto, String nombreLista) - Crea un teclado a partir de un alfabeto y una lista de palabras con frecuencias.
- crearTecladoConTexto(String nombre, String nombreAlfabeto, String nombreTexto) - Crea un teclado a partir de un alfabeto y un texto.
- crearTexto(String nombre, String texto) - Crea un texto y lo añade al sistema.
- crearAlfabeto(String nombre, String alfabeto) - Crea un alfabeto y lo añade al sistema.
- crearListaDePalabras(String nombre, HashMap<String, Integer> listaPalabras) - Crea una lista de palabras y la añade al sistema.
- eliminarUsuario(String password) - Elimina el usuario activo del sistema.
- eliminarTexto(String nombre) - Elimina un texto del sistema.
- eliminarAlfabeto(String nombre) - Elimina un alfabeto del sistema.
- eliminarListaDePalabras(String nombre) - Elimina una lista de palabras del sistema.
- eliminarTeclado(String nombre) - Elimina un teclado del sistema.
- editarTexto(String nombre, String nuevoTexto) - Edita un texto del sistema.
- editarAlfabeto(String nombre, String nuevoAlfabeto) - Edita un alfabeto del sistema.
- editarLista(String nombre, HashMap<String,Integer> nuevaLista) - Edita una lista del sistema.
- editarNombreTeclado(String nombre, String nuevoNombre): Edita el nombre de un teclado.
- editarNombreTexto(String nombre, String nuevoNombre): Edita el nombre de un texto.
- editarNombreAlfabeto(String nombre, String nuevoNombre): Edita el nombre de un alfabeto.
- editarNombreLista(String nombre, String nuevoNombre) - Edita el nombre de una lista.
- verTexto(String nombre, StringBuilder texto): Devuelve un texto del sistema.
- verAlfabeto(String nombre, StringBuilder texto): Devuelve un alfabeto del sistema.
- verLista(String nombre, HashMap<String,Integer> lista): Devuelve una lista de palabras con frecuencia del sistema.
- verTeclado(String nombre, ArrayList<ArrayList<Character>> teclado): Devuelve una matriz de caracteres que representa un teclado del sistema.
- listarAlfabetos(ArrayList<String> lista): Devuelve el nombre de todos los alfabetos que posee el usuario actual.
- listarTextos(ArrayList<String> lista): Devuelve el nombre de todos los textos que posee el usuario actual.
- listarListas(ArrayList<String> lista): Devuelve el nombre de todas las listas de palabras con frecuencia que posee el usuario actual.
- listarTeclados(ArrayList<String> lista): Devuelve el nombre de todos los teclados que posee el usuario actual.
- importarTexto(String path, String nombre): Importa un texto y se lo agrega al usuario.
- importarAlfabeto(String path, String nombre): Importa un alfabeto y se lo agrega al usuario.
- importarLista(String path, String nombre): Importa una lista de palabras con frecuencias y se la agrega al usuario.
- importarTecladoConLista(String path, String nombreTeclado, String nombreAlfabeto, String nombreLista): Importa un teclado, su alfabeto y su lista de palabras con frecuencia y se la agrega al usuario.
- importarTecladoConTexto(String path, String nombreTeclado, String nombreAlfabeto, String nombreTexto): Importa un teclado, su alfabeto y su texto y se la agrega al usuario.
- exportarTexto(String path, String nombre): Exporta un texto del sistema, generando un archivo, que contiene el texto.
- exportarAlfabeto(String path, String nombre): Exporta un alfabeto del sistema, generando un archivo, que contiene el alfabeto.
- exportarLista(String path, String nombre): Exporta una lista de palabras con frecuencia del sistema, generando un archivo, que contiene la lista.

- exportarTeclado(String path, String nombre): Exporta un teclado del sistema, generando un archivo, que contiene el teclado, su alfabeto y texto o lista.

Clase Algoritmo

Descripción: La clase abstracta Algoritmo representa un marco genérico para implementar teclados dado un alfabeto y una lista de palabras con sus respectivas frecuencias. Está diseñada para ser extendida por algoritmos específicos.

Atributos:

- **String alfabeto:** Este atributo almacena el alfabeto sobre el cual se aplicará el algoritmo.
- **HashMap<String,Integer> palabras:** Un mapa que contiene palabras y su frecuencia asociada después de aplicar el algoritmo.

Métodos:

Métodos de Acceso(Getters)

- getLayout(): Método abstracto que debe ser implementado por las clases hijas. Devuelve la distribución de teclado después de aplicar el algoritmo correspondiente.

Clase QAP

Descripción: La clase QAP representa una implementación especializada para resolver problemas de asignación cuadrática (QAP). Derivada de la clase abstracta Algoritmo, esta clase se centra en la resolución de problemas que involucran la asignación de ubicaciones y la minimización de costos asociados con flujos y distancias. En nuestro caso genera el layout de un teclado.

Atributos:

- **Integer ubicaciones:** Número total de ubicaciones en el problema, calculado como la raíz cuadrada del tamaño del alfabeto.
- **Integer sizeLayout:** Dimensión de la matriz de distribución de teclado, basada en la raíz cuadrada del tamaño del alfabeto.
- **Integer mejorCosteActual:** Almacena el mejor costo logrado durante la ejecución del algoritmo.
- **Integer nodos:** Contador de nodos explorados durante el proceso de búsqueda.
- **Long inicio:** Marca de tiempo que ayuda a controlar el tiempo de ejecución del algoritmo.
- **Int[][] flujo:** Matriz que almacena los flujos entre ubicaciones.
- **Int[][] distancias:** Matriz que representa las distancias entre ubicaciones.
- **Char[][] layout:** Matriz que finalmente contendrá la distribución de teclado.
- **Int[][] mejorSolucionActual:** Arreglo que guarda la mejor solución encontrada.
- **String alfabetoExtendido:** Alfabeto ampliado para asegurar que sea al menos del tamaño de la matriz de distribución de teclado.

Métodos:

Métodos de Acceso(Getters)

- getLayout(): Método heredado de la clase Algoritmo que devuelve la distribución de teclado después de aplicar el algoritmo específico.
- getFrecuencias(): Genera la matriz de flujo a partir de la lista de palabras con frecuencias.

- getDistancias(): Crea la matriz de distancias basada en el tamaño de la distribución de teclado.
- getSolucionInical(): Genera una solución inicial aleatoria para iniciar la exploración del espacio de búsqueda.

Métodos de Manipulación de Datos

- resolver(): Inicializa parámetros y comienza la exploración del espacio de búsqueda utilizando el algoritmo de branch&bound.
- exploracion(int costeActual, int sizeSolucionActual, int[] solucionActual, boolean[] yaEnSolucion) Implementa el algoritmo de branch&bound, generando y podando soluciones parciales.
- cotaInferiorParaSolucionParcial(int sizeSolucionParcial, boolean[] yaEnSolucion, int[][] c1): Calcula la cota inferior para podar soluciones parciales.
- evitarPermutaciones(int indice, boolean[] evitarPermutaciones): Evita ciertas permutaciones para reducir el espacio de búsqueda.

Clase Hungarian

Descripción: La clase Hungarian implementa el algoritmo para resolver el problema de asignación óptima. Se utiliza para encontrar la asignación de coste mínimo en una matriz de valores dada, donde cada celda representa el coste de asignar una tarea a un trabajador.

Atributos:

- **int[] filas**: Almacena la asignación de columnas a cada fila.
- **int[] columnasOcupadas:** Indica las columnas que ya están asignadas.
- **int[][] matrizOriginal:** Matriz original de costes.
- **int[][] valores:** Copia de la matriz original para realizar operaciones.
- **int[][] lineas:** Matriz que registra las líneas dibujadas durante el proceso.
- **int numLineas:** Número de líneas dibujadas.

Métodos:

Métodos de Acceso(Getters)

- getResult(): Obtiene la asignación final de columnas a filas.
- getTotal(): Obtiene la suma total de los costes de la asignación óptima.

Métodos principales

- restarFilaMinima(): Resta el valor mínimo de cada fila a todos los elementos de esa fila.
- restarColumnaMinima(): Resta el valor mínimo de cada columna a todos los elementos de esa columna.
- cubrirCeros(): Cubre los ceros dibujando líneas en direcciones específicas.
- maxVH(int fila, int columna): Determina si la línea a dibujar debe ser vertical u horizontal.
- pintarVecino(int fila, int col, int maxVH): Pinta los vecinos de una celda según la dirección dada.
- crearCerosAdicionales(): Crea ceros adicionales pintando el mínimo valor de celdas no cubiertas.
- optimization(int fila): Realiza la optimización de asignación de fila a columna.
- optimization(): Sobrecarga del método de optimización para iniciar desde la fila 0.
- cloneMatrix(int[][] matrix): Realiza la copia de una matriz dada.

Clase ALG2

Descripción: La clase ALG2 implementa un algoritmo de optimización conocido como Simulated Annealing para encontrar una distribución eficiente de teclado.

Atributos:

- **solucion**: Un arreglo de enteros que representa la solución actual generada por el algoritmo.
- **temp**: Temperatura inicial para el algoritmo de recocido simulado.
- **ratioEnfriamiento**: La tasa de enfriamiento del sistema durante el algoritmo.
- **iteracion**: Número de iteraciones por paso del algoritmo.

Métodos:

- **Constructor ALG2(String alfabeto, HashMap<String,Integer> palabras)):** Constructor de la clase ALG2 que inicializa la solución inicial aleatoria y comienza la exploración del espacio de búsqueda. Invoca al constructor de la clase base Algoritmo con un alfabeto y un mapa de palabras. Genera una solución inicial aleatoria mediante el método getSolucionInical(). Inicia la resolución del problema con el método resolver().
- **getSolucionInical():** Genera una solución inicial aleatoria para establecer un punto base de costo mínimo. Calcula las sumas de flujo y distancias para cada fila de las matrices correspondientes. Asigna teclas a posiciones basadas en la suma de flujo y la suma de distancias. Establece la solución inicial en el atributo solucion.
- **getMaxIndex(int[] array):** Devuelve el índice del elemento con el valor máximo en un array. Retorna el Índice del elemento con el valor máximo o -1 si el array está vacío.
- **getMinIndex(int[] array):** Devuelve el índice del elemento con el valor mínimo en un array. Devuelve el Índice del elemento con el valor mínimo o -1 si el array está vacío.
- **resolver(List<Integer>solucionActual):** Aplica el algoritmo de Simulated Annealing (Recocido Simulado). Itera a través del algoritmo de Simulated Annealing actualizando la solución. Utiliza la función probabilidadAceptar() para decidir si se acepta o no la nueva solución.
- **probabilidadAceptar(int costeActual, int costeVecino, double temperatura):** Devuelve la probabilidad de aceptar una nueva solución. Retorna la probabilidad de aceptar la nueva solución basada en la diferencia de coste y la temperatura.
- **getLayout():** Devuelve la distribución de teclado después de aplicar el algoritmo correspondiente. Devuelve la matriz de caracteres que representa la distribución de teclado.

3. ESTRUCTURAS DE DATOS Y ALGORITMOS DOMINIO

3.1. Elección de estructuras de datos de dominio

Clase Usuario

HashMap para Mapa de Teclados, Alfabetos, Textos y Listas.

Atributos

```
HashMap<String, Integer> mapaTeclados;

HashMap<String, Integer> mapaAlfabetos;

HashMap<String, Integer> mapaTextos;
```

HashMap<String, Integer> mapaListasPalabras;

En un principio implementamos cuatro ArrayList de id 's para cada colección de objetos en propiedad del usuario. Después de darnos cuenta de que hacíamos accesos continuados a esta información decidimos implementar una estructura de datos que nos facilitara un coste de búsqueda inferior al del ArrayList. Nos decidimos por un HashMap no solo por el acceso eficiente a través del nombre para buscar, insertar o eliminar teclados, alfabetos, textos y listas asociadas al usuario sino también porque de esta manera nos ahorramos tener un atributo nombre en las clases de teclado, alfabeto, texto y lista de palabras. La complejidad promedio es O(1) en las operaciones, crucial para manipulaciones frecuentes.

Clase Error

Uso de una enumeración para diferenciar los tipos de errores.

Atributo

typeError te;

Empezamos implementando excepciones en el código pero daban una imagen brusca y poco elegante en la estructura del código, por ello nos acabamos decantando por una enumeración de errores para controlar el flujo de la ejecución. Proporciona un conjunto claro y legible de tipos de error con mensajes asociados. Facilita la identificación y gestión de errores en el sistema.

Clase Alfabeto

ArrayList para id ‘s de Teclados Asociados.

Atributo

ArrayList<Integer> idsTecladosAsociados;

Durante el desarrollo del proyecto nos dimos cuenta de la necesidad de tener una estructura que almacenara de manera eficiente los identificadores de teclados asociados al alfabeto debido a que puede generar conflictos a la hora de hacer diversas operaciones en el alfabeto. Además las operaciones de búsqueda y manipulación son poco frecuentes en este contexto, por lo que el impacto de la complejidad lineal es mínimo.

Clase Conjunto De Alfabetos

HashMap para Almacenar Alfabetos.

Atributo

HashMap<Integer, Alfabeto> alfabetos;

La elección de utilizar un HashMap<Integer, Alfabeto> como atributo en la clase Conjunto De Alfabetos se fundamenta en la necesidad de almacenar y gestionar múltiples alfabetos asociados al usuario. Este enfoque proporciona un acceso eficiente a través de identificadores únicos, permitiendo la rápida recuperación de alfabetos específicos.

La complejidad promedio de las operaciones de búsqueda y manipulación se mantiene en O(1), lo que es crucial dado que estas operaciones son frecuentes en el contexto de un conjunto de alfabetos. Esta elección optimiza el rendimiento del sistema al minimizar el tiempo de búsqueda y asegurar una gestión eficiente de los alfabetos asociados al usuario.

Clase Texto

ArrayList para id ‘s de Teclados Asociados.

Atributo

```
ArrayList<Integer> idsTecladosAsociados;
```

Nos percatamos de la necesidad de guardar los id ‘s de los teclados igual que en la clase alfabeto. El uso del ArrayList permite un almacenamiento eficiente de los identificadores de teclados asociados. Dado que las operaciones de búsqueda en este contexto son raras, se minimiza el impacto de la complejidad lineal, manteniendo un equilibrio adecuado entre el espacio de almacenamiento y la eficiencia operativa.

Clase Conjunto De Textos

HashMap para Almacenar Textos.

Atributo

```
HashMap<Integer, Texto> textos;
```

El uso de HashMap<Integer, Texto> como atributo textos en la clase Conjunto De Textos permite un acceso eficiente a los textos a través de identificadores únicos. Con operaciones frecuentes de búsqueda y manipulación, se mantiene una complejidad promedio de O(1), asegurando un rendimiento óptimo.

Clase Lista De Palabras

ArrayList para id ‘s de Teclados Asociados.

Atributo

```
ArrayList<Integer> idsTecladosAsociados;
```

El atributo ArrayList<Integer> idsTecladosAsociados en la clase Lista De Palabras se justifica por su uso específico en el contexto de almacenar identificadores de teclados asociados a la lista de palabras al igual que en la clase alfabeto y texto. La elección de un ArrayList se basa en la naturaleza de las operaciones, donde las búsquedas son poco frecuentes y se prioriza la eficiencia en términos de acceso secuencial. El uso de un ArrayList minimiza el impacto de la complejidad lineal en este contexto, ofreciendo un rendimiento eficiente para las operaciones asociadas.

Clase Conjunto De Listas

HashMap para Almacenar Listas de Palabras.

Atributo

```
HashMap<Integer, String> listasDePalabras;
```

El atributo HashMap<Integer, String> listasDePalabras en la clase Conjunto De Listas se fundamenta en un HashMap para asegurar un acceso eficiente a las listas de palabras mediante identificadores únicos. Este enfoque se justifica por la frecuencia de operaciones de búsqueda y manipulación, donde la complejidad promedio de O(1) se mantiene constante, proporcionando un rendimiento óptimo para estas operaciones. La elección de esta estructura de datos facilita la gestión eficaz de las listas de palabras en el contexto de la aplicación.

Clase Teclado

Matriz Bidimensional para Layout del Teclado.

Atributo

char[][] layout;

La elección de la estructura de datos char[][] para el atributo layout en la clase Teclado se justifica por su eficiencia en representar la disposición física de las teclas. La matriz bidimensional permite un acceso constante (O(1)) a la disposición de una tecla específica a través de índices en la matriz. Esto asegura un rendimiento óptimo para las operaciones de acceso y manipulación de la disposición del teclado, proporcionando una solución eficiente y directa para representar la información clave asociada a un teclado.

ArrayList para IDs de Alfabeto, Texto y Lista Asociados.

Atributos

Integer idAlfabeto;

Integer idTexto;

Integer idLista;

La elección de utilizar los atributos idAlfabeto, idTexto e idLista como variables individuales en la clase Teclado permite almacenar eficientemente los identificadores asociados al teclado. Al utilizar variables individuales en lugar de estructuras más complejas, se minimiza el impacto de la complejidad lineal, ya que las operaciones de búsqueda son raras en este contexto. Esto garantiza un acceso eficiente a la información asociada al teclado, contribuyendo a la simplicidad y claridad del diseño de la clase.

Clase Conjunto De Teclados

HashMap para Almacenar Teclados.

Atributo

HashMap<Integer, Teclado> teclados;

La elección de utilizar un HashMap<Integer, Teclado> en la clase Conjunto Teclados para almacenar teclados proporciona un acceso eficiente a través de identificadores únicos. Dado que las operaciones de búsqueda y manipulación son frecuentes en este contexto, el uso de un HashMap garantiza una complejidad promedio de O(1), lo que mejora la eficiencia en la gestión y recuperación de teclados en el conjunto. Esto contribuye a un rendimiento óptimo del sistema al reducir los tiempos de búsqueda y manipulación de datos asociados a teclados.

Clase Controlador 1

Uso de Clases ConjuntoDeAlfabetos, ConjuntoDeTextos, ConjuntoDeListas, y HashMaps.

Atributos

CjtAlfabeto alfabetos;

CjtTeclado teclados;

CjtListaPalabras listasDePalabras;

CjtTextos textos;

El uso de las clases ConjuntoDeAlfabetos, ConjuntoDeTeclados, ConjuntoDeListas y ConjuntoDeTextos, junto con HashMaps, en la clase Controlador1 proporciona una eficiente gestión de conjuntos de alfabetos, teclados, listas de palabras y textos. Al utilizar estas estructuras de datos, se facilita la realización de operaciones frecuentes de búsqueda y manipulación con una complejidad promedio de O(1), lo cual garantiza un rendimiento óptimo en la gestión de los elementos mencionados, contribuyendo así a la eficiencia general del sistema.

Clase Controlador De Dominio

Uso de Utilidad, persistenciaFake y HashMap para Asociación Nombre-Id.

Atributos

Utilidad util;

persistencia persis;

HashMap<String, Integer> usernameId;

El uso de la clase Utilidad, persis y HashMap en la clase ControladorDeDominio se justifica por su contribución a un acceso eficiente y optimizado a datos críticos y operaciones esenciales. El HashMap<String, Integer>usernameId permite asociar de manera rápida y eficiente los nombres de usuario con sus identificadores correspondientes, facilitando búsquedas y accesos directos en operaciones clave para la autenticación y manipulación de usuarios. En un principio se implementó una clase entera de Conjunto De Usuarios que no solo mantenía este mapa si no que también gestionaba toda la información de los usuarios registrados, luego nos percatamos de que solo podíamos tener un usuario concurrente por lo que se decidió eliminar la clase y sustituirla por este HashMap en Controlador de Dominio.

La presencia de la clase Utilidad en los atributos asegura la disponibilidad de funciones auxiliares y comprobaciones esenciales para el controlador de dominio. Estas funciones pueden abordar aspectos críticos, como la validación de contraseñas y nombres de usuario, contribuyendo a la coherencia y seguridad del sistema.

Y por último persis que es una instancia de persistencia. Que se utiliza para guardar o leer archivos en disco, lo que nos permite mantener (persistir) la información de cada usuario.

En resumen, la combinación de estas estructuras y clases en la clase ControladorDeDominio se traduce en un diseño eficiente y técnico, asegurando un manejo óptimo de operaciones críticas y una mayor flexibilidad en el desarrollo y prueba del sistema.

3.2. Algoritmos de diseño de teclado

3.2.1. Implementación algoritmo QAP

Problema

Dado un alfabeto y un texto o lista de palabras con frecuencia debemos generar una distribución de teclado óptima para que pueda ser usado con un solo dedo.

Para ello se nos propone usar un algoritmo de QAP, el cual por sus siglas en inglés significa Quadratic Assignment Problem (QAP) y éste se trata de asignar N instalaciones a una cantidad N de sitios o locaciones en donde se considera un costo asociado a cada una de las asignaciones. Este costo dependerá de las distancias y flujo entre las instalaciones. De este modo se buscará que este costo, en función de la distancia y flujo, sea mínimo.

Y se nos propone la resolución del mismo implementando un algoritmo de branch & bound.

Modelo

Como se mencionó anteriormente, nuestro problema trata de asignar N instalaciones a una cantidad N de sitios, en nuestro caso las instalaciones se traducen a letras del alfabeto y los sitios a posiciones de una matriz cuadrada que representará la distribución del teclado.

También tendremos dos matrices, la matriz de flujo (F) y la matriz de distancias (D), que en este problema, F representa la frecuencia entre pares de letras obtenidas del texto o lista de palabras con frecuencias y D la distancia entre posiciones de la matriz que representa nuestro teclado, ambas matrices son simétricas.

El coste de un asignación dependerá de ambas matrices, se calcula de la siguiente manera:

$$\text{cost} = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij}\, d_{p(i)p(j)}$$

$F = [f_{ij}]$ Representa el coste entre las letras i y j.

$D = [d_{ij}]$ Representa la distancia entre las ubicaciones i y j.

$p(i)$ La posición en la que está asignada la letra i

Siendo el objetivo de nuestro problema encontrar el coste mínimo entre todas las posibles asignaciones.

<u>Gráficamente:</u>

Freq

	a	b	c	d	e	f	g	h	i
a	0	98	45	0	0	59	56	7	0
b	98	0	0	0	40	0	0	0	3
c	45	0	0	0	0	0	0	13	5
d	0	0	0	0	0	0	0	0	0
e	0	40	0	0	0	4	0	2	0
f	59	0	0	0	4	0	0	0	9
g	56	0	0	0	0	0	0	0	53
h	7	0	13	0	2	0	0	0	54
i	0	3	5	0	0	9	53	54	0

Dist

	0	1	2	3	4	5	6	7	8
0	0	10	20	10	14	22	20	22	28
1	10	0	10	14	10	14	22	20	22
2	20	10	0	22	14	10	28	22	20
3	10	14	22	0	10	20	10	14	22
4	14	10	14	10	0	10	14	10	14
5	22	14	10	20	10	0	22	14	10
6	20	22	28	10	14	22	0	10	20
7	22	20	22	14	10	14	10	0	10
8	28	22	20	22	14	10	20	10	0

En el ejemplo anterior observamos la frecuencia entre letras, como puede ser frecuencia ab=98, y en la matriz de distancias podemos las distancias entre posiciones del layout (distribución) del teclado.

Layout		
0	1	2
3	4	5
6	7	8

por ejemplo la distancia 08=28, que se ve claramente en la imagen de arriba.

Así quedan definidas las estructuras relacionadas exclusivamente al QAP, a continuación definimos las relacionadas al branch & bound.

Para representar nuestra solución, es decir, nuestra distribución de teclado, utilizaremos un array de números enteros, donde cada elemento del array, representa la posición en la que se colocará la letra con el mismo índice dentro del alfabeto.

Ejemplo:

Alfabeto	a	b	c	d	e	f	g	h	i
Solución	0	7	1	5	4	6	3	2	8

Es decir, la Solución[0] representa la posición en el layout de a. La permutación de esta solución daría la siguiente distribución.

Layout		
a	c	h
g	e	d
f	b	i

Layout		
0	1	2
3	4	5
6	7	8

El teclado generado siempre será de un tamaño mxm, siendo éste el mínimo posible para contener la cantidad de símbolos del alfabeto. Los símbolos sobrantes se representarán como un espacio, por lo tanto no se verán en la solución.

Algoritmo

Branch & bound:

Nuestro objetivo es encontrar la asignación con el menor coste entre todas las posibles asignaciones. Para ello se nos propuso el uso de un algoritmo de branch & bound, el cual es un algoritmo de fuerza bruta, que nos permite recorrer el espacio de soluciones factibles evitando las ramas que no posean potencial, guardando en todo momento la mejor solución de todas, para que cuando termine el recorrido, haber obtenido la solución óptima (la mejor) a nuestro problema. El algoritmo consta de dos partes importantes que son:
Branching: La función del algoritmo que nos permite generar a partir de la solución actual el resto de soluciones posibles. Que en nuestro caso es típica estructura de backtracking donde antes de llamar al paso recursivo le agregamos a la solución una ubicación disponible a la próxima letra que hay que procesar. Luego del paso recursivo se deshace ese cambio y se asigna otra ubicación libre, hasta probar para esa letra todas las combinaciones posibles.
Bounding: Para cada solución posible que ha resultado del proceso de branching obtenemos una cuota de la mejor solución obtenible a partir de ese punto, para descartar aquellas soluciones posibles cuya cota sea peor que la mejor solución encontrada hasta ese momento. Es decir que el bounding, es la función del algoritmo que nos permite podar los caminos sin potencial, evitando recorrer una gran cantidad del espacio de soluciones. En este caso en particular se usa la cota propuesta, la cota de Gilmore-Lawler.

Cota Gilmore-Lawler:

La cota de GIlmore-Lawler es un tipo de cota inferior que se basa en el hecho de que se puede obtener una cota inferior para el producto escalar de dos vectores a y b multiplicando el elemento más grande de a por el elemento más pequeño de b, el segundo elemento más grande de a por el segundo elemento más pequeño de b, etc.

Dada una solución parcial, es decir, una solución con N letras ya asignadas, la cota se calcula como la adición de tres términos.

Término 1:

Simplemente hace referencia al coste entre las letras ya asignadas. En nuestro código esto hace referencia a la variable de costeActual, que en cada iteración calculamos el incremento de coste de una asignar una ubicación a una letra y simplemente sumamos ese incremento al coste total.

Término 2:

El segundo término aproxima el coste del tráfico entre las instalaciones colocadas y las que quedan por colocar.

Término 3:

Contempla el coste del tráfico entre las instalaciones aún no emplazadas.

Para estimar los valores del segundo y del tercer término lo que hacemos es construir una matriz de dimensión (N − m) × (N − m) donde el elemento (i, k) intenta acotar el coste adicional que supondría colocar la i´ésima letra no emplazada en la k´ésima ubicación. Una vez la tenemos, intentamos encontrar la asignación óptima para las letras no emplazadas y el coste de esta se usa como estimador de la suma del segundo y tercer términos. Esta matriz se puede calcular como la suma de dos matrices C1 y C2.

C1:

En C1 el elemento (i, k) representa el coste de colocar la i´ésima letra no emplazada en la k´ésima ubicación, con respecto a las letras ya emplazadas. En nuestro código calculamos esto al mismo tiempo que calculamos el incremento de coste que supondrá la asignación de la próxima letra en todas las ubicaciones restantes, ya que si nos damos cuenta, este cálculo representa una de las filas de esta matriz, entonces simplemente la extendemos al resto de letras.

```
int ubicacionesFaltantantes = ubicaciones - sizeSolucionActual;
int[][] c1  = new int[ubicacionesFaltantantes][ubicacionesFaltantantes];

ArrayList<Map.Entry<Integer,Integer>> incrementosCoste = new ArrayList<>();

for (int k=sizeSolucionActual; k  < ubicaciones; ++k) {
    for (Integer i = 0, indiceFila = 0; i < ubicaciones; i++) {
        if (!yaEnSolucion[i]) {
            Integer incrementoCoste = 0;
            for (int j = 0; j < sizeSolucionActual; j++) {
                incrementoCoste += 2 * distancias[solucionActual[j]][i] * flujo[j][k];
            }
            if (k == sizeSolucionActual)
                incrementosCoste.add(new AbstractMap.SimpleEntry<>(i, incrementoCoste) );
            int fila = k-sizeSolucionActual;
            c1[fila][indiceFila] = incrementoCoste;
            ++indiceFila;
        }
    }
}
```

La parte que calcula el incrementoCoste, el índice k que se utiliza en la matriz de flujo representa a qué letra se le está calculando el incremento de coste. También observamos que si la k coincide con el tamaño de la

solución significa que esta será la próxima letra a asignar a si que aprovechamos el recorrido para calcular el incremento de la misma.

C2:

En C2 el elemento (i, k) representa el coste de colocar la i´ésima letra no emplazada en la k´ésima ubicación, con respecto a las instalaciones aún no emplazadas. Sin embargo, los elementos de esta matriz no se pueden calcular de manera exacta y se han de aproximar. Sea T el vector de frecuencias desde la i´ésima letra al resto (de las no emplazadas) y sea D el vector de distancias desde la k´ésima ubicación al resto (de las no ocupadas); la cota inferior del coste de emplazar i en k resulta del producto escalar del T ordenado crecientemente con D ordenado decrecientemente. En nuestro caso concreto lo que hicimos fue crear un nuevo par de matrices de flujo y distancias, como las letras se asignan en el mismo orden que el de la matriz, es nueva matriz se obtiene de manera directa, ya que es simplemente la parte de la matriz que falta por se colocada, y en el caso de la matriz de distancias se complica un poco, ya que cada letra puede tomar cualquiera de la ubicaciones, así que para su obtención, recorremos la matriz original omitiendo la filas y la columnas de las ubicaciones ya utilizadas.

En nuestro código, este apartado se representa por la función cotaInferiorSolucionParcial.

```
private int cotaInferiorParaSolucionParcial(int sizeSolucionParcial, boolean[] yaEnSolucion, int[][] c1) {
    int ubicacionesFaltantes = ubicaciones - sizeSolucionParcial;
    int[][] nuevoFlujo = new int[ubicacionesFaltantes][ubicacionesFaltantes];
    int[][] nuevoDistancia = new int[ubicacionesFaltantes][ubicacionesFaltantes];

    for (int i = sizeSolucionParcial,indiceFila=0; i < ubicaciones; i++) {
        for (int j = sizeSolucionParcial, indiceColumna=0; j < ubicaciones; j++) {
            nuevoFlujo[indiceFila][indiceColumna++] = flujo[i][j];
        }
        Arrays.sort(nuevoFlujo[i - sizeSolucionParcial]);
        ++indiceFila;
    }


    for (int i = 0, indiceFila=0; i < ubicaciones; i++) {
        if (yaEnSolucion[i])
            continue;

        for (int j = 0,indiceColumna = 0; j < ubicaciones; j++) {
            if (yaEnSolucion[j])
                continue;
            nuevoDistancia[indiceFila][indiceColumna] = distancias[i][j];
            indiceColumna++;
        }

        Integer[] subarray = Arrays.stream(nuevoDistancia[indiceFila]).boxed().toArray(Integer[]::new);
        Arrays.sort(subarray, 0, ubicacionesFaltantes, Collections.reverseOrder());
        nuevoDistancia[indiceFila] = Arrays.stream(subarray).mapToInt(Integer::intValue).toArray();
        indiceFila++;
    }
```

Si nos damos cuenta cada fila de esta matriz representan los vectores que se mencionan para el cálculo, solo que no ordenados de la manera deseada, así que simplemente ordenamos las filas, siguiendo el criterio anteriormente mencionado. Una vez ordenado es simplemente ir haciendo el producto escalar entre vectores para cada uno de los elementos de la matriz.

Obtención de las nuevas dos matrices y su ordenación correspondiente.

Creación de la matriz c2.

```
int[][] c2 = new int[ubicacionesFaltantes][ubicacionesFaltantes];

for(int i = 0; i < ubicacionesFaltantes; ++i)
    for(int j = 0; j < ubicacionesFaltantes; ++j)
        for(int k = 0; k < ubicacionesFaltantes-1; ++k)
            c2[i][j] += nuevoDistancia[j][k]*nuevoFlujo[i][k];

int[][] c1c2 = new int[ubicacionesFaltantes][ubicacionesFaltantes];
```

Luego de obtener c1 y c2, la suma de ambas matrices, se obtiene una nueva matriz entre letras y ubicaciones, y la mejor asignación de esta será el aproximado que utilizaremos para la obtención de la cota inferior.

Así que ahora el problema se convierte en un problema de Linear Assignment Problem (LAP), para la resolución de este, ya se conoce un algoritmo que logra obtener el óptimo en un tiempo razonable, este se llama Hungarian Algorithm.

Hungarian:

Todo el proceso se sustenta sobre el siguiente teorema: Si un número se añade o sustrae de todos los valores de un columna o fila en una matriz de costes, entonces.

Así que el algoritmo se puede resumir en 4 pasos:

1) Se le resta a cada elemento de una fila el valor mínimo de esa misma.

```java
/**
 * Restamos por cada elemento el minimo valor de su fila
 */
1 usage   nicolas.gonzalo.longueira
public void restarFilaMinima() {
    int[] rowMinValue = new int[valores.length];
    // gurdamos el minimo de cada fila en rowMinValue[]
    for (int row = 0; row < valores.length; row++) {
        rowMinValue[row] = valores[row][0];
        for (int col = 1; col < valores.length; col++) {
            if (valores[row][col] < rowMinValue[row])
                rowMinValue[row] = valores[row][col];
        }
    }

    //restamos el minimo de cada fila usando rowMinValue[]
    for (int row = 0; row < valores.length; row++) {
        for (int col = 0; col < valores.length; col++) {
            valores[row][col] -= rowMinValue[row];
        }
    }
}
```

2) Se le resta a cada elemento de una columna el valor mínimo de esa misma columna.

```java
/**
 * Recorremos todos los elementos y llamamos a la funcion pintarVecino cuando un elemento es igual a zero
 */
2 usages   nicolas.gonzalo.longueira
public void cubrirCeros() {
    numLineas = 0;
    lineas = new int[valores.length][valores.length];

    for (int row = 0; row < valores.length; row++) {
        for (int col = 0; col < valores.length; col++) {
            if (valores[row][col] == 0)
                pintarVecino(row, col, maxVH(row, col));
        }
    }
}
```

3) Se recorre la matriz y se identifican los ceros. Se pintan líneas en las filas y columnas que contienen ceros de manera que todos los ceros estén cubiertos y se utilice el menor número posible de líneas.


```java
/**
 * Restamos por cada elemento el minimo valor de su columna
 */
1 usage   nicolas.gonzalo.longueira
public void restarColumnaMinima() {
    int colMinValue[] = new int[valores.length];
```

```java
/**
 * Este paso no siempre se ejecuta. Fijarse en la construccion
 * Crea ceros adicionales, pintando el minimo valor de celdas no cubiertas
 */
1 usage   nicolas.gonzalo.longueira
public void crearCerosAdicionales() {
    int minUncoveredValue = 0;

    // Encontrar el minimo en los numeros no curbiertos
    for (int row = 0; row < valores.length; row++) {
        for (int col = 0; col < valores.length; col++) {
            if (lineas[row][col] == 0 && (valores[row][col] < minUncoveredValue || minUncoveredValue == 0))
                minUncoveredValue = valores[row][col];
        }
    }

    // Restarle el minimo a todos los elementos no cubiertos, y añadirlo a todos los elementos cubiertos dos veces
    for (int row = 0; row < valores.length; row++) {
        for (int col = 0; col < valores.length; col++) {
            if (lineas[row][col] == 0)
                valores[row][col] -= minUncoveredValue;

            else if (lineas[row][col] == 2)
                valores[row][col] += minUncoveredValue;
        }
    }
}
```

4) Mientras no se hayan cubierto todos los ceros en la matriz, se repiten los pasos 3 y 4 para crear ceros adicionales y cubrirlos.
5) Se realiza una optimización para asignar de manera única cada fila a una columna, garantizando que cada fila y columna estén asignadas exactamente una vez. Se utiliza un enfoque de fuerza bruta para explorar todas las asignaciones posibles, un backtracking.
6) Después de haber aplicado el algoritmo de backtracking ya tenemos nuestra mejor asignación y simplemente calculamos el coste de esta.

3.2.2. Implementación del segundo algoritmo

Debido que el problema que nos enfrentamos no cambia, en esencia, el problema sigue siendo un problema del tipo Quadratic Assignment Problem (QAP), así que lo que haremos, no es cambiar la forma de plantear el problema, sino, de cómo obtenemos la solución del QAP. En el anterior algoritmo la solución se conseguía a través del branch and bound que mencionamos anteriormente. Como vimos, este tenía muchos problemas en cuanto a velocidad, debido a que este intenta encontrar el óptimo (la mejor distribución), por lo que, para el enfoque de este algoritmo, intentaremos priorizar el tiempo para generar una solución en vez de su calidad, aunque manteniendo evidentemente un mínimo de esta. Para ello implementaremos un algoritmo de Simulated Annealing.

Modelo

Como se mencionó arriba, solo cambiaremos la forma de obtener la solución, por lo tanto el modelado del problema es exactamente el mismo que el del algoritmo anterior, por ello se había implementado una función abstracta como padre de los algoritmos QAP y ALG2, la cual se encarga de todo lo relacionado con el modelado del problema el cual se comparte entre ambos algoritmos.

Algoritmo

Simulated Annealing:

Simulated annealing es un algoritmo de optimización probabilística que se utiliza para encontrar soluciones aproximadas a problemas de optimización, en nuestro caso, encontrar un mínimo. Este algoritmo está inspirado en el proceso físico que ocurre cuando metales se mezclan, donde un material se calienta y luego se enfría lentamente para reducir las imperfecciones y alcanzar un estado de mínima energía.

En términos sencillos, el algoritmo de simulated annealing simula el proceso de calentamiento y enfriamiento para explorar el espacio de soluciones de un problema. Comienza con una solución inicial y realiza pequeños cambios aleatorios. A medida que el algoritmo avanza, la probabilidad de aceptar soluciones peores disminuye con el tiempo, de manera análoga a cómo la probabilidad de aceptar configuraciones de baja energía disminuye durante el enfriamiento en el proceso físico.

Este enfoque permite al algoritmo escapar de óptimos locales y buscar soluciones más amplias en el espacio de búsqueda.

Entonces los puntos principales del algoritmo son los siguientes:

1) Inicialización

Creamos una solución inicial que puede ser aleatoria, en nuestro caso decidimos implementar un pequeño algoritmo para intentar conseguir una solución con un coste inicial decente. El objetivo de nuestro algoritmo es el de colocar las letras más frecuentes en las posiciones con menor distancia al resto. Esto no necesariamente nos dará una buena solución, ya que las letras de mayor frecuencia no tienen porqué estar relacionadas entre sí, pero lo más probables es que lo estén y en términos generales, mantener la piezas de mayor frecuencia en posiciones centrales hará que el coste sea menor. Para lograr esto lo que hacemos es lo siguiente.

Creamos dos arrays que nos guardan la suma de las filas de matriz de flujo y distancias.

Java

```
        int size = distancias.length;

        // Creamos dos arrays, con la cantidad de filas que poseen las matrices de
        // flujo y distancia.
        int[] sumaFlujos = new int[size];
        int[] sumaDistancias = new int[size];

        //Creamos una array que contendrá la solución inicial
        solucion = new int[size];

        // Vamos colocando en el array, la suma total de cada fila de la matriz de
        // flujo
        for (int i = 0; i < size; i++) {
            int aux = 0;
            for (int j = 0; j < size; j++) {
                aux += flujo[i][j];
            }
            sumaFlujos[i] = aux;
        }

        // Vamos colocando en el array, la suma total de cada fila de la matriz de
        // distancias
        for (int i = 0; i < size; i++) {
            int aux = 0;
```

```
            for (int j = 0; j < size; j++) {
                aux += distancias[i][j];
            }
            sumaDistancias[i] = aux;
        }
```

Luego simplemente vamos construyendo la solución.

Java

```
        // Aquí creamos una solución inicial con un minimo de criterio, que es el
        // siguiente:
        // Buscamos la posicion que tiene la menor distancia con el resto y le
        // asignamos la tecla que posee el mayor flujo.
        // Eliminamos las mejores posciones obtenidas anteriormente y repetimos el
        // anterior paso hasta completar la solucion.
        // Esto no conseguirá una solucion muy buena, ya que el criterio anterior no
        // necesariamente significa que se agrupen
        // las teclas de buena manera, pero por el hecho de ir colocando las teclas de
        // mayor flujo en las mejores posciones
        // masomenos asegura que algunas coincidan.
        for (int i = 0; i < size; i++) {
            //Aquí es donde colocamos la posicion que tiene la menor distancia con el
            //resto y le asignamos la tecla que posee el mayor flujo.
            solucion[Math.abs(getMinIndex(sumaDistancias))] = getMaxIndex(sumaFlujos);
            //Aquí descartamos las mejores posiciones
            sumaDistancias[Math.abs(getMinIndex(sumaDistancias))] = 10000000;
            sumaFlujos[Math.abs(getMaxIndex(sumaFlujos))] = -1;
```

2) Iteración

Esta es la parte más importante del algoritmo, aquí es donde se busca la “mejor” solución vecina de la solución actual (recordar que el SA, cuanto mayor temperatura, más probabilidad de tomar una solución pero que la actual tiene, lo que le permite explorar más el espacio de solución y no estancarse en mínimos locales no tan buenos), aplicando los operadores, que en nuestro caso es simplemente un operador de swap entre dos posiciones aleatorias, posteriormente se compara las soluciones y bajo el criterio de la función de aceptación, se queda con una u otra.

Aqui el codigo de lo que realiza el bloque de iteración:

Java

```
       for(int i = 0 ; i < iteracion ; i++) {

                // Creamos una nueva lista para guardar la nueva solucion
                List<Integer> nuevaSolucion = new ArrayList<Integer>(solucionActual);
```

```
        // Instanciamos una clase random.
        Random rand = new Random();

        // Obtenemos dos posiciones posibles aleatorias.
        int pos1 = rand.nextInt(solucionActual.size());
        int pos2 = rand.nextInt(solucionActual.size());

        // En caso de haber obtenido la misma, buscamos una nueva posicion
        // hasta que sean distintas
        while (pos1 == pos2) {
            pos2 = rand.nextInt(solucionActual.size());
        }

        // Almacenamos los valores de las posiciones aleatorias obtenidas
        // anteriormente
        int swap1 = nuevaSolucion.get(pos1);
        int swap2 = nuevaSolucion.get(pos2);

        // Intercambiamos los valores de las posicones, nuestro unico
        // operador.
        nuevaSolucion.set(pos2, swap1);
        nuevaSolucion.set(pos1, swap2);

        // Convertimos la solucionActual en un array
        int[] csol = new int[solucionActual.size()];
        for (int j = 0; j < solucionActual.size(); j++) {
            csol[j] = solucionActual.get(j);
        }

        // Convertimos la nuevaSolucion en un array
        int[] nsol = new int[nuevaSolucion.size()];
        for (int j = 0; j < nuevaSolucion.size(); j++) {
            nsol[j] = nuevaSolucion.get(j);
        }

        // Obtenemos los costes de la soluciones
        int costeActual = super.getCoste(csol);
        int costeVecino = super.getCoste(nsol);

        // Obtenemos un número aleatorio para usarlo en la etapa de aceptación
        double random = rand.nextDouble();

        // Decidimos si debemos aceptar esta solucion.
        if ((probabilidadAceptar(costeActual, costeVecino, temp) > random))
            solucionActual = nuevaSolucion;

    }
```

Nosotros establecemos 1000 iteraciones por “paso” de temperatura. Esta asignación es producto de un pequeño trabajo estadístico realizado, para ver qué combinación de parámetros logra el mejor beneficio coste/tiempo.

3) Enfriamiento

El paso anterior se ejecuta una cantidad x de iteraciones, y las iteraciones se ejecutan hasta que el sistema se termine de enfriar, nosotros en vez de elegir un bucle externo con simplemente “steps”, lo que hacemos es ir decremento un 5% de la temperatura, hasta llegar a una temperatura <= 0.05.

```Java
while(temp > 0.05) {

	//Bloque de iteracion
	temp = temp * (1 - ratioEnfriamiento);
}
```

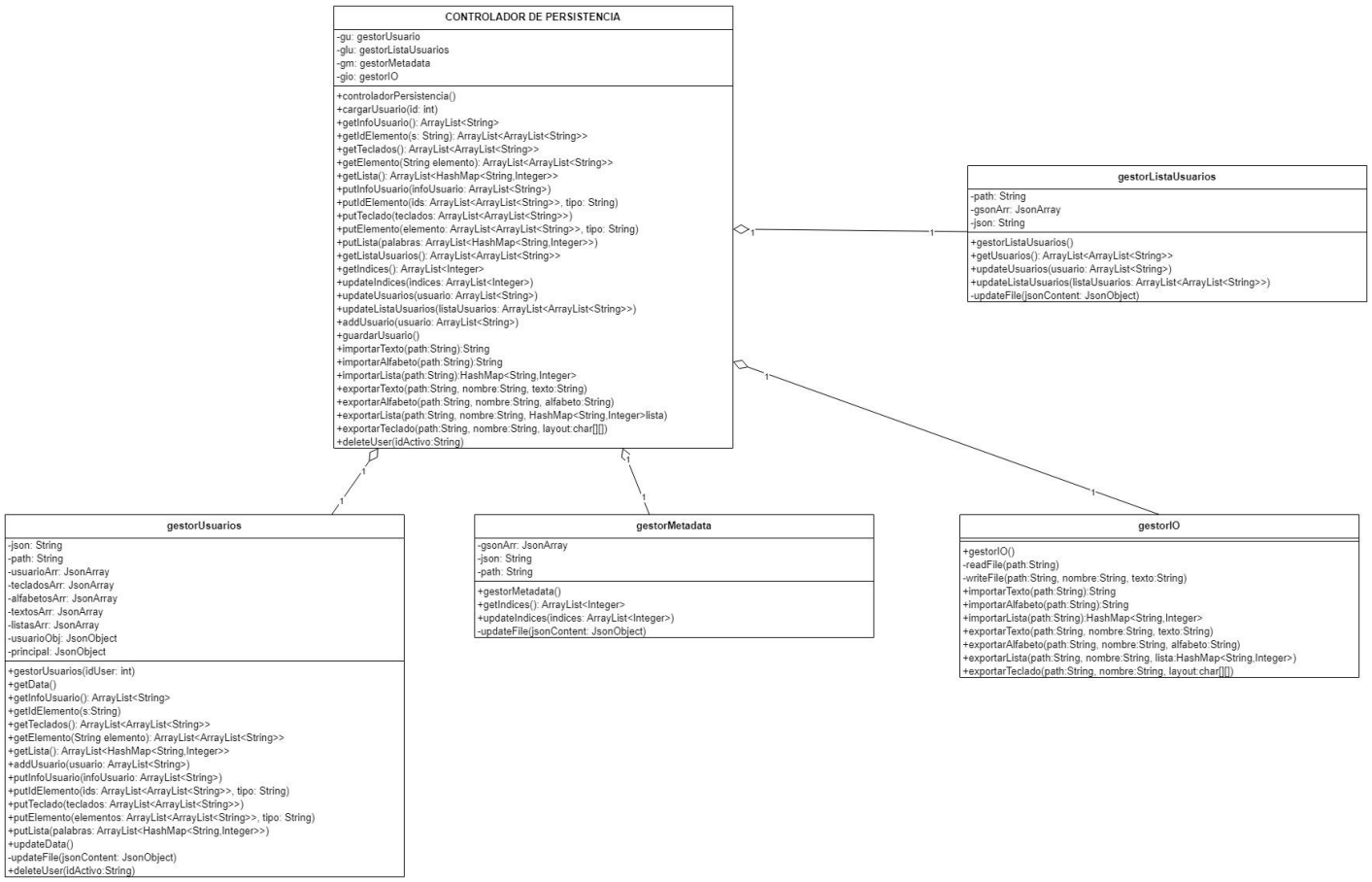
4) Función de aceptación

Por último la función de aceptación, que teniendo cuenta criterios como los costes de la solución vecina, la solución actual y la temperatura del sistema, genera la probabilidad de aceptación, para actualizar la solución actual por la vecina. En nuestro caso, si el coste de la solución vecina es mejor que la actual, la toma con un 100% de probabilidad y en caso contrario lo calculamos de la siguiente manera: p $= \frac{(solucionActual - solucionVecina)}{temperatura}$, siendo “p” la probabilidad de aceptación. Básicamente lo que queremos es aceptar cuanto menor sea la diferencia entre solución y/o mayor la temperatura del sistema, exactamente como el proceso físico del que surge la idea del SA.

El código de la función en cuestión:

```Java
    private double probabilidadAceptar(int costeActual, int costeVecino, double
temperatura) {
        if (costeVecino < costeActual)
            return 1.0;
        return Math.exp((costeActual - costeVecino) / temperatura);
    }
```

4. MODELO CONCEPTUAL PERSISTENCIA



4.1. Descripción de las clases de persistencia

Controlador de persistencia

Descripción: La clase controladorPersistencia actúa como un controlador central para la gestión de la persistencia de datos en la aplicación. Se encarga de coordinar la interacción entre los diferentes gestores de persistencia, como el gestor de usuarios, el gestor de listas de usuarios, el gestor de metadata, el gestor de I/O y el controlador de domino. Este simplemente recibe el comando de dominio o lo delega a los gestores.

Atributos:

- **gestorUsuarios gu:** Gestor encargado de manejar la persistencia de datos relacionados con usuarios.
- **gestorListaUsuarios glu:** Gestor encargado de manejar la persistencia de listas de usuarios.
- **gestorMetadata gm:** Gestor encargado de manejar la persistencia de datos de metadata.
- **gestorIO gio:** Gestor encargado de manejar las operaciones de entrada/salida.

Métodos:

Todas estas funciones se refieren a la información exportada o que se importará de disco.

- **controladorPersistencia()**: Constructor de la clase que inicializa los gestores relacionados con la persistencia.
- **void cargarUsuario(int id):** Inicializa una instancia de gestorUsuarios con el identificador de usuario proporcionado.
- **ArrayList<String> getInfoUsuario():** Obtiene la información pertinente a un usuario y la devuelve en un formato preestablecido.
- **ArrayList<ArrayList<String>> getIdElemento(String tipo)**: Obtiene los identificadores de un tipo de elemento del usuario.
- **ArrayList<ArrayList<String>> getTeclados():** Obtiene la información de todos los teclados de un usuario.
- **ArrayList<ArrayList<String>> getElemento(String elemento):** Obtiene la información de todos los elementos de un tipo específico de un usuario.
- **ArrayList<HashMap<String, Integer>> getLista():** Obtiene el conjunto de listas de palabras de un usuario.
- **void putInfoUsuario(ArrayList<String> infoUsuario):** Modifica la información pertinente a un usuario.
- **void putIdElemento(ArrayList<ArrayList<String>> ids, String tipo):** Actualiza los identificadores de un tipo de elemento para un usuario.
- **void putTeclado(ArrayList<ArrayList<String>> teclados):** Actualiza los teclados de un usuario.
- **void putElemento(ArrayList<ArrayList<String>> elemento, String tipo):** Actualiza los elementos de un tipo específico del usuario.
- **void putLista(ArrayList<HashMap<String, Integer>> palabras):** Actualiza las listas de palabras del usuario.
- **ArrayList<ArrayList<String>> getListaUsuarios():** Obtiene una lista de los usuarios y sus identificadores.
- **ArrayList<Integer> getIndices():** Obtiene los índices de metadata.
- **void updateIndices(ArrayList<Integer> indicies):** Actualiza los índices de metadata.
- **void updateUsuarios(ArrayList<String> usuario):** Agrega un usuario a la lista de usuarios.
- **void updateListaUsuarios(ArrayList<ArrayList<String>> listaUsuarios):** Actualiza la lista de usuarios.
- **void addUsuario(ArrayList<String> usuario)**: Crea un nuevo usuario con su información.
- **void guardarUsuario():** Actualiza el archivo en disco correspondiente al usuario.
- **String importarTexto(String path):** Extrae el texto de un archivo.
- **String importarAlfabeto(String path):** Extrae el alfabeto de un archivo.
- **HashMap<String, Integer> importarLista(String path):** Extrae una lista de palabras de un archivo.
- **void exportarTexto(String path, String nombre, String texto):** Genera un archivo con un texto.
- **void exportarAlfabeto(String path, String nombre, String alfabeto):** Genera un archivo con un alfabeto.
- **void exportarLista(String path, String nombre, HashMap<String, Integer> lista):** Genera un archivo con una lista de palabras.
- **void exportarTeclado(String path, String nombre, char[][] layout):** Genera un archivo con un teclado.

Gestor de usuarios

Descripción: Aquí es donde se realiza todo el trabajo referente a la escritura y lectura de disco de la información relacionada con un usuario, que en nuestro caso es id, username, password, email, identificador de los elementos, y los elementos. En el momento en el que un usuario se registre o ingrese, en esta función se cargará en memoria la información de este usuario y cuando el usuario realice un logout esta información se reemplazará por la última información de este usuario carga en memoria de dominó y se guardará en disco. El usuario se guarda en disco con el nombre “usuario_x”, siendo x el identificador de este mismo.

Atributos:

- **String json:** Cadena de texto que representa el json que posteriormente se guardará en disco..
- **String path:** Dirección en la que se almacenará el usuario en disco.
- **JsonArray usuarioArr:** JsonArray que contiene un conjunto de los usuarios de este archivo, que siempre será un único usuario.
- **JsonArray tecladosArr:** JsonArray que contiene un conjunto de los teclados del usuario.

- **JsonArray alfabetosArr:** JsonArray que contiene un conjunto de los alfabetos del usuario.
- **JsonArray textosArr:** JsonArray que contiene un conjunto de los textos del usuario.
- **JsonArray listasArr:** JsonArray que contiene un conjunto de las listas de palabras con frecuencia del usuario.
- **JsonObject usuaroObj:** El único objeto que contiene el array de usuarios del json, y el que contiene la información referente al usuario.
- **JsonObject principal:** Objeto final que se guarda a disco que contiene todos los arrays anteriormente mencionados.

Métodos:

- **gestorUsuarios(int idUser):** Constructor que inicializa la clase con la ruta del archivo del usuario y carga o crea el archivo según sea necesario.
- **getData():** Obtiene la información del usuario desde el archivo JSON y asigna los valores a los atributos correspondientes.
- **getInfoUsuario():** Retorna un ArrayList con la información básica del usuario (id, nombre de usuario, contraseña y correo).
- **getIdElemento(String s):** Retorna un ArrayList con información detallada de los elementos asociados al usuario (teclados, alfabetos, textos, listas) identificados por la cadena s.
- **getTeclados():** Retorna un ArrayList con información detallada de los teclados asociados al usuario.
- **getElemento(String elemento):** Retorna un ArrayList con información detallada de los elementos (teclados, alfabetos, textos, listas) identificados por la cadena elemento.
- **getLista():** Retorna un ArrayList con información detallada de las listas asociadas al usuario.
- **addUsuario(ArrayList<String> usuario):** Añade un nuevo usuario con la información proporcionada al archivo JSON.
- **putInfoUsuario(ArrayList<String> infoUsuario):** Actualiza la información del usuario en el objeto usuarioObj.
- **putIdElemento(ArrayList<ArrayList<String>> ids, String tipo):** Actualiza la información de los elementos (teclados, alfabetos, textos, listas) asociados al usuario en el objeto usuarioObj.
- **putTeclado(ArrayList<ArrayList<String>> teclados):** Actualiza la información de los teclados asociados al usuario en el objeto tecladosArr.
- **putElemento(ArrayList<ArrayList<String>> elementos, String tipo):** Actualiza la información de los elementos (teclados, alfabetos, textos, listas) en el objeto correspondiente (tecladosArr, alfabetosArr, textosArr, listasArr).
- **putLista(ArrayList<HashMap<String, Integer>> palabras):** Actualiza la información de las listas en el objeto listasArr.
- **updateData():** Actualiza el objeto principal con los datos actuales y llama a updateFile() para escribir la información en el archivo JSON.
- **updateFile(JsonObject jsonContent):** Actualiza el archivo JSON con el contenido proporcionado en el objeto jsonContent.

Gestor lista de usuarios

Descripción: La clase gestorListaUsuarios se encarga de gestionar la lista de usuarios almacenada en un archivo JSON. Permite obtener la lista actual de usuarios y agregar nuevos usuarios a la lista.

Atributos:

- **path (tipo: String)**: Ruta del archivo JSON que contiene la lista de usuarios.
- **gsonArr (tipo: JsonArray):** Arreglo JSON que almacena la información de los usuarios.
- **json (tipo: String):** Cadena que almacena el contenido del archivo JSON.

Métodos:

- **gestorListaUsuarios():** Constructor que inicializa la clase con la ruta del archivo de usuarios y carga el contenido del archivo JSON.
- **getUsuarios():** Retorna un ArrayList con la información de los usuarios actuales (nombre de usuario y ID).
- **updateUsuarios(ArrayList<String> usuario):** Agrega un nuevo usuario a la lista de usuarios.
- **updateListaUsuarios(ArrayList<ArrayList<String>> listaUsuarios):** Actualiza la lista de usuarios en un archivo JSON, convirtiendo la información de la lista de usuarios proporcionada a formato JSON y escribiéndola en el archivo correspondiente.
- **updateFile(JsonObject jsonContent):** Actualiza el archivo JSON con el contenido proporcionado en el objeto jsonContent.

Gestor de metadata

Descripción: La clase gestorMetadata se encarga de gestionar la metadata relacionada con los índices de diferentes entidades (usuario, teclado, alfabeto, texto, lista) almacenados en un archivo JSON. Permite obtener los índices actuales y actualizarlos en el archivo JSON.

Atributos:

- **gsonArr (tipo: JsonArray):** Arreglo JSON que almacena la información de los índices.
- **json (tipo: String):** Cadena que almacena el contenido del archivo JSON.
- **path (tipo: String):** Ruta del archivo JSON que contiene la metadata de índices.

Métodos:

- **gestorMetadata():** Constructor que inicializa la clase con la ruta del archivo de metadata y carga el contenido del archivo JSON.
- **getIndices():** Retorna un ArrayList con los índices actuales de usuario, teclado, alfabeto, texto y lista.
- **updateIndices(ArrayList<Integer> indices):** Actualiza los índices en el archivo JSON con los valores proporcionados.
- **updateFile(JsonObject jsonContent):** Actualiza el archivo JSON con el contenido proporcionado en el objeto jsonContent.

Gestor IO

Descripción: La clase gestorIO es responsable de gestionar las operaciones de entrada y salida (I/O) relacionadas con la importación y exportación de texto, alfabeto, listas y teclados desde y hacia archivos. Proporciona métodos para leer y escribir archivos de diferentes tipos.

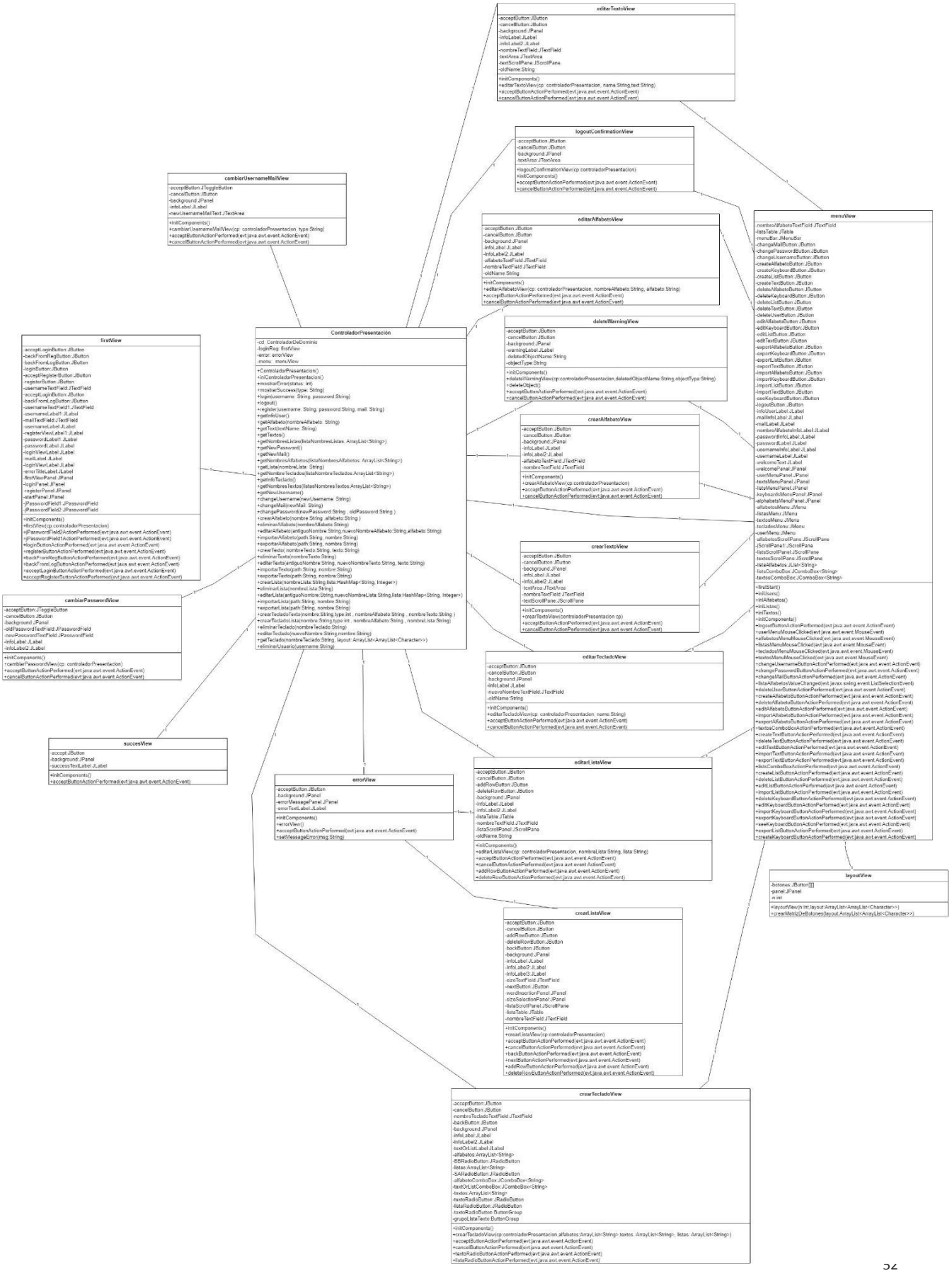
Atributos: No tiene atributos privados

Métodos:

- **gestorIO():** Constructor de la clase.
- **private String readFile(String path):** Método privado que lee el contenido de un archivo y lo devuelve como una cadena de texto.
- **private void writeFile(String path, String nombre, String texto):** Método privado que escribe el texto proporcionado en un archivo con el nombre especificado en la ruta dada. El archivo se crea si no existe.
- **public String importarTexto(String path):** Método público que importa el contenido de un archivo de texto y lo devuelve como una cadena de texto.
- **public String importarAlfabeto(String path):** Método público que importa el contenido de un archivo de alfabeto y lo devuelve como una cadena de texto.
- **public HashMap<String, Integer>importarLista(String path):** Método público que importa el contenido de un archivo de lista de palabras y lo devuelve como un HashMap, donde las claves son palabras y los valores son enteros.

- **public void exportarAlfabeto(String path, String nombre, String texto):** Método público que exporta el contenido de un alfabeto a un archivo con el nombre especificado en la ruta dada.
- **public void exportarTexto(String path, String nombre, String texto):** Método público que exporta el contenido de un texto a un archivo con el nombre especificado en la ruta dada.
- **public void exportarTeclado(String path, String nombre, char[][] layout):** Método público que exporta el contenido de un teclado a un archivo con el nombre especificado en la ruta dada.
- **public void exportarLista(String path, String nombre, HashMap<String, Integer> lista):** Método público que exporta el contenido de una lista de palabras a un archivo con el nombre especificado en la ruta dada.

5. MODELO CONCEPTUAL PRESENTACIÓN



5.1. Descripción de las clases de presentación

Clase Controlador Presentacion

Descripción: La clase **ControladorPresentacion** pertenece al paquete presentación y actúa como un controlador de la interfaz de usuario, conectando la capa de presentación con la lógica de dominio. Aquí se gestionan eventos y se comunican las acciones del usuario con las operaciones del dominio.

Atributos:

- **cd** (tipo ControladorDominio): Controlador de dominio que maneja la lógica de la aplicación.
- **loginReg** (tipo FirstView): Ventana de inicio de sesión y registro.
- **error** (tipo ErrorView): Ventana para mostrar mensajes de error.
- **menu** (tipo MenuView): Ventana principal del menú de la aplicación.

Métodos:

- **iniControladorPresentacion()**: Inicializa la interfaz de usuario mostrando la ventana de inicio de sesión y registro.
- **mostrarError(int status)**: Muestra una ventana de error con un mensaje asociado a un código de error.
- **mostrarSuccess(String type)**: Muestra una ventana de éxito con un mensaje asociado al tipo de éxito.
- **login(String username, String password)**: Realiza el inicio de sesión y muestra el menú principal si tiene éxito.
- **logout()**: Cierra la sesión actual y muestra la ventana de inicio de sesión.
- **register(String username, String password, String mail)**: Registra un nuevo usuario y muestra el menú principal si tiene éxito.
- **getInfoUser():** Retorna un ArrayList con la información del usuario actual.
- **getAlfabeto(String nombreAlfabeto):** Retorna un String con el alfabeto asociado al nombre proporcionado.
- **getNombresAlfabetos(ArrayList<String> listaNombresAlfabetos):** Rellena la lista dada con los nombres de todos los alfabetos.
- **getLista(String nombreLista):** Retorna un HashMap con la información de la lista de palabras asociada al nombre proporcionado.
- **getNombresListas(ArrayList<String> listaNombresListas):** Rellena la lista dada con los nombres de todas las listas.
- **getNombreTeclados(ArrayList<String> listaNombreTeclados):** Rellena la lista dada con los nombres de todos los teclados.
- **getInfoTeclado(String nombreTeclado):** Retorna un ArrayList con la información del teclado asociado al nombre proporcionado.
- **getNombresTextos(ArrayList<String> listasNombresTextos):** Rellena la lista dada con los nombres de todos los textos.
- **getText(String textName):** Retorna un String con el contenido del texto asociado al nombre proporcionado.
- **getTeclado(String nombreTeclado, ArrayList<ArrayList<Character>> layout):** Obtiene el layout del teclado asociado al nombre proporcionado.
- **changeUsername(String newUsername):** Cambia el nombre de usuario.
- **changeMail(String newMail):** Cambia la dirección de correo del usuario.
- **changePassword(String newPassword, String oldPassword):** Cambia la contraseña del usuario.
- **crearAlfabeto(String nombre, String alfabeto):** Crea un nuevo alfabeto.
- **eliminarAlfabeto(String nombreAlfabeto):** Elimina un alfabeto existente.
- **editarAlfabeto(String antiguoNombre, String nuevoNombreAlfabeto, String alfabeto):** Edita un alfabeto existente.
- **importarAlfabeto(String path, String nombre):** Importa un alfabeto desde un archivo.

- **exportarAlfabeto(String path, String nombre):** Exporta un alfabeto a un archivo.
- **crearTexto(String nombreTexto, String texto):** Crea un nuevo texto.
- **eliminarTexto(String nombreTexto):** Elimina un texto existente.
- **editarTexto(String antiguoNombre, String nuevoNombreTexto, String texto):** Edita un texto existente.
- **importarTexto(String path, String nombre):** Importa un texto desde un archivo.
- **exportarTexto(String path, String nombre):** Exporta un texto a un archivo.
- **crearLista(String nombreLista, HashMap<String, Integer> lista):** Crea una nueva lista de palabras.
- **eliminarLista(String nombreLista):** Elimina una lista de palabras existente.
- **editarLista(String antiguoNombre, String nuevoNombreLista, HashMap<String, Integer> lista):** Edita una lista de palabras existente.
- **importarLista(String path, String nombre)**: Importa una lista de palabras desde un archivo.
- **exportarLista(String path, String nombre)**: Exporta una lista de palabras a un archivo.
- **crearTecladoTexto(String nombre, int type, String nombreAlfabeto, String nombreTexto)**: Crea un nuevo teclado asociado a un texto.
- **crearTecladoLista(String nombre, int type, String nombreAlfabeto, String nombreLista)**: Crea un nuevo teclado asociado a una lista de palabras.
- **eliminarTeclado(String nombreTeclado)**: Elimina un teclado existente.
- **editarTeclado(String nombre, String nuevoNombre)**: Edita el nombre de un teclado existente.
- **eliminarUsuario(String username)**: Elimina el usuario activo en la aplicación.

Clase Success View

Descripción: La clase successView representa una ventana de éxito en una interfaz gráfica de usuario (GUI) para informar al usuario sobre el éxito de una operación específica, como cambios, creaciones, eliminaciones, ediciones, importaciones o exportaciones. Esta ventana proporciona un mensaje informativo sobre el éxito de la operación y un botón "Aceptar" para cerrar la ventana.

Atributos:

- **background**: Panel que sirve como fondo de la ventana, con un color de fondo.
- **successTextLabel**: Etiqueta de texto que muestra el mensaje de éxito.
- **acceptButton**: Botón "Aceptar" que cierra la ventana al hacer clic.

Métodos:

- **successView(String type):** Constructor que inicializa la ventana de éxito con un mensaje específico según el tipo de operación (cambio, creación, eliminación, edición, importación o exportación).
- **initComponents():** Inicializa los componentes de la interfaz gráfica, configurando el diseño y asignando eventos.
- **acceptButtonActionPerformed(java.awt.event.ActionEvent evt)**: Manejador de eventos que oculta la ventana al hacer clic en el botón "Aceptar".

Clase First View

Descripción: La clase firstView representa la interfaz gráfica de usuario (GUI) para la primera vista de una aplicación. Proporciona opciones para iniciar sesión o registrarse, con formularios correspondientes para cada acción. La clase interactúa con un controlador de presentación (controladorPresentacion) para gestionar las operaciones de inicio de sesión y registro**.**

Atributos:

- **cp**: Referencia al controlador de presentación que gestiona las operaciones.
- **acceptLoginButton**: Botón para aceptar la información de inicio de sesión.
- **acceptRegisterButton**: Botón para aceptar la información de registro.

- **appNameLabel**: Etiqueta que muestra el nombre de la aplicación.
- **backFromLogButton**: Botón para volver atrás desde el formulario de inicio de sesión.
- **backFromRegButton**: Botón para volver atrás desde el formulario de registro.
- **firstViewPanel**: Panel que contiene las diferentes vistas de la interfaz.
- **jPasswordField1**: Campo de contraseña en el formulario de inicio de sesión.
- **jPasswordField2**: Campo de contraseña en el formulario de registro.
- **loginButton**: Botón para acceder al formulario de inicio de sesión.
- **loginPanel**: Panel que contiene el formulario de inicio de sesión.
- **loginViewLabel**: Etiqueta que indica la vista de inicio de sesión.
- **mailLabel**: Etiqueta que indica el campo de correo en el formulario de registro.
- **mailTextField**: Campo de correo en el formulario de registro.
- **passwordLabel**: Etiqueta que indica el campo de contraseña en el formulario de inicio de sesión.
- **passwordLabel1**: Etiqueta que indica el campo de contraseña en el formulario de registro.
- **registerButton**: Botón para acceder al formulario de registro.
- **registerPanel**: Panel que contiene el formulario de registro.
- **registerViewLabel1**: Etiqueta que indica la vista de registro.
- **startPanel**: Panel principal con las opciones iniciales.
- **usernameLabel**: Etiqueta que indica el campo de usuario en el formulario de inicio de sesión.
- **usernameLabel1**: Etiqueta que indica el campo de usuario en el formulario de registro.
- **usernameTextField**: Campo de usuario en el formulario de inicio de sesión.
- **usernameTextField1**: Campo de usuario en el formulario de registro.

Métodos:

- **firstView(controladorPresentacion cp):** Constructor que recibe un controlador de presentación y lo asocia con la instancia.
- **initComponents()**: Inicializa los componentes de la interfaz gráfica, configurando el diseño y asignando eventos.
- **jPasswordField1ActionPerformed(java.awt.event.ActionEvent evt):** Método invocado al realizar una acción en el campo de contraseña del formulario de inicio de sesión.
- **loginButtonActionPerformed(java.awt.event.ActionEvent evt):** Método invocado al hacer clic en el botón de inicio de sesión, muestra el formulario correspondiente.
- **registerButtonActionPerformed(java.awt.event.ActionEvent evt):** Método invocado al hacer clic en el botón de registro, muestra el formulario correspondiente.
- **backFromLogButtonActionPerformed(java.awt.event.ActionEvent evt):** Método invocado al hacer clic en el botón de retroceso desde el formulario de inicio de sesión.
- **backFromRegButtonActionPerformed(java.awt.event.ActionEvent evt):** Método invocado al hacer clic en el botón de retroceso desde el formulario de registro.
- **jPasswordField2ActionPerformed(java.awt.event.ActionEvent evt):** Método invocado al realizar una acción en el campo de contraseña del formulario de registro.
- **acceptRegisterButtonActionPerformed(java.awt.event.ActionEvent evt):** Método invocado al hacer clic en el botón de registro en el formulario de registro, recopila la información y realiza la operación correspondiente a través del controlador de presentación.
- **acceptLoginButtonActionPerformed(java.awt.event.ActionEvent evt):** Método invocado al hacer clic en el botón de inicio de sesión en el formulario de inicio de sesión, recopila la información y realiza la operación correspondiente a través del controlador de presentación.

Clase Menu View

Descripción: La clase menuView es una interfaz gráfica de usuario construida con Java Swing. Representa el menú principal de una aplicación y proporciona funcionalidades para la gestión de alfabetos, listas, teclados, textos y usuarios.

Atributos:

- **cp** (tipo: controladorPresentacion): Representa una instancia del controlador de presentación.
- **alfabetosMenu** (tipo: javax.swing.JMenu): Menú relacionado con alfabetos.
- **alfabetosScrollPanel** (tipo: javax.swing.JScrollPane): Panel de desplazamiento para la tabla de alfabetos.
- **alfabetosTable** (tipo: javax.swing.JTable): Tabla que muestra información sobre alfabetos.
- **alphabetsMenuPanel** (tipo: javax.swing.JPanel): Panel que contiene componentes relacionados con alfabetos, como botones para crear, eliminar, editar, importar y exportar alfabetos.
- **listsMenu** (tipo: javax.swing.JMenu): Menú relacionado con listas.
- **listsComboBox** (tipo: javax.swing.JComboBox<String>): Menú desplegable para seleccionar listas.
- **listsScrollPanel** (tipo: javax.swing.JScrollPane): Panel de desplazamiento para la tabla de listas.
- **listsTable** (tipo: javax.swing.JTable): Tabla que muestra información sobre listas.
- **tecladosMenu** (tipo: javax.swing.JMenu): Menú relacionado con teclados.
- **keyboardScrollPanel** (tipo: javax.swing.JScrollPane): Panel de desplazamiento para la tabla de teclados.
- **keyboardTable** (tipo: javax.swing.JTable): Tabla que muestra información sobre teclados.
- **createKeyboardButton**, **deleteKeyboardButton**, **editKeyboardButton**, **exportKeyboardButton**, **seeKeyboardButton** (tipo: javax.swing.JButton): Botones para realizar operaciones relacionadas con teclados.
- **textosMenu** (tipo: javax.swing.JMenu): Menú relacionado con textos.
- **textosComboBox** (tipo: javax.swing.JComboBox<String>): Menú desplegable para seleccionar textos.
- **textosScrollPane** (tipo: javax.swing.JScrollPane): Panel de desplazamiento para el área de texto.
- **textArea** (tipo: javax.swing.JTextArea): Área de texto que muestra información sobre textos.
- **createTextButton**, **deleteTextButton**, **editTextButton**, **importTextButton**, **exportTextButton** (tipo: javax.swing.JButton): Botones para realizar operaciones relacionadas con textos.
- **userMenu** (tipo: javax.swing.JMenu): Menú relacionado con el usuario.
- **infoUserLabel**, **usernameInfoLabel**, **passwordInfoLabel**, mailInfoLabel (tipo: javax.swing.JLabel): Etiquetas que muestran información sobre el usuario.
- **usernameLabel**, **passwordLabel**, **mailLabel** (tipo: javax.swing.JLabel): Etiquetas que muestran el nombre de usuario, contraseña y correo electrónico del usuario.
- **logoutButton**, **changeUsernameButton**, **changePasswordButton**, **changeMailButton**, **deleteUserButton** (tipo: javax.swing.JButton): Botones para realizar operaciones relacionadas con el usuario.
- **welcomePanel** (tipo: javax.swing.JPanel): Panel de bienvenida.
- **welcomeText** (tipo: javax.swing.JLabel): Etiqueta que muestra un mensaje de bienvenida.

Esta clase contiene varios métodos muy extensos para poder visualizar el menú principal como se ve en la aplicación.

Clase Cambiar Password View

Descripción: La clase cambiarPasswordView representa una interfaz de usuario para cambiar la contraseña de un usuario. Esta interfaz contiene campos para ingresar la contraseña actual y la nueva contraseña, así como botones para aceptar o cancelar la operación.

Atributos:

- **cp** (tipo: controladorPresentacion): Representa una instancia del controlador de presentación.
- **acceptButton** (tipo: javax.swing.JToggleButton): Botón que permite aceptar la operación de cambio de contraseña.
- **background** (tipo: javax.swing.JPanel): Panel que sirve como fondo de la interfaz.
- **cancelButton** (tipo: javax.swing.JButton): Botón que permite cancelar la operación de cambio de contraseña.

- **infoLabel** (tipo: javax.swing.JLabel): Etiqueta que muestra el texto "Password:".
- **infoLabel2** (tipo: javax.swing.JLabel): Etiqueta que muestra el texto "Nueva password:".
- **newPasswordTextField** (tipo: javax.swing.JPasswordField): Campo de texto para ingresar la nueva contraseña.
- **oldPasswordTextField** (tipo: javax.swing.JPasswordField): Campo de texto para ingresar la contraseña actual.

Métodos:

- **initComponents():** Inicializa y configura los componentes de la interfaz, como etiquetas, campos de texto y botones.
- **acceptButtonActionPerformed(java.awt.event.ActionEvent evt):** Método invocado cuando se hace clic en el botón de aceptar. Obtiene las contraseñas ingresadas, llama al método changePassword del controlador de presentación (cp) para cambiar la contraseña, y oculta la interfaz.
- **cancelButtonActionPerformed(java.awt.event.ActionEvent evt):** Método invocado cuando se hace clic en el botón de cancelar. Oculta la interfaz sin realizar cambios en la contraseña.

Clase Cambiar Username Mail View

Descripción: La clase cambiarUsernameMailView representa una interfaz de usuario que permite cambiar el nombre de usuario o la dirección de correo electrónico de un usuario. Dependiendo del tipo especificado ("usr" para nombre de usuario, cualquier otro valor para correo electrónico), la interfaz mostrará un campo de texto correspondiente y etiquetas apropiadas.

Atributos:

- **cp** (tipo: controladorPresentacion): Representa una instancia del controlador de presentación.
- **type** (tipo: String): Almacena el tipo de cambio que se va a realizar ("usr" para nombre de usuario, cualquier otro valor para correo electrónico).
- **acceptButton** (tipo: javax.swing.JToggleButton): Botón que permite aceptar la operación de cambio de nombre de usuario o correo electrónico.
- **background** (tipo: javax.swing.JPanel): Panel que sirve como fondo de la interfaz.
- **cancelButton** (tipo: javax.swing.JButton): Botón que permite cancelar la operación de cambio.
- **infoLabel** (tipo: javax.swing.JLabel): Etiqueta que muestra el texto "Nuevo username:" o "Nuevo correo:" según el tipo de cambio.
- **newUsernameMailText** (tipo: javax.swing.JTextArea): Área de texto donde se ingresa el nuevo nombre de usuario o correo electrónico.

Métodos:

- **initComponents():** Inicializa y configura los componentes de la interfaz, como etiquetas, campos de texto y botones.
- **acceptButtonActionPerformed(java.awt.event.ActionEvent evt)**: Método invocado cuando se hace clic en el botón de aceptar. Obtiene el texto ingresado en el campo correspondiente y llama al método changeUsername o changeMail del controlador de presentación (cp) según el tipo de cambio especificado. Luego, oculta la interfaz.
- **cancelButtonActionPerformed(java.awt.event.ActionEvent evt):** Método invocado cuando se hace clic en el botón de cancelar. Oculta la interfaz sin realizar cambios.

Clase Crear Alfabeto View

Descripción: La clase crearAlfabetoView representa una interfaz de usuario que permite al usuario crear un nuevo alfabeto. La interfaz solicita al usuario ingresar un nombre para el alfabeto y una cadena que represente

el conjunto de símbolos que forman dicho alfabeto. La clase está diseñada para interactuar con un controlador de presentación (controladorPresentacion) que gestionará la creación del alfabeto.

Atributos:

- **cp** (tipo: controladorPresentacion): Representa una instancia del controlador de presentación.
- **acceptButton** (tipo: javax.swing.JButton): Botón que permite aceptar la creación del alfabeto.
- **alfabetoTextField** (tipo: javax.swing.JTextField): Campo de texto donde el usuario ingresa la cadena que representa el alfabeto.
- **background** (tipo: javax.swing.JPanel): Panel que sirve como fondo de la interfaz.
- **cancelButton** (tipo: javax.swing.JButton): Botón que permite cancelar la operación de creación del alfabeto.
- **infoLabel** (tipo: javax.swing.JLabel): Etiqueta que indica al usuario que debe ingresar el nombre del alfabeto.
- **infoLabel2** (tipo: javax.swing.JLabel): Etiqueta que indica al usuario que debe ingresar la cadena que representa el alfabeto.
- **nombreTextField** (tipo: javax.swing.JTextField): Campo de texto donde el usuario ingresa el nombre del alfabeto.

Métodos:

- **initComponents**(): Inicializa y configura los componentes de la interfaz, como etiquetas, campos de texto y botones.
- **acceptButtonActionPerformed(java.awt.event.ActionEvent evt):** Método invocado cuando se hace clic en el botón de aceptar. Obtiene los valores ingresados en los campos de texto (nombreTextField y alfabetoTextField) y llama al método crearAlfabeto del controlador de presentación (cp) para gestionar la creación del alfabeto. Luego, oculta la interfaz.
- **cancelButtonActionPerformed(java.awt.event.ActionEvent evt):** Método invocado cuando se hace clic en el botón de cancelar. Oculta la interfaz sin realizar cambios.

Clase Crear Lista View

Descripción: La clase crearListaView representa una interfaz de usuario que permite al usuario crear una lista de palabras con sus respectivas frecuencias. La interfaz consta de dos paneles: uno para seleccionar el tamaño de la lista y otro para insertar las palabras y frecuencias correspondientes. La clase utiliza un controlador de presentación (controladorPresentacion) para gestionar la creación de la lista.

Atributos:

- **cp** (tipo: controladorPresentacion): Representa una instancia del controlador de presentación.
- **acceptButon** (tipo: javax.swing.JButton): Botón que permite aceptar y confirmar la creación de la lista.
- **addRowButton** (tipo: javax.swing.JButton): Botón que permite agregar una nueva fila para insertar una palabra y su frecuencia en la tabla.
- **backButton** (tipo: javax.swing.JButton): Botón que permite regresar a la selección del tamaño de la lista.
- **cancelButton** (tipo: javax.swing.JButton): Botón que permite cancelar la operación de creación de la lista.
- **deleteRowButton** (tipo: javax.swing.JButton): Botón que permite eliminar la última fila de la tabla de palabras y frecuencias.
- **infoLabel**, **infoLabel2**, **infoLabel3** (tipo: javax.swing.JLabel): Etiquetas que proporcionan información o instrucciones al usuario en los distintos paneles.

- **listaScrollPanel** (tipo: javax.swing.JScrollPane): Panel de desplazamiento que contiene la tabla para mostrar las palabras y frecuencias.
- **listaTable** (tipo: javax.swing.JTable): Tabla donde el usuario puede insertar palabras y frecuencias.
- **nextButton** (tipo: javax.swing.JButton): Botón que permite avanzar al panel de inserción de palabras después de seleccionar el tamaño de la lista.
- **nombreTextField** (tipo: javax.swing.JTextField): Campo de texto donde el usuario ingresa el nombre de la lista.
- **sizeSelectionPanel**, **wordInsertionPanel** (tipo: javax.swing.JPanel): Paneles que contienen los componentes para la selección del tamaño de la lista y la inserción de palabras, respectivamente.
- **sizeTextField** (tipo: javax.swing.JTextField): Campo de texto donde el usuario ingresa el tamaño de la lista.

Métodos:

- **initComponents():** Inicializa y configura los componentes de la interfaz, como etiquetas, campos de texto, botones y paneles.
- **acceptButonActionPerformed(java.awt.event.ActionEvent evt):** Método invocado cuando se hace clic en el botón de aceptar. Obtiene los datos ingresados en la tabla y el nombre de la lista, realiza validaciones y llama al método crearLista del controlador de presentación (cp) para gestionar la creación de la lista. Muestra un mensaje de error si hay un formato incorrecto en la frecuencia.
- **cancelButtonActionPerformed(java.awt.event.ActionEvent evt):** Método invocado cuando se hace clic en el botón de cancelar. Regresa al panel de selección del tamaño de la lista y reinicia el nombre de la lista.
- **backButtonActionPerformed(java.awt.event.ActionEvent evt):** Método invocado cuando se hace clic en el botón de regresar. Cierra la ventana de creación de lista.
- **nextButtonActionPerformed(java.awt.event.ActionEvent evt):** Método invocado cuando se hace clic en el botón de siguiente. Obtiene el tamaño de la lista ingresado, crea un modelo de tabla con esa cantidad de filas y avanza al panel de inserción de palabras.
- **addRowButtonActionPerformed(java.awt.event.ActionEvent evt):** Método invocado cuando se hace clic en el botón de agregar fila. Añade una nueva fila a la tabla para insertar una palabra y su frecuencia.
- **deleteRowButtonActionPerformed(java.awt.event.ActionEvent evt):** Método invocado cuando se hace clic en el botón de eliminar fila. Elimina la última fila de la tabla de palabras y frecuencias.

Clase Crear Tecaldo View

Descripción: La clase crearTecladoView es una interfaz de usuario que permite al usuario crear un teclado con diferentes configuraciones. El teclado puede estar asociado a un alfabeto específico y puede ser creado a partir de un texto o una lista de palabras. La interfaz proporciona opciones para seleccionar el nombre del teclado, el alfabeto, el tipo de objeto (texto o lista) y el algoritmo de generación del teclado (BB o SA). La clase utiliza un controlador de presentación (controladorPresentacion) para gestionar la creación del teclado.

Atributos:

- **cp** (tipo: controladorPresentacion): Representa una instancia del controlador de presentación.
- **alfabetos** (tipo: ArrayList<String>): Lista de nombres de alfabetos disponibles.
- **textos** (tipo: ArrayList<String>): Lista de nombres de textos disponibles.
- **listas** (tipo: ArrayList<String>): Lista de nombres de listas disponibles.
- **BBRadioButton** y **SARadioButton** (tipo: javax.swing.JRadioButton): Botones de opción para seleccionar el algoritmo de generación del teclado (BB o SA).
- **acceptButton** y **cancelButton** (tipo: javax.swing.JButton): Botones que permiten aceptar y cancelar la creación del teclado, respectivamente.
- **alfabetoComboBox**, **textOrListComboBox** (tipo: javax.swing.JComboBox<String>): Cuadros combinados para seleccionar el alfabeto y el objeto (texto o lista) asociado al teclado.

- **background** (tipo: javax.swing.JPanel): Panel de fondo que contiene los componentes de la interfaz.
- **grupoListaTexto** y **grupoAlgoritmos** (tipo: javax.swing.ButtonGroup): Grupos de botones de opción para garantizar que solo se seleccione un tipo de objeto (texto o lista) y un algoritmo de generación.
- **infoLabel**, **infoLabel2**, **textOrListLabel** (tipo: javax.swing.JLabel): Etiquetas que proporcionan información o instrucciones al usuario.
- **nombreTecladoTextField** (tipo: javax.swing.JTextField): Campo de texto donde el usuario ingresa el nombre del teclado.
- **textoRadioButton** y **listaRadioButton** (tipo: javax.swing.JRadioButton): Botones de opción para seleccionar el tipo de objeto a utilizar en la creación del teclado (texto o lista).

Métodos:

- **initComponents**(): Inicializa y configura los componentes de la interfaz, como etiquetas, campos de texto, botones y paneles.
- **textoRadioButtonActionPerformed**(java.awt.event.ActionEvent evt): Método invocado cuando se selecciona el botón de opción para usar un texto en la creación del teclado. Actualiza las opciones del cuadro combinado textOrListComboBox con los textos disponibles.
- **listaRadioButtonActionPerformed**(java.awt.event.ActionEvent evt): Método invocado cuando se selecciona el botón de opción para usar una lista en la creación del teclado. Actualiza las opciones del cuadro combinado textOrListComboBox con las listas disponibles.
- **acceptButtonActionPerformed**(java.awt.event.ActionEvent evt): Método invocado cuando se hace clic en el botón de aceptar. Obtiene los parámetros necesarios (nombre del teclado, tipo de objeto, algoritmo, nombre del alfabeto, nombre del texto o lista) y llama al controlador de presentación (cp) para gestionar la creación del teclado.
- **cancelButtonActionPerformed**(java.awt.event.ActionEvent evt): Método invocado cuando se hace clic en el botón de cancelar. Cierra la ventana de creación de teclado.

Clase Crear Texto View

Descripción: La clase crearTextoView es una interfaz de usuario que permite al usuario crear un nuevo objeto de texto. La interfaz proporciona campos para ingresar el nombre del texto y su contenido. Además, hay botones para aceptar la creación del texto o cancelar la operación. La clase utiliza un controlador de presentación (controladorPresentacion) para gestionar la creación del texto.

Atributos:

- **cp** (tipo: controladorPresentacion): Representa una instancia del controlador de presentación.
- **acceptButton** y **cancelButton** (tipo: javax.swing.JButton): Botones que permiten aceptar o cancelar la creación del texto, respectivamente.
- **background** (tipo: javax.swing.JPanel): Panel de fondo que contiene los componentes de la interfaz.
- **infoLabel** y **infoLabel2** (tipo: javax.swing.JLabel): Etiquetas que proporcionan información sobre el nombre del texto y su contenido.
- **nombreTextField** (tipo: javax.swing.JTextField): Campo de texto donde el usuario ingresa el nombre del texto.
- **textArea** (tipo: javax.swing.JTextArea): Área de texto donde el usuario ingresa el contenido del texto.
- **textScrollPane** (tipo: javax.swing.JScrollPane): Barra de desplazamiento que envuelve el área de texto, permitiendo visualizar todo el contenido si es extenso.

Métodos:

- **initComponents**(): Inicializa y configura los componentes de la interfaz, como etiquetas, campos de texto, botones y paneles.

- **cancelButtonActionPerformed**(java.awt.event.ActionEvent evt): Método invocado cuando se hace clic en el botón de cancelar. Cierra la ventana de creación de texto.
- **acceptButtonActionPerformed**(java.awt.event.ActionEvent evt): Método invocado cuando se hace clic en el botón de aceptar. Obtiene el nombre y el contenido del texto desde los campos correspondientes y llama al controlador de presentación (cp) para gestionar la creación del texto.

Clase Delete Warning View

Descripción: La clase deleteWarningView es una interfaz de usuario que muestra una advertencia de eliminación para un objeto específico, como un alfabeto, texto, lista, teclado o usuario. La interfaz proporciona botones para confirmar o cancelar la eliminación del objeto. La clase utiliza un controlador de presentación (controladorPresentacion) para llevar a cabo las operaciones de eliminación correspondientes.

Atributos:

- **cp** (tipo: controladorPresentacion): Representa una instancia del controlador de presentación.
- **deletedObjectName** (tipo: String): Almacena el nombre del objeto que se va a eliminar.
- **objectType** (tipo: String): Indica el tipo de objeto que se va a eliminar (Alfabeto, Texto, Lista, Teclado, Usuario).
- **acceptButton** y **cancelButton** (tipo: javax.swing.JButton): Botones que permiten confirmar o cancelar la eliminación del objeto, respectivamente.
- **background** (tipo: javax.swing.JPanel): Panel de fondo que contiene los componentes de la interfaz.
- **warningLabel** (tipo: javax.swing.JLabel): Etiqueta que muestra el mensaje de advertencia sobre la eliminación del objeto.

Métodos:

- **initComponents**(): Inicializa y configura los componentes de la interfaz, como etiquetas, botones y paneles.
- **deleteObject**(): Método que, según el tipo de objeto especificado, llama al controlador de presentación para llevar a cabo la eliminación del objeto correspondiente (Alfabeto, Texto, Lista, Teclado, Usuario).
- **cancelButtonActionPerformed**(java.awt.event.ActionEvent evt): Método invocado cuando se hace clic en el botón de cancelar. Cierra la ventana de advertencia de eliminación.
- **acceptButtonActionPerformed**(java.awt.event.ActionEvent evt): Método invocado cuando se hace clic en el botón de aceptar. Llama al método deleteObject() para eliminar el objeto y luego cierra la ventana de advertencia.

Clase Editar Alfabeto View

Descripción: La clase editarAlfabetoView representa una interfaz de usuario que permite editar un alfabeto existente. La interfaz contiene campos de texto para modificar el nombre y el contenido del alfabeto, así como botones para confirmar o cancelar la edición. La clase utiliza un controlador de presentación (controladorPresentacion) para gestionar las operaciones de edición del alfabeto.

Atributos:

- **cp** (tipo: controladorPresentacion): Representa una instancia del controlador de presentación.
- **oldName** (tipo: String): Almacena el nombre original del alfabeto antes de la edición.
- **acceptButton** y **cancelButton** (tipo: javax.swing.JButton): Botones que permiten confirmar o cancelar la edición del alfabeto, respectivamente.
- **background** (tipo: javax.swing.JPanel): Panel de fondo que contiene los componentes de la interfaz.
- **infoLabel** y **infoLabel2** (tipo: javax.swing.JLabel): Etiquetas que describen los campos de texto para el nombre y el alfabeto, respectivamente.

- **nombreTextField** y **alfabetoTextField** (tipo: javax.swing.JTextField): Campos de texto que permiten la edición del nombre y del contenido del alfabeto.

Métodos:

- **initComponents**(): Inicializa y configura los componentes de la interfaz, como etiquetas, campos de texto y botones.
- **cancelButtonActionPerformed**(java.awt.event.ActionEvent evt): Método invocado cuando se hace clic en el botón de cancelar. Cierra la ventana de edición del alfabeto sin realizar cambios.
- **acceptButtonActionPerformed**(java.awt.event.ActionEvent evt): Método invocado cuando se hace clic en el botón de aceptar. Llama al controlador de presentación (cp) para editar el alfabeto con los nuevos valores proporcionados en los campos de texto. Después de la edición, cierra la ventana de edición del alfabeto.

Clase Editar Lista View

Descripción: La clase editarListaView representa una interfaz de usuario que permite editar una lista existente. La interfaz incluye campos para modificar el nombre de la lista y su contenido, que consiste en palabras y sus frecuencias. Utiliza un controlador de presentación (controladorPresentacion) para gestionar las operaciones de edición de la lista.

Atributos:

- **cp** (tipo: controladorPresentacion): Representa una instancia del controlador de presentación.
- **oldName** (tipo: String): Almacena el nombre original de la lista antes de la edición.
- **acceptButon** y **cancelButton** (tipo: javax.swing.JButton): Botones que permiten confirmar o cancelar la edición de la lista, respectivamente.
- **addRowButton** y **deleteRowButton** (tipo: javax.swing.JButton): Botones que permiten agregar o eliminar filas en la tabla de la lista, respectivamente.
- **background** (tipo: javax.swing.JPanel): Panel de fondo que contiene los componentes de la interfaz.
- **infoLabel** y **infoLabel2** (tipo: javax.swing.JLabel): Etiquetas que describen los campos de texto para el nombre y la lista, respectivamente.
- **nombreTextField** (tipo: javax.swing.JTextField): Campo de texto que permite la edición del nombre de la lista.
- **listaScrollPanel** (tipo: javax.swing.JScrollPane): Panel de desplazamiento que contiene la tabla de la lista.
- **listaTable** (tipo: javax.swing.JTable): Tabla que muestra las palabras y sus frecuencias en la lista.

Métodos:

- **initComponents**(): Inicializa y configura los componentes de la interfaz, como etiquetas, campos de texto, botones y la tabla.
- **acceptButonActionPerformed**(java.awt.event.ActionEvent evt): Método invocado cuando se hace clic en el botón de aceptar. Guarda los cambios realizados en la lista, como la edición del nombre y la modificación de las palabras y sus frecuencias.
- **cancelButtonActionPerformed**(java.awt.event.ActionEvent evt): Método invocado cuando se hace clic en el botón de cancelar. Cierra la ventana de edición de la lista sin realizar cambios.
- **addRowButtonActionPerformed**(java.awt.event.ActionEvent evt): Método invocado cuando se hace clic en el botón de agregar fila. Añade una nueva fila a la tabla de la lista.
- **deleteRowButtonActionPerformed**(java.awt.event.ActionEvent evt): Método invocado cuando se hace clic en el botón de eliminar fila. Elimina la última fila de la tabla de la lista.

Clase Editar Teclado View

Descripción: La clase editarTecladoView representa una interfaz de usuario que permite editar el nombre de un teclado existente. La interfaz incluye un campo para ingresar el nuevo nombre del teclado y botones para confirmar o cancelar la operación de edición. Utiliza un controlador de presentación (controladorPresentacion) para gestionar las operaciones de edición del teclado.

Atributos:

- **cp** (tipo: controladorPresentacion): Representa una instancia del controlador de presentación.
- **oldName** (tipo: String): Almacena el nombre original del teclado antes de la edición.
- **acceptButton** y **cancelButton** (tipo: javax.swing.JButton): Botones que permiten confirmar o cancelar la edición del nombre del teclado, respectivamente.
- **background** (tipo: javax.swing.JPanel): Panel de fondo que contiene los componentes de la interfaz.
- **infoLabel** (tipo: javax.swing.JLabel): Etiqueta que describe el campo de texto para el nuevo nombre del teclado.
- **nuevoNombreTextField** (tipo: javax.swing.JTextField): Campo de texto que permite la edición del nuevo nombre del teclado.

Métodos:

- **initComponents**(): Inicializa y configura los componentes de la interfaz, como etiquetas, campos de texto y botones.
- **acceptButtonActionPerformed**(java.awt.event.ActionEvent evt): Método invocado cuando se hace clic en el botón de aceptar. Guarda el cambio realizado en el nombre del teclado utilizando el controlador de presentación.
- **cancelButtonActionPerformed**(java.awt.event.ActionEvent evt): Método invocado cuando se hace clic en el botón de cancelar. Cierra la ventana de edición del teclado sin realizar cambios.

Clase Editar Texto View

Descripción: La clase editarTextoView representa una interfaz de usuario que permite editar un documento de texto existente. La interfaz incluye campos para ingresar el nuevo nombre del texto y el contenido del texto, así como botones para confirmar o cancelar la operación de edición. Utiliza un controlador de presentación (controladorPresentacion) para gestionar las operaciones de edición del texto.

Atributos:

- **cp** (tipo: controladorPresentacion): Representa una instancia del controlador de presentación.
- **oldName** (tipo: String): Almacena el nombre original del texto antes de la edición.
- **acceptButton** y cancelButton (tipo: javax.swing.JButton): Botones que permiten confirmar o cancelar la edición del texto, respectivamente.
- **background** (tipo: javax.swing.JPanel): Panel de fondo que contiene los componentes de la interfaz.
- **infoLabel** y **infoLabel2** (tipo: javax.swing.JLabel): Etiquetas que describen los campos de texto para el nuevo nombre del texto y el contenido del texto, respectivamente.
- **nombreTextField** (tipo: javax.swing.JTextField): Campo de texto que permite la edición del nuevo nombre del texto.
- **textArea** (tipo: javax.swing.JTextArea): Área de texto que permite la edición del contenido del texto.
- **textScrollPane** (tipo: javax.swing.JScrollPane): Panel de desplazamiento que contiene el área de texto, permitiendo la visualización de texto extenso.

Métodos:

- **initComponents**(): Inicializa y configura los componentes de la interfaz, como etiquetas, campos de texto, área de texto y botones.
- **acceptButtonActionPerformed**(java.awt.event.ActionEvent evt): Método invocado cuando se hace clic en el botón de aceptar. Guarda los cambios realizados en el nombre y contenido del texto utilizando el controlador de presentación.
- **cancelButtonActionPerformed**(java.awt.event.ActionEvent evt): Método invocado cuando se hace clic en el botón de cancelar. Cierra la ventana de edición del texto sin realizar cambios.

Clase Layout View

Descripción: La clase layoutView representa una interfaz de usuario que muestra un diseño (layout) en forma de matriz de botones. Esta interfaz es utilizada para visualizar y editar un diseño específico que se pasa como parámetro al constructor de la clase. La interfaz presenta una matriz de botones, donde cada botón contiene un carácter del diseño.

Atributos:

- **panel** (tipo: JPanel): Panel que contiene la matriz de botones y define el layout de la interfaz.
- **botones** (tipo: JButton[][]): Matriz de botones que representa visualmente el diseño. Cada botón contiene un carácter del diseño.
- **n** (tipo: int): Tamaño de la matriz (n x n) y también indica el número de botones en cada fila y columna.

Métodos:

- **layoutView**(int n, ArrayList<ArrayList<Character>> layout): Constructor de la clase que recibe el tamaño n de la matriz y un diseño representado por una lista de listas de caracteres (ArrayList<ArrayList<Character>>). Configura la ventana principal y llama al método crearMatrizDeBotones para inicializar y mostrar la matriz de botones.
- **crearMatrizDeBotones**(ArrayList<ArrayList<Character>> layout): Método privado que crea la matriz de botones utilizando la información del diseño pasado como parámetro. Cada botón se inicializa con el carácter correspondiente y se configuran sus propiedades visuales, como el color de fondo y la fuente.

Clase Error View

Descripción: La clase errorView representa una ventana de interfaz de usuario diseñada para mostrar mensajes de error. Puede ser utilizada para notificar al usuario sobre eventos o condiciones de error dentro de la aplicación. La ventana incluye un mensaje descriptivo del error y un botón "Aceptar" para cerrar la ventana.

Atributos:

- **background** (tipo: javax.swing.JPanel): Panel que actúa como fondo de la ventana de error, con un color específico.
- **errorTitleLabel** (tipo: javax.swing.JLabel): Etiqueta que muestra el título "ERROR" en un tamaño grande y con un color rojo.
- **errorMessagePanel** (tipo: javax.swing.JPanel): Panel que contiene la etiqueta errorTextLabel para mostrar el mensaje de error.
- **errorTextLabel** (tipo: javax.swing.JLabel): Etiqueta que muestra el mensaje de error específico.
- **acceptButton** (tipo: javax.swing.JButton): Botón "Aceptar" que permite al usuario cerrar la ventana de error.

Métodos:

- **errorView**(): Constructor de la clase que inicializa y configura la interfaz de la ventana de error mediante el método initComponents().
- **initComponents**(): Método privado que inicializa y configura visualmente los componentes de la interfaz de la ventana de error, como paneles, etiquetas y botones.
- **acceptButtonActionPerformed**(java.awt.event.ActionEvent evt): Método que se ejecuta cuando se hace clic en el botón "Aceptar". Oculta la ventana de error.
- **setMessageError**(String msg): Método público que establece el mensaje de error que se mostrará en la etiqueta errorTextLabel. Permite cambiar dinámicamente el contenido del mensaje de error.

6. RELACIÓN DE LAS CLASES IMPLEMENTADAS

Clase Usuario: Alejandra Traveria Marti

Clase Controlador de Dominio: Enric Teixidó Álvarez, Nicolás Gonzalo Longueira, Alejandra Traveria Marti

Clase Utilidad: Enric Teixidó Álvarez

Clase Algoritmo: Nicolás Gonzalo Longueira

Clase QAP: Nicolás Gonzalo Longueira

Clase Hungarian: Nicolás Gonzalo Longueira

Clase ALG2: Nicolás Gonzalo Longueira

Clase Error: Alejandra Traveria Marti , Nicolás Gonzalo Longueira

Clase Texto: Santiago Cervera Pérez, Enric Teixidó Álvarez

Clase Conjunto de Textos: Santiago Cervera Pérez, Enric Teixidó Álvarez

Clase Teclado: Nicolás Gonzalo Longueira, Enric Teixidó Álvarez

Clase Conjunto de Teclados: Nicolás Gonzalo Longueira, Enric Teixidó Álvarez

Clase Alfabeto: Nicolás Gonzalo Longueira, Enric Teixidó Álvarez

Clase Conjunto de Alfabetos: Nicolás Gonzalo Longueira, Enric Teixidó Álvarez

Clase Lista de Palabras: Enric Teixidó Álvarez

Clase Conjunto de Lista de Palabras: Enric Teixidó Álvarez

Clase Driver Controlador de Dominio: Nicolás Gonzalo Longueira

Clase TestQAP: Nicolás Gonzalo Longueira

Clase TestAlfabeto: Santiago Cervera Pérez

Clase TestListaDePalabras: Santiago Cervera Pérez, Alejandra Traveria Marti

Clase TestTeclado: Santiago Cervera Pérez

Clase TestTexto: Santiago Cervera Pérez

Clase TestUsuario: Santiago Cervera Pérez

Clase TestUtilidad: Enric Teixidó Álvarez

Persistencia: Nicolás Gonzalo Longueira

Presentación: Enric Teixidò Álvarez, Nicolás Gonzalo Longueira

Diagrama casos de uso: Alejandra Traveria Marti

Documentación casos de uso: Alejandra Traveria Marti, Enric Teixidó Álvarez, Nicolás Gonzalo Longueira, Santiago Cervera Pérez

Diagrama UML Dominio: Alejandra Traveria Marti, Enric Teixidó Álvarez, Nicolás Gonzalo Longueira, Santiago Cervera Pérez

Diagrama UML Persistencia: Nicolás Gonzalo Longueira, Alejandra Traveria Marti

Diagrama UML Presentación: Santiago Cervera Pérez

Documentación de las clases: Alejandra Traveria Marti

Documentación elección de estructuras de datos: Alejandra Traveria Marti

Documentación implementación algoritmo QAP: Nicolás Gonzalo Longueira

Documentación juegos de prueba: Enric Teixidó Álvarez, Alejandra Traveria Marti, Santiago Cervera Pérez

Documentación persistencia: Alejandra Traveria Marti, Nicolás Gonzalo Longueira

Documentación presentación: Alejandra Traveria Marti

Manual de Usuario: Alejandra Traveria Marti, Nicolás Gonzalo Longueira

Juegos de prueba Presentacion: Alejandra Traveria Marti, Nicolás Gonzalo Longueira