ÍNDICE

INTRODUCCION **2**
PLANTEAMIENTO INICIAL Y ALGORITMO EMPLEADO **2**
DEFINICION DEL FUNCIONAMIENTO BASICO **6**
DEFINICION UML DEL PROGRAMA **8**
UNLOGGED MENU **8**
ACTORES 9
CASOS DE USO 9
LOGGED MENU (ROL USER) **9**
ACTORES 10
CASOS DE USO 10
LOGGED MENU (ROL ADMIN) **12**
ACTORES 13
CASOS DE USO 13
MODIFY MENU (ROL USER) **15**
ACTORES 15
CASOS DE USO 16
ADMINISTRAR CUENTAS **17**
ACTORES 18
CASOS DE USO 18
DEFINICION INDIVIDUAL DE CLASE DEL PROGRAMA **22**
DIAGRAMA COMPLETO **22**
MAIN **23**
ADMIN **24**
CONFIG **25**
COORDINATE **27**
DATABASE **27**
INTERFACE **28**
LABYRINTH **29**
MODUSER **31**
SESSION **32**
USER **33**
UTILS **34**
DESARROLLOS DE LA IMPLEMENTACION DE CADA FUNCIONALIDAD DEL PROGRAMA **36**

MAIN **36**
ADMIN **36**
CONFIG **39**
COORDINATE **40**
DATABASE **40**
INTERFACE **41**
LABYRINTH **41**
MODUSER **44**
SESSION **46**
USER **47**
UTILS **47**
DESARROLLO DE LA IMPLEMENTACION DEL ALGORITMO **49**
Primer camino **49**
Camino más corto **50**

INTRODUCCION

Se ha planteado y desarrollado un proyecto de gestión de usuarios y roles permitiendo crear, borrar, hacer log in y modificar tanto el usuario logeado si se trata de un usuario con rol de usuario, como otros usuarios si el rol de usuario es de admistrador.

Resolución de un laberinto mediante la implementación del algoritmo de Tremaux.

Conexión con base de datos para el sistema de usuarios y uso del sistema de ficheros para tener un control por log de sistema y carga de laberintos.

El objetivo principal es aprender sobre programación orientada a objetos, viendo funcionalidades concretas que se permiten con el lenguaje de programación escogido tales como gestión de ficheros, gestión de clases o manipulación de bases de datos.

PLANTEAMIENTO INICIAL Y ALGORITMO EMPLEADO

El planteamiento de la realización del proyecto es comprender, aprender y usar herramientas y conocimientos dentro del lenguaje de programación “Java” aprendiendo y aplicando la lógica de la programación orientada a objetos, gestión de ficheros y manipulación de base de datos con una implementación del código siguiendo un estilo limpio y ordenado.

El algoritmo empleado para la resolución del laberinto es el algoritmo de “Tremaux”. Se trata de un algoritmo conceptual y matemático, que permite resolver un laberinto siempre y cuando tenga una solución posible.

Se basa en las siguientes bases:

1. No siga el mismo camino dos veces.
2. Si llega a un cruce nuevo, no importa qué camino siga.
3. Si un camino nuevo lo lleva a un cruce viejo, o a un callejón sin salida, retroceda hasta la entrada del camino.
4. Si un camino viejo lo lleva a un cruce viejo, tome un camino nuevo, y si no lo hay, tome cualquiera.

Para este proyecto se requiere una constante comunicación entre el usuario y el programa, generando la necesidad de una constante recogida de datos introducidos por el usuario para que el programa sepa qué procesos realizar para cada situación. Por lo que se ha implementado una clase específica para la obtención de estos datos: Interface.java

Esta clase es necesaria en procesos de recogida de datos para opciones dentro de menús, datos de usuarios, establecer casillas de entrada y salida, continuar…

En cuestión a los datos que gestiona el programa y deben guardarse existen dos formas de asegurar la persistencia de estos, dependiendo del tipo que sean. Observando dos fuentes:

1. Log del sistema: Simplemente se asegura la persistencia de esta información mediante el sistema de archivos, quedando guardado cada evento que se produce y que recoge el sistema de log en un fichero almacenado en una ruta especificada, manteniendo la integridad y seguridad de dicho fichero por medio del administrador que gestiona el Sistema Operativo.
2. Datos de usuarios: Para estos datos el sistema que asegura la persistencia es una base de datos SQL (MySQL en este caso), la cual nos permite agregar reglas sobre los datos que eviten la redundancia y la inconsistencia en los datos almacenados. Así como la posibilidad de modificar, insertar o eliminar los datos de una forma ordenada.

Para la realización de este proyecto, en mi caso, he utilizado dos softwares de entorno de desarrollo.

1. Hasta la versión 1.0.0 del proyecto he usado Visual Studio Code, ya que tiene algunas utilidades que personalmente me gustan y quería experimentar con este software, que no es tan intuitivo como Eclipse y hasta esta versión lo máximo de implementación de funcionalidad que se había visto era la gestión de archivos.
2. A partir de la versión 1.1.0 del proyecto he usado Eclipse, ya que permite una configuración más sencilla para implementar la funcionalidad de conexión con base de datos y quería desarrollar con este software, ya que en realidad, en todo el curso ha sido el que menos he usado.

Centrándome en el apartado de base de datos hay varias opciones disponibles: xampp, apache + mysql, mysql server, docker usando un contenedor…

Sinceramente, al ser un proyecto orientado al aprendizaje personal, no veo necesario más que el hecho de permitir la conexión al servidor desde el sistema local, por lo que, descartando servicios web para cualquier conexión ya sea con el programa o la base de datos. Por lo que he escogido MySQL server instalado directamente en mi máquina, que es sobre la que he desarrollado el software. A demás es un SGBD con el que ya estoy familiarizado, teniendo la facilidad de que la conexión se hace directamente desde la máquina local.

El programa ha pasado por algunas versiones mientras se ha ido desarrollando:

1. 0.1.0: Esta fue la versión principal, sobre la que se implementaron los cimientos de la app. El objetivo principal de esta versión fue crear la estructura del programa, así como la gestión de usuarios usando el sistema de ficheros como medio para la persistencia de los datos de dichos usuarios. Las principales clases y funcionalidades fueron:

- ✓ Main.java: Clase que contiene el programa principal y desde la que se implementan todas las clases y métodos declarados.
- ✓ Config.java: Clase que contiene tanto información de mensajes, como rutas, información útil, requerimientos y menús.
- ✓ Interface.java: Clase que contiene el escáner por teclado encargada de pedir los diferentes datos al usuario que maneja el programa y devolver esos valores. Sus métodos pueden contener texto o no contenerlo.
- ✓ User.java: Clase que contiene las propiedades de usuario.
- ✓ Session.java Clase que contiene las propiedades y métodos necesarios para gestionar el sistema de usuarios y sesión, con métodos para leer el fichero de usuarios, escribir en el fichero de usuarios, registrar nuevos usuarios, iniciar sesión, establecer estados de log in, comprobar daros, guardar y consultar las propiedades de los usuarios de la clase User.

2. 0.2.0: Versión que implementa sobre la versión 0.1.0. la funcionalidad de cargar un laberinto, ver el laberinto cargado y establecer casillas de entrada y salida con las modificaciones pertinentes sobre la versión 0.1.0. y la implementación de la clase Labyrinth.java que contiene todas las propiedades y métodos necesarios para realizar dichas funcionalidades.
3. 1.0.0: Versión que incorpora el algoritmo de resolución del laberinto, ayudándose de la creación de la clase Coordinate.java para el control de las coordenadas dentro del mapa cargado. Cuando el laberinto es resuelto se muestra tanto el camino seguido en el laberinto como una lista de los pasos que se han dado. Para la resolución del laberinto se implementan dos algoritmos basados en el algoritmo de Tremaux en los que tenemos dos opciones:
   - ✓ El primer camino posible: El sistema ejecuta el algoritmo hasta que encuentra el camino por primera vez.
   - ✓ El camino más corto: El sistema ejecuta el algoritmo hasta que recorre todas las casillas disponibles en el laberinto, guardando el camino cada vez que encuentra la salida si resulta que contiene menos pasos que la anterior vez que fue encontrada la salida, descargando todos los caminos posibles.
4. 1.1.0: Versión que implementa le gestión de usuarios mediante base de datos. Estas son las modificaciones más notables:
   - ✓ Añadir información Config.java sobre la conexión con la base de datos.
   - ✓ Eliminación/modificación de los métodos ubicados en la clase Sesión de registro, login y writeUser.
   - ✓ Creación de la clase Database.java que contiene métodos para leer en la base de datos, escribir en la base de datos y comparar valores sobre datos existentes en la base de datos.

- ✓ Se añade el campo id a las propiedades de la clase User.
- ✓ Creación de la clase Utils.java que contiene los métodos necesarios para verificar requisitos establecidos para los datos que se requieren en el registro, comprobar valores duplicados en la base de datos en el proceso de registro, encriptar contraseñas, cambiar formatos de fechas o calcular la edad de un usuario determinado.

5. 1.2.0: Versión que implementa la funcionalidad de log del sistema:
   - ✓ Se crea la clase SysLog.java que contiene los métodos para escribir sobre un fichero el resultado de los demás métodos de la clase, estos métodos recogen información específica de algunos de los eventos que se producen en el sistema.
   - ✓ Se agrega la llama al método correspondiente dentro de los métodos pertinentes para que la función del registro del evento se realice.
6. 1.3.0: Versión que implementa la funcionalidad de poder modificar los datos de un usuario autenticado:
   - ✓ Se modifica el menú principal de un usuario autenticado, se crea un nuevo menú para la modificación de datos en la clase Config a la cual se llama desde Main.
   - ✓ Se crean nuevos métodos en la clase Database para la modificación de los datos.
   - ✓ Se crea la clase ModUser.java que contiene los métodos necesarios para modificar la contraseña, nombre, NIF, email, dirección o fecha de nacimiento haciendo las comprobaciones pertinentes usando los métodos creados anteriormente en Utils y llamando a métodos específicos dentro de la clase Database para la modificación de los datos.
7. 1.4.0: Versión que implementa el uso del rol de administrador dentro del sistema, permitiendo modificar, crear, o eliminar usuarios registrados en la base de datos.
   - ✓ Se implementa un condicional dentro de Main por el que al autenticarse se comprueba si el usuario tiene rol de admi, si es correcto el menú principal es diferente, ya que la anterior opción de modificar usuario se llama “Administrar cuentas”
   - ✓ Como es lógico se define un nuevo menú especial para el rol de admin y otro menú con las opciones que se encuentran dentro de “Administrar cuentas” dentro de la clase Config.
   - ✓ Se crea la clase Admin.java que contiene la propiedad booleana admin junto con un constructor, un getter (isAdmin) para llevar el control del rol.
   - ✓ La clase Admin implementa los métodos necesarios para seleccionar un usuario, cambiar contraseña, cambiar nombre, cambiar NIF, cambiar email, cambiar dirección, cambiar año de nacimiento y borrar usuario, valiéndose de los métodos pertinentes de otras clases.

Los laberintos se encuentran ubicados dentro del sistema de ficheros del sistema operativo con extensión .txt en texto plano, tanto las paredes como la forma interna del laberinto están delimitados por el carácter '#', permitiendo de esta manera la gestión y diferenciación de lo que es un camino posible al que no lo es. A la hora de cargar los laberintos en memoria se usa un vector de tipo char y se muestra por pantalla añadiendo información extra y espacios para que sea más legible para el ojo humano.

Una vez finalizado el proyecto se ha comprendido la programación orientada a objetos así como la implementación de un código razonable y el uso de elementos externos, habiendo experimentado, investigado e implementado un código que supone los conocimientos adquiridos durante este año en la asignatura de Programación.

DEFINICION DEL FUNCIONAMIENTO BASICO

El funcionamiento básico del programa consiste en un sistema de menús con el que un usuario puede interactuar y con el que se puede conseguir tales funciones como iniciar sesión, registrarse en el sistema, cargar un laberinto, establecer casillas de entrada y salida, ver el laberinto una vez cargado, resolución del laberinto, ver usuario actual, modificar usuario, gestión de usuarios como administrador, cerrar sesión o salir del programa.

El sistema a su vez interactuará tanto con el usuario que lo maneja como con el Sistema Operativo para la gestión de archivos como con un servicio de MySQL donde se almacenará y se recuperará la información sobre los usuarios del sistema.

El sistema utilizará varias clases para la gestión individual de cada parte del programa, así como el uso de características propias de cada apartado.

Los datos de los usuarios como ya se ha mencionado están sujetas a modificaciones, ya sea por parte de un usuario con rol de usuario o por un administrador. Las principales modificaciones que se pueden hacer dentro del sistema de usuarios son las siguientes:

1. Creación de un usuario nuevo: Esta función puede hacerlo tanto un usuario que no se encuentra en el sistema mediante el registro o puede hacerlo el propio administrador. Los campos comprueban requisitos para ser introducidos, así como el control de redundancia que se especifica en la propia base de datos sobre los campos pertinentes, así como la encriptación de las contraseñas.
2. Borrar un usuario existente: Esta acción puede ejecutarla tanto un usuario autenticado como el administrador, el sistema preguntará si está seguro de borrar el usuario y pedirá la contraseña para confirmar la eliminación del usuario.
3. Modificar contraseña: Esta acción puede ejecutarla tanto un usuario autenticado como el administrador. El sistema requerirá de la introducción de la nueva contraseña, así como la confirmación de la nueva contraseña, encriptándola antes de introducirla en la base de datos.

4. Modificar nombre: Esta acción puede ejecutarla tanto un usuario autenticado como el administrador. El sistema requerirá de la introducción del nuevo nombre, verificando que es válido con el método pertinente de la clase Utils.
5. Modificar NIF: Esta acción puede ejecutarla tanto un usuario autenticado como el administrador. El sistema requerirá de la introducción del nuevo NIF, verificando tanto que es válido como que no exista ya en la base de datos dicho NIF, usando los métodos pertinentes de la clase Utils.
6. Modificar email: Esta acción puede ejecutarla tanto un usuario autenticado como el administrador. El sistema requerirá la introducción del nuevo email, verificando tanto que es válido como que no exista ya en la base de datos dicho email, usando los métodos pertinentes de la clase Utils.
7. Modificar dirección: Esta acción puede ejecutarla tanto un usuario autenticado como el administrador. El sistema requerirá de la introducción de la nueva dirección y la escribirá directamente en la base de datos, ya que no hace falta verificar la dirección.
8. Modificar la fecha de nacimiento: Esta acción puede ejecutarla tanto un usuario autenticado como el administrador. El sistema requerirá de la introducción de la nueva fecha de nacimiento en formato dd/MM/aaaa. Antes de ser escrita en la base de datos el formato de la nueva fecha debe modificarse para que se encuentre en formato aaaa-MM-dd para que sea legible para la base de datos. Para esta acción se usará el método pertinente dentro de la clase Utils.

El control de errores es muy importante para el programa, ya que cualquier error de programación por mi parte o por la introducción de datos por parte del usuario podría provocar que el programa terminase, así que se ha hecho un tratamiento de errores por el que se sigue una serie de instrucciones en caso de que se produzca un error.

Los principales errores que se puede dar por parte del usuario son los siguientes:

1. Introducir valores no numéricos dentro de los métodos “getInt()” ubicados en la clase Interface.
2. Dentro del registro que el usuario introduzca valores requeridos dentro del registro que no se correspondan con los requisitos de cada campo. Para este error simplemente se ha llevado un control de condicionales, aunque dentro del método “signUp()” de la clase Database si se ha llevado un control de excepciones.
3. Dentro de la modificación de usuarios, el usuario puede introducir datos que no correspondan con los requisitos de cada sección o introducir datos que supongan campos únicos dentro de la base de datos. Este error se controla mediante los métodos dentro de la clase Utils.
4. En la sección de cargar laberinto, el usuario puede seleccionar un laberinto que no se muestre ni exista, el control de este error se hace a partir de condicionales.

5. En el método de establecer casillas de entrada y salida el usuario podría introducir casillas que se encuentran fuera del rango posible del laberinto, “paredes” dentro del laberinto, o la misma casilla de entrada que de salida. Estos errores se controlan mediante condicionales y también desde tratamiento de excepciones a la hora del error de fuera de rango.
6. El usuario puede intentar establecer casillas de entrada y salida o intentar resolver el laberinto sin haber cargado previamente un laberinto, este error se controla mediante condicionales.
7. El usuario puede intentar resolver un laberinto cargado, pero sin haber establecido previamente casillas de entrada y salida. Este error se controla mediante condicionales.

Hemos estado hablando bastante de modificaciones de los datos de usuarios, creación de usuarios y eliminación de perfiles de usuario. Y como se ha comentado hay acciones compartidas que se pueden realizar mediante un usuario registrado o un administrador.

Todos los usuarios tienen un rol, que puede ser user o admin. Un user puede modificar o borrar únicamente su propio usuario, mientras que un admin puede actuar creando, borrando o modificando los campos de cualquier usuario existente en el sistema, así como listar los usuarios existentes.

Los roles de admin solo puede asignarlos el administrador del SGBD, por lo que estos roles están claramente diferenciados.

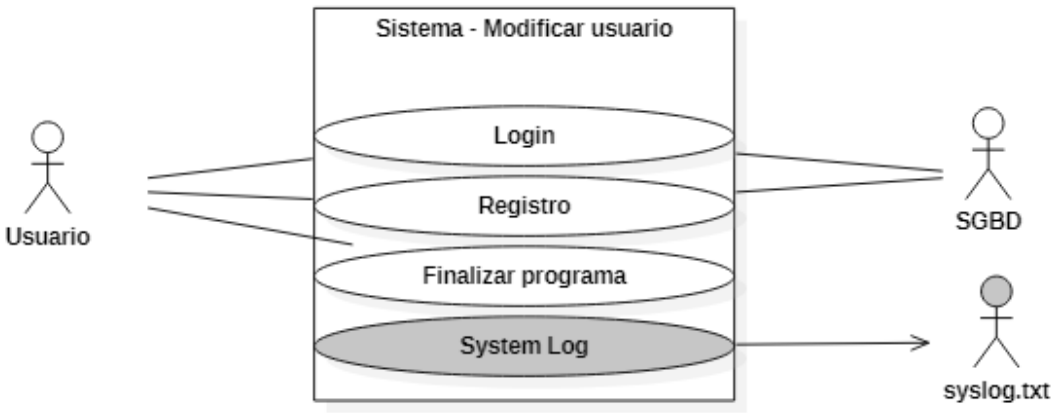
DEFINICION UML DEL PROGRAMA

Nos encontramos con diferentes actores y usos dentro del sistema dependiendo del estado del mismo. Principalmente se diferencian dos escenarios con matices dependiendo del actor que está comunicándose con el programa, ya que puede ser un usuario, o un usuario con rol de administrador.

Adicionalmente podemos encontrar dos escenarios más que veremos.

A continuación, vamos a analizar el primer escenario:

UNLOGGED MENU



ACTORES

Para este escenario encontramos tres actores.

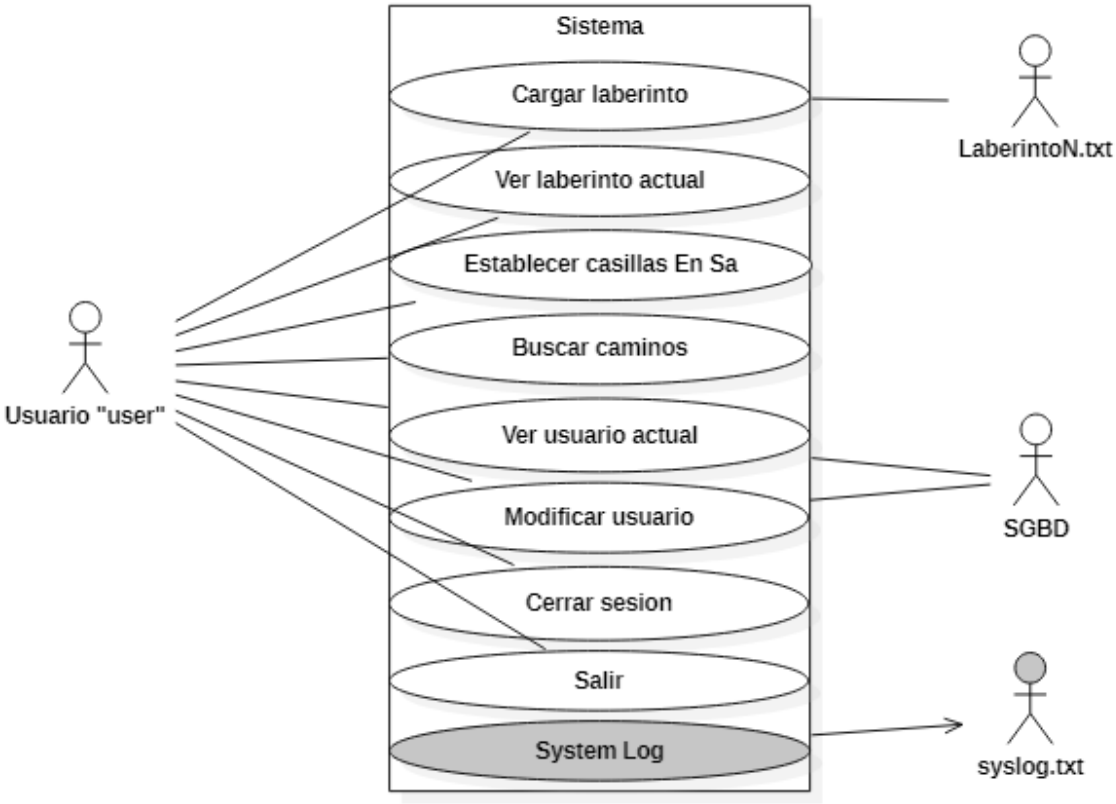
1. El usuario, que es el actor que interactúa de forma activa con el sistema.
2. El SGBD interactúa con el sistema como gestor de información sobre el que el sistema se va a apoyar siendo capaz de gestionar el sistema de usuarios. Esta comunicación es mutua, ya que el sistema es el encargado de realizar las consultas pertinentes al SGBD para el caso de login como la escritura de nuevos datos sobre la base de datos de users que alberga el SGBD.
3. Syslog.txt es un actor pasivo, ya que el sistema es el que se comunica con syslog.txt para almacenar la información de los eventos del sistema pertinentes.

CASOS DE USO

1. Login: el usuario que está en comunicación con el sistema puede hacer login y autenticarse. Para ello, el sistema pedirá los datos pertinentes al usuario y hará una consulta sobre la base de datos para comparar los datos introducidos por el usuario con los datos obtenidos de la base de datos. Si la autenticación ha sido un éxito el sistema mostrará el menú para usuarios registrados, que dependiendo del rol del usuario puede ser diferente. Hablaremos de ello más adelante
2. Registro: El sistema pide los datos pertinentes y necesarios al usuario que quiere realizar un registro, conforme el usuario va proporcionando la información necesaria al sistema, este va comprobando que los datos introducidos sean válidos, así como que los datos introducidos altamente sensibles no se vean repetidos en la base de datos. Es decir, realiza consultas sobre la base de datos para evitar la redundancia de información establecida por el sistema y el SGBD. Una vez que el sistema tiene toda la información y comprobado que los datos son válidos los introduce en la base de datos para asegurar su consistencia.
3. Finalizar el programa: El usuario puede seleccionar esta opción ofrecida por el sistema. El programa finaliza con un mensaje de despedida.

LOGGED MENU (ROL USER)

Este escenario sucede cuando un usuario se autentifica de forma correcta. El programa comprueba y determina que el rol del usuario autenticado es de tipo “user” y muestra el menú pertinente para un usuario con este rol.



ACTORES

1. El usuario, que es el actor que interactúa de forma activa con el sistema, al tener rol “user” tiene acceso a las opciones básicas.
2. LaberintoN.txt almacena el contenido de los laberintos, siendo N el número de laberinto al que corresponde (Ej: 1, 2, 3…) el sistema se comunica con estos ficheros para leer la información que contienen a la hora de “Cargar laberinto”.
3. El SGBD interactúa con el sistema como gestor de información sobre el que el sistema se va a apoyar siendo capaz de gestionar el sistema de usuarios. Esta comunicación es mutua, ya que el sistema es el encargado de realizar las consultas pertinentes al SGBD como la escritura de nuevos datos sobre la tabla users de la base de datos para el sistema que alberga el SGBD.
4. Syslog.txt es un actor pasivo, ya que el sistema es el que se comunica con syslog.txt para almacenar la información de los eventos del sistema pertinentes.

CASOS DE USO

1. Cargar laberinto: El usuario puede seleccionar la opción ofrecida por el sistema de “Cargar laberinto”, el sistema leerá el directorio donde se encuentran los ficheros que almacenan la información de los laberintos y ofrece una lista con todos los laberintos disponibles. Cuando el usuario selecciona uno, el sistema carga el contenido del fichero correspondiente en memoria.

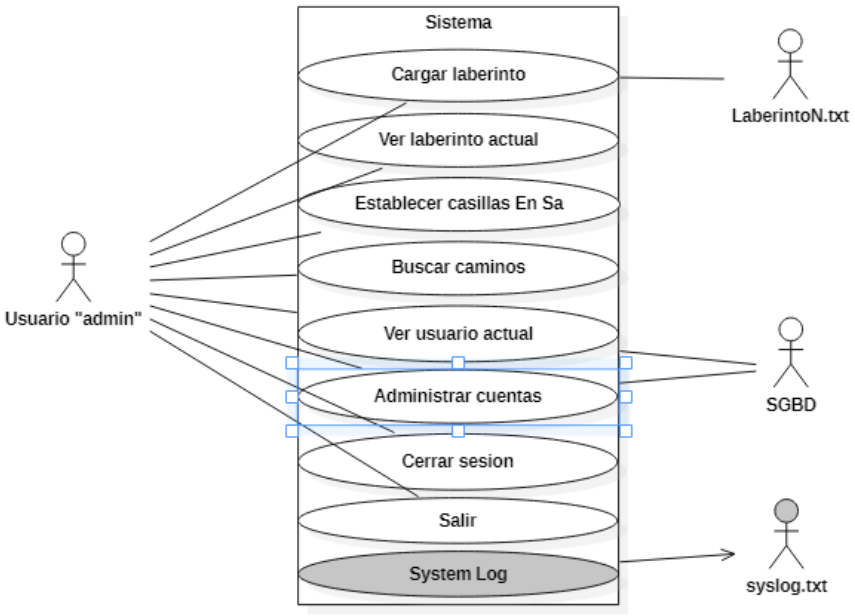
2. Ver laberinto actual: El usuario puede seleccionar la opción ofrecida por el sistema de “Ver laberinto actual”, el sistema comprobará si existe un laberinto cargado previamente en memoria, si existe comprobará si el laberinto ha sido resuelto para determinar si debe mostrar el resultado de la solución o solamente el laberinto cargado en memoria. En caso de que no exista un laberinto cargado previamente, el sistema avisará de ello.
3. Establecer casillas de Entrada y Salida: El usuario puede seleccionar la opción ofrecida por el sistema de “Establecer casillas de Entrada y Salida. Seleccionada esta opción el sistema comprobará que se encuentra un laberinto cargado en memoria previamente, si no se encuentra un laberinto cargado arrojará un mensaje informativo y volverá al menú previo. Si el sistema detecta que se ha cargado un laberinto, pedirá la inserción de las coordenadas de entrada y salida por parte del usuario, que, tras verificar que son válidas, es decir, que las casillas se encuentren dentro del rango del laberinto y que ninguna casilla se corresponde con una pared, establece dichas coordenadas en memoria.
4. Buscar caminos: El usuario puede seleccionar la opción ofrecida por el sistema de “Buscar caminos”. Al seleccionar esta opción el sistema muestra un menú con dos opciones, que para el sistema determina el algoritmo que usar a la hora de buscar el camino a la salida:
    1) El primer camino posible: Cuando el usuario selecciona esta opción el sistema comprueba si hay un laberinto previamente cargado y a demás si ese laberinto cargado contiene casillas establecidas de entrada y salida para poder determinar el camino a seguir por al algoritmo. En caso de no haber cargado un laberinto o no haber establecido casillas de entrada o salida el sistema arrojará un aviso al usuario. Si esas dos comprobaciones han resultado un éxito, ejecuta el algoritmo para encontrar la salida sobre el laberinto seleccionado con las coordenadas seleccionadas y se detiene en la primera vez que encuentra la salida, quedándose con un único camino posible.
    2) El camino más corto: Cuando el usuario selecciona esta opción el sistema comprueba si hay un laberinto previamente cargado y a demás si ese laberinto cargado contiene casillas establecidas de entrada y salida para poder determinar el camino a seguir por al algoritmo. En caso de no haber cargado un laberinto o no haber establecido casillas de entrada o salida el sistema arrojará un aviso al usuario. Si esas dos comprobaciones han resultado un éxito, ejecuta el algoritmo para encontrar la salida sobre el laberinto seleccionado con las coordenadas seleccionadas y no se detiene hasta que llega de nuevo a la entrada. Cada vez que guarda la salida compara los pasos almacenados que ha tenido que dar para resolver el laberinto, y si la nueva solución es más corta que la anterior

(o si es la primera vez que ha encontrado la salida), guarda esta solución en memoria como resultado válido.

5. Ver usuario actual: El usuario puede seleccionar la opción ofrecida por el sistema de “Ver usuario actual”. El sistema se comunica con la base de datos y consulta todos los registros establecidos para esta opción (todas las propiedades del usuario a excepción de la contraseña) y los guarda en memoria para mostrarlos por pantalla.
6. Modificar usuario: El usuario puede seleccionar la opción ofrecida por el sistema de “Modificar usuario”. El sistema despliega un nuevo menú con todas las opciones de todos los atributos que se permiten modificar del usuario autenticado en ese momento en el sistema. Más adelante se entrará más en detalle sobre cada una de las opciones, quedando nombradas serían las siguientes:
    1) Cambiar contraseña
    2) Cambiar nombre
    3) Cambiar NIF
    4) Cambiar email
    5) Cambiar dirección
    6) Cambiar fecha de nacimiento
    7) Eliminar cuenta
    8) Cancelar
7. Cerrar sesión: El usuario puede seleccionar la opción ofrecida por el sistema de “Cerrar Sesión”. El sistema borra los datos de usuario previamente autenticado de memoria y establece que no se encuentra ningún usuario autenticado. Esta opción devuelve al usuario al primer menú de todos, que solo permite hacer login, registrarse o finalizar programa.

LOGGED MENU (ROL ADMIN)

Este escenario sucede cuando un usuario se autentifica de forma correcta. El programa comprueba y determina que el rol del usuario autenticado es de tipo “admin” y muestra el menú pertinente para un usuario con este rol.



ACTORES

1. El usuario, que es el actor que interactúa de forma activa con el sistema, al tener rol “admin” tiene acceso tanto a las opciones básicas como a un menú específico para administrar cualquier cuenta de usuario.
2. LaberintoN.txt almacena el contenido de los laberintos, siendo N el número de laberinto al que corresponde (Ej: 1, 2, 3…) el sistema se comunica con estos ficheros para leer la información que contienen a la hora de “Cargar laberinto”.
3. El SGBD interactúa con el sistema como gestor de información sobre el que el sistema se va a apoyar siendo capaz de gestionar el sistema de usuarios. Esta comunicación es mutua, ya que el sistema es el encargado de realizar las consultas pertinentes al SGBD como la escritura de nuevos datos sobre la tabla users de la base de datos para el sistema que alberga el SGBD.
4. Syslog.txt es un actor pasivo, ya que el sistema es el que se comunica con syslog.txt para almacenar la información de los eventos del sistema pertinentes.

CASOS DE USO

1. Cargar laberinto: El usuario puede seleccionar la opción ofrecida por el sistema de “Cargar laberinto”, el sistema leerá el directorio donde se encuentran los ficheros que almacenan la información de los laberintos y ofrece una lista con todos los laberintos disponibles. Cuando el usuario selecciona uno, el sistema carga el contenido del fichero correspondiente en memoria.
2. Ver laberinto actual: El usuario puede seleccionar la opción ofrecida por el sistema de “Ver laberinto actual”, el sistema comprobará si existe un laberinto cargado previamente en memoria, si existe comprobará si el laberinto ha sido resuelto para determinar si debe mostrar el resultado de la solución o solamente el laberinto cargado en memoria. En caso de que no exista un laberinto cargado previamente, el sistema avisará de ello.
3. Establecer casillas de Entrada y Salida: El usuario puede seleccionar la opción ofrecida por el sistema de “Establecer casillas de Entrada y Salida. Seleccionada esta opción el sistema comprobará que se encuentra un laberinto cargado en memoria previamente, si no se encuentra un laberinto cargado arrojará un mensaje informativo y volverá al menú previo. Si el sistema detecta que se ha cargado un laberinto, pedirá la inserción de las coordenadas de entrada y salida por parte del usuario, que, tras verificar que son válidas, es decir, que las casillas se encuentren dentro del rango del laberinto y que ninguna casilla se corresponde con una pared, establece dichas coordenadas en memoria.
4. Buscar caminos: El usuario puede seleccionar la opción ofrecida por el sistema de “Buscar caminos”. Al seleccionar esta opción el sistema muestra un menú con dos opciones, que para el sistema determina el algoritmo que usar a la hora de buscar el camino a la salida:

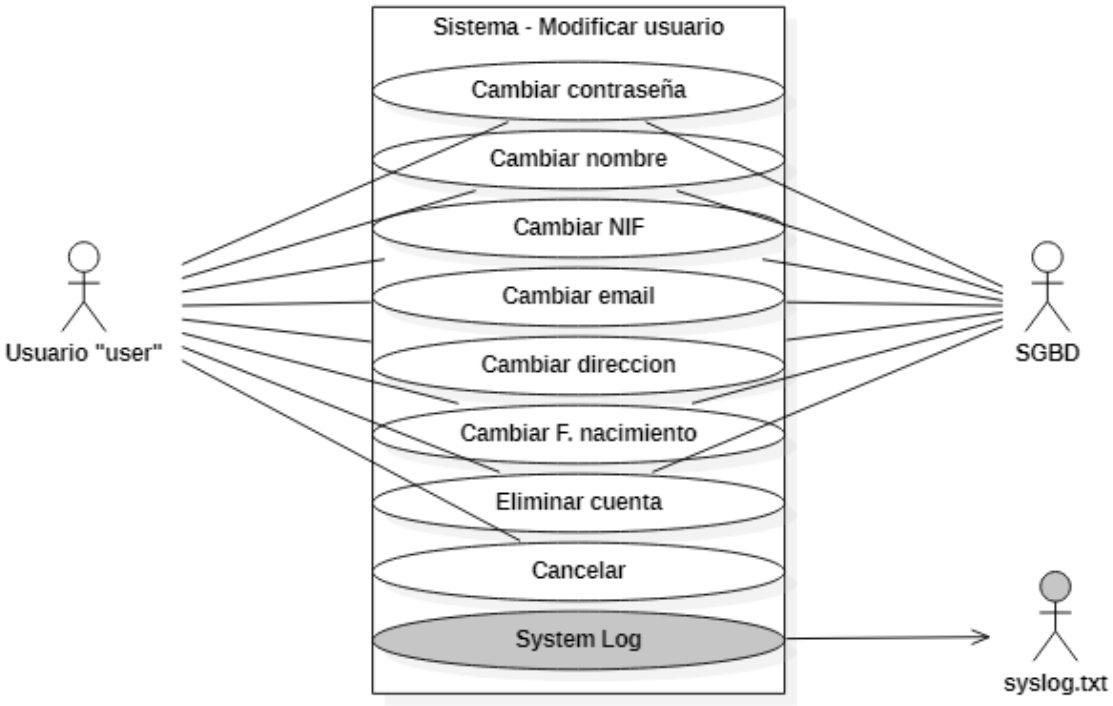
1) El primer camino posible: Cuando el usuario selecciona esta opción el sistema comprueba si hay un laberinto previamente cargado y a demás si ese laberinto cargado contiene casillas establecidas de entrada y salida para poder determinar el camino a seguir por al algoritmo. En caso de no haber cargado un laberinto o no haber establecido casillas de entrada o salida el sistema arrojará un aviso al usuario. Si esas dos comprobaciones han resultado un éxito, ejecuta el algoritmo para encontrar la salida sobre el laberinto seleccionado con las coordenadas seleccionadas y se detiene en la primera vez que encuentra la salida, quedándose con un único camino posible.
2) El camino más corto: Cuando el usuario selecciona esta opción el sistema comprueba si hay un laberinto previamente cargado y a demás si ese laberinto cargado contiene casillas establecidas de entrada y salida para poder determinar el camino a seguir por al algoritmo. En caso de no haber cargado un laberinto o no haber establecido casillas de entrada o salida el sistema arrojará un aviso al usuario. Si esas dos comprobaciones han resultado un éxito, ejecuta el algoritmo para encontrar la salida sobre el laberinto seleccionado con las coordenadas seleccionadas y no se detiene hasta que llega de nuevo a la entrada. Cada vez que guarda la salida compara los pasos almacenados que ha tenido que dar para resolver el laberinto, y si la nueva solución es más corta que la anterior (o si es la primera vez que ha encontrado la salida), guarda esta solución en memoria como resultado válido.

5. Ver usuario actual: El usuario puede seleccionar la opción ofrecida por el sistema de “Ver usuario actual”. El sistema se comunica con la base de datos y consulta todos los registros establecidos para esta opción (todas las propiedades del usuario a excepción de la contraseña) y los guarda en memoria para mostrarlos por pantalla.
6. Administrar cuentas: El usuario puede seleccionar la opción ofrecida por el sistema de “Administrar cuentas”. El sistema despliega un nuevo menú con todas las opciones de todas las funciones que puede hacer el usuario con rol “admin” sobre el sistema de usuarios. Más adelante se entrará más en detalle sobre cada una de las opciones, quedando nombradas serían las siguientes:
    1) Crear nuevo usuario
    2) Ver todos los usuarios
    3) Cambiar contraseña
    4) Cambiar nombre
    5) Cambiar NIF
    6) Cambiar email
    7) Cambiar dirección
    8) Cambiar fecha de nacimiento
    9) Eliminar usuario
    10) Cancelar

7. Cerrar sesión: El usuario puede seleccionar la opción ofrecida por el sistema de “Cerrar Sesión”. El sistema borra los datos de usuario previamente autenticado de memoria y establece que no se encuentra ningún usuario autenticado. Esta opción devuelve al usuario al primer menú de todos, que solo permite hacer login, registrarse o finalizar programa.

MODIFY MENU (ROL USER)

El usuario puede acceder a este menú al seleccionar la opción 6 – “Modificar usuario” dentro de “LoggedMenu” para el rol “user”. Este menú contiene las opciones sobre las que el sistema permite modificar el usuario actualmente autenticado. El sistema se encuentra en estrecho contacto tanto con el usuario como con el SGBD.



ACTORES

1. El usuario, que es el actor que interactúa de forma activa con el sistema, al tener rol “user” tiene acceso a opciones que únicamente repercuten en su propio perfil.
2. El SGBD interactúa con el sistema como gestor de información sobre el que el sistema se va a apoyar siendo capaz de gestionar el sistema de usuarios. Esta comunicación es mutua, ya que el sistema es el encargado de realizar las consultas pertinentes al SGBD como la escritura de nuevos datos sobre la tabla users de la base de datos para el sistema que alberga el SGBD.
3. Syslog.txt es un actor pasivo, ya que el sistema es el que se comunica con syslog.txt para almacenar la información de los eventos del sistema pertinentes.

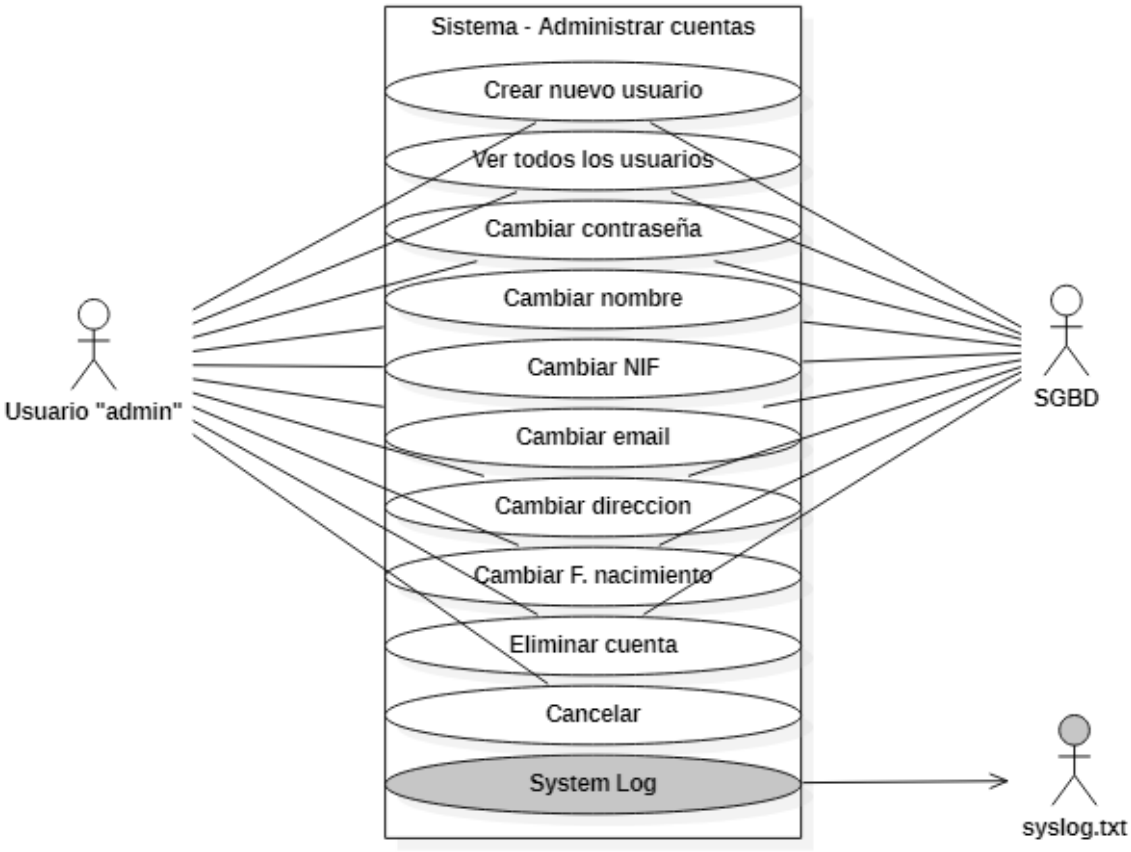
CASOS DE USO

1. Cambiar contraseña: El usuario puede seleccionar la opción ofrecida por el sistema de “Cambiar contraseña”. Al seleccionar esta opción el sistema requerirá la inserción de la nueva contraseña por parte del usuario, comprobará que la contraseña introducida cumple los requisitos establecidos por el sistema, en caso de que no sea así mostrará un aviso y volverá a “LoggedMenu” en caso de que cumpla con los requisitos pedirá de nuevo la nueva contraseña y comprobará que ambas coinciden. En caso de que no coincidan se mostrará un aviso por pantalla y volverá a “LoggedMenu”, en caso de que coincidan encriptará la nueva contraseña y la sustituirá en la tabla users dentro de la base de datos.
2. Cambiar nombre: El usuario puede seleccionar la opción ofrecida por el sistema de “Cambiar nombre”. Al seleccionar esta opción el sistema requerirá la inserción del nuevo nombre por parte del usuario y comprobará que el nombre introducido cumple con los requisitos establecidos por el sistema. En caso de que no cumpla dichos requisitos mostrará un aviso por pantalla y regresará a “LoggedMenu”. En caso de que sí que cumpla con los requisitos sustituirá el nombre antiguo por el nuevo dentro de la tabla users de la base de datos.
3. Cambiar NIF: El usuario puede seleccionar la opción ofrecida por el sistema de “Cambiar NIF”. Al seleccionar esta opción el sistema requerirá la inserción del nuevo NIF por parte del usuario y comprobará que el NIF introducido cumple con los requisitos establecidos por el sistema. En caso de que no cumpla dichos requisitos mostrará un aviso por pantalla y regresará a “LoggedMenu”. En caso de que sí que cumpla con los requisitos el sistema hará una consulta para comprobar si el NIF establecido ya se encuentra dentro de la tabla users en la base de datos, ya que este campo supone un campo único dentro de la tabla y no puede contener redundancia. Si ya existe el sistema mostrará un aviso y regresará a “LoggedMenu”, y si no existe sustituirá el valor del campo NIF en la tabla users dentro de la base de datos.
4. Cambiar email: El usuario puede seleccionar la opción ofrecida por el sistema de “Cambiar email”. Al seleccionar esta opción el sistema requerirá la inserción del nuevo email por parte del usuario y comprobará que el email introducido cumple con los requisitos establecidos por el sistema. En caso de que no cumpla dichos requisitos mostrará un aviso por pantalla y regresará a “LoggedMenu”. En caso de que sí que cumpla con los requisitos del sistema hará una consulta para comprobar si el email establecido ya se encuentra dentro de la tabla users en la base de datos, ya que este campo supone un campo único dentro de la tabla y no puede contener redundancia. Si ya existe el sistema mostrará un aviso y regresará a “LoggedMenu”, y si no existe sustituirá el valor del campo email en la tabla users dentro de la base de datos.

5. Cambiar dirección: El usuario puede seleccionar la opción ofrecida por el sistema de “Cambiar dirección”. Al seleccionar esta opción el sistema requerirá la inserción de la nueva dirección por parte del usuario. A diferencia de las demás opciones, para esta opción no realizará ninguna comprobación, escribirá esta información dentro de la tabla users en la base de datos.
6. Cambiar fecha de nacimiento: El usuario puede seleccionar la opción ofrecida por el sistema de “Cambiar fecha de nacimiento”. Al seleccionar esta opción el sistema requerirá la inserción de la nueva fecha de nacimiento por parte del usuario, usando un formato “dd/MM/aaaa”. Una vez introducida la fecha de nacimiento el sistema verifica de que se trata de una fecha válida. En caso de que no sea una fecha válida el sistema mostrará un aviso y volverá a “LoggedMenú”. Si la fecha si es válida convertirá el formato de la fecha a “aaaa-MM-dd”, fecha legible para el SGBD y sustituirá el valor antiguo por el nuevo dentro de la tabla users de la base de datos.
7. Eliminar cuenta: El usuario puede seleccionar la opción ofrecida por el sistema de “Eliminar cuenta”. Al seleccionar esta opción lo primero que hace el sistema es enviar un aviso y preguntar al usuario si desea seguir con el proceso. En caso negativo el sistema vuelve a “LoggedMenu”, en caso afirmativo requiere de la contraseña del usuario actualmente autenticado. Al introducir el usuario la contraseña el sistema compara la contraseña introducida con la contraseña almacenada en la base de datos. En caso de que las contraseñas no coincidan mostrará un aviso y regresará a “LoggedMenu”, en caso de que coincidan mostrará de nuevo el aviso de si desea continuar con el proceso. En caso negativo el sistema mostrará un aviso y regresará a “LoggedMenu”, en caso afirmativo eliminará los datos del usuario de la base de datos, limpiará los datos del usuario guardados en memoria y volverá a “UnloggedMenu”.
8. Cancelar: El usuario puede seleccionar la opción ofrecida por el sistema de “Cancelar”. Al seleccionar esta opción el sistema volverá a “LoggedMenu”.

ADMINISTRAR CUENTAS

El usuario puede acceder a este menú al seleccionar la opción 6 – “Administrar cuentas” dentro de “LoggedMenu” para el rol “admin”. Este menú contiene las opciones sobre las que el sistema permite actuar sobre el sistema de usuarios. El sistema se encuentra en estrecho contacto tanto con el usuario como con el SGBD.



ACTORES

1. El usuario, que es el actor que interactúa de forma activa con el sistema, al tener rol “admin” tiene acceso a sus propias opciones como a las de cualquier usuario, esté o no registrado en el sistema.
2. El SGBD interactúa con el sistema como gestor de información sobre el que el sistema se va a apoyar siendo capaz de gestionar el sistema de usuarios. Esta comunicación es mutua, ya que el sistema es el encargado de realizar las consultas pertinentes al SGBD como la escritura de nuevos datos sobre la tabla users de la base de datos para el sistema que alberga el SGBD.
3. Syslog.txt es un actor pasivo, ya que el sistema es el que se comunica con syslog.txt para almacenar la información de los eventos del sistema pertinentes.

CASOS DE USO

1. Crear nuevo usuario: El usuario puede seleccionar la opción ofrecida por el sistema de “Crear nuevo usuario”. Al seleccionar esta opción el sistema pide los datos pertinentes y necesarios al usuario administrador para la creación del nuevo usuario. Conforme el usuario va proporcionando la información necesaria al sistema, este va comprobando que los datos introducidos sean válidos, así como que los datos introducidos altamente sensibles no se vean repetidos en la base de datos. Es decir, realiza

consultas sobre la base de datos para evitar la redundancia de información establecida por el sistema y el SGBD. Una vez que el sistema tiene toda la información y comprobado que los datos son válidos los introduce en la base de datos para asegurar su consistencia.

2. Ver todos los usuarios: El usuario puede seleccionar la opción ofrecida por el sistema de “Ver todos los usuarios”. Al seleccionar esta opción el sistema consulta al SGBD todos los registros de la tabla users, los almacena en memoria y los muestra por pantalla.
3. Cambiar contraseña: El usuario puede seleccionar la opción ofrecida por el sistema de “Cambiar contraseña”. Al seleccionar esta opción el sistema requiere la inserción al usuario sobre el nombre de usuario concreto del que se quiere actuar. Si el usuario no existe el sistema mostrará un aviso y volverá a “LoggedMenu”, por el contrario, el usuario pide los datos al SGBD y lo carga en memoria. Una vez cargado el usuario el sistema requerirá la inserción de la nueva contraseña por parte del usuario, comprobará que la contraseña introducida cumple los requisitos establecidos por el sistema, en caso de que no sea así mostrará un aviso y volverá a “LoggedMenu” en caso de que cumpla con los requisitos pedirá de nuevo la nueva contraseña y comprobará que ambas coinciden. En caso de que no coincidan se mostrará un aviso por pantalla y volverá a “LoggedMenu”, en caso de que coincidan encriptará la nueva contraseña y la sustituirá en la tabla users dentro de la base de datos.
4. Cambiar nombre: El usuario puede seleccionar la opción ofrecida por el sistema de “Cambiar nombre”. Al seleccionar esta opción el sistema requiere la inserción al usuario sobre el nombre de usuario concreto del que se quiere actuar. Si el usuario no existe el sistema mostrará un aviso y volverá a “LoggedMenu”, por el contrario, el usuario pide los datos al SGBD y lo carga en memoria. Una vez cargado el perfil en memoria el sistema requerirá la inserción del nuevo nombre por parte del usuario y comprobará que el nombre introducido cumple con los requisitos establecidos por el sistema. En caso de que no cumpla dichos requisitos mostrará un aviso por pantalla y regresará a “LoggedMenu”. En caso de que sí que cumpla con los requisitos sustituirá el nombre antiguo por el nuevo dentro de la tabla users de la base de datos.
5. Cambiar NIF: El usuario puede seleccionar la opción ofrecida por el sistema de “Cambiar NIF”. Al seleccionar esta opción el sistema requiere la inserción al usuario sobre el nombre de usuario concreto del que se quiere actuar. Si el usuario no existe el sistema mostrará un aviso y volverá a “LoggedMenu”, por el contrario, el usuario pide los datos al SGBD y lo carga en memoria. Una vez cargado el perfil en memoria el sistema requerirá la inserción del nuevo NIF por parte del usuario y comprobará que el NIF introducido cumple con los requisitos establecidos por el sistema. En caso de que no cumpla dichos requisitos mostrará un aviso por pantalla y regresará a

"LoggedMenu". En caso de que sí que cumpla con los requisitos el sistema hará una consulta para comprobar si el NIF establecido ya se encuentra dentro de la tabla users en la base de datos, ya que este campo supone un campo único dentro de la tabla y no puede contener redundancia. Si ya existe el sistema mostrará un aviso y regresará a "LoggedMenu", y si no existe sustituirá el valor del campo NIF en la tabla users dentro de la base de datos.

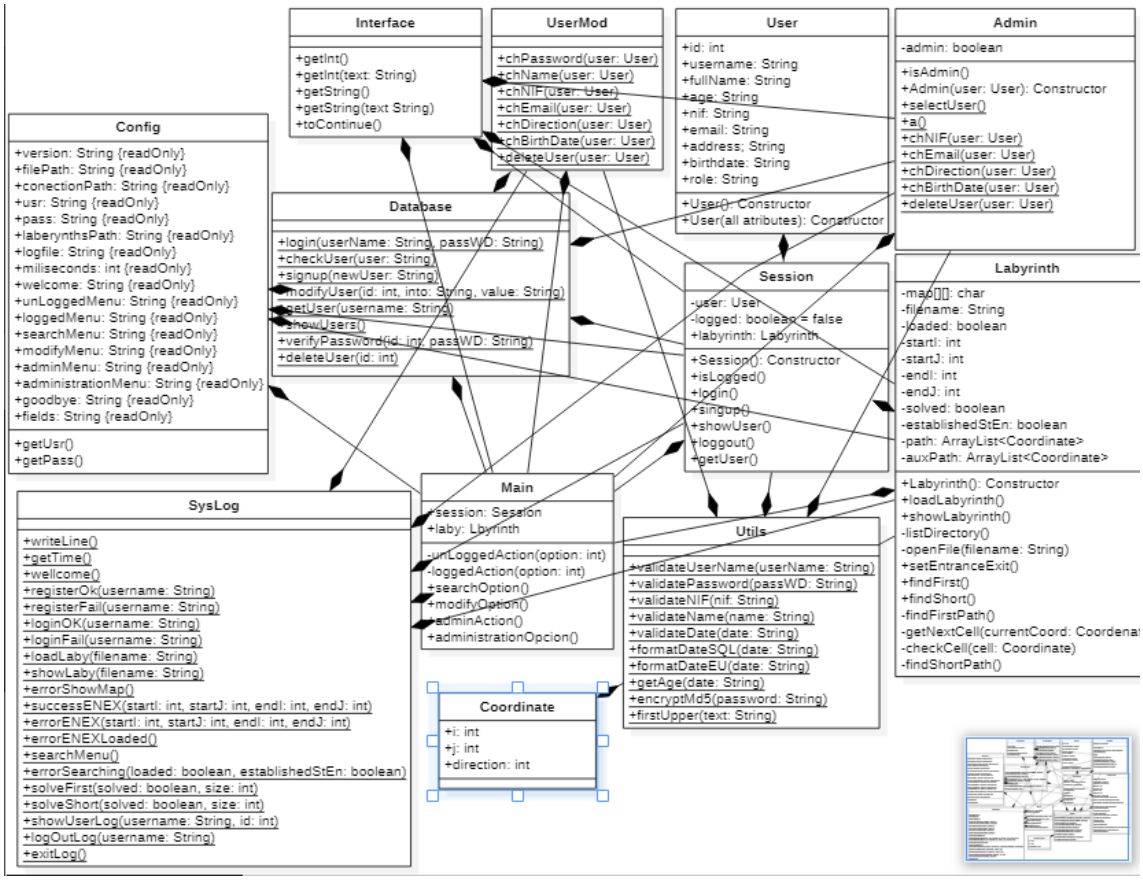
6. Cambiar email: El usuario puede seleccionar la opción ofrecida por el sistema de "Cambiar email". Al seleccionar esta opción el sistema requiere la inserción al usuario sobre el nombre de usuario concreto del que se quiere actuar. Si el usuario no existe el sistema mostrará un aviso y volverá a "LoggedMenu", por el contrario, el usuario pide los datos al SGBD y lo carga en memoria. Una vez cargado el perfil en memoria el sistema requerirá la inserción del nuevo email por parte del usuario y comprobará que el email introducido cumple con los requisitos establecidos por el sistema. En caso de que no cumpla dichos requisitos mostrará un aviso por pantalla y regresará a "LoggedMenu". En caso de que sí que cumpla con los requisitos del sistema hará una consulta para comprobar si el email establecido ya se encuentra dentro de la tabla users en la base de datos, ya que este campo supone un campo único dentro de la tabla y no puede contener redundancia. Si ya existe el sistema mostrará un aviso y regresará a "LoggedMenu", y si no existe sustituirá el valor del campo email en la tabla users dentro de la base de datos.
7. Cambiar dirección: El usuario puede seleccionar la opción ofrecida por el sistema de "Cambiar dirección". Al seleccionar esta opción el sistema requiere la inserción al usuario sobre el nombre de usuario concreto del que se quiere actuar. Si el usuario no existe el sistema mostrará un aviso y volverá a "LoggedMenu", por el contrario, el usuario pide los datos al SGBD y lo carga en memoria. Una vez cargado el perfil en memoria el sistema requerirá la inserción de la nueva dirección por parte del usuario. A diferencia de las demás opciones, para esta opción no realizará ninguna comprobación, escribirá esta información dentro de la tabla users en la base de datos.
8. Cambiar fecha de nacimiento: El usuario puede seleccionar la opción ofrecida por el sistema de "Cambiar fecha de nacimiento". Al seleccionar esta opción el sistema requiere la inserción al usuario sobre el nombre de usuario concreto del que se quiere actuar. Si el usuario no existe el sistema mostrará un aviso y volverá a "LoggedMenu", por el contrario, el usuario pide los datos al SGBD y lo carga en memoria. Una vez cargado el perfil en memoria el sistema requerirá la inserción de la nueva fecha de nacimiento por parte del usuario, usando un formato "dd/MM/aaaa". Una vez introducida la fecha de nacimiento el sistema verifica de que se trata de una fecha válida. En caso de que no sea una fecha válida el sistema mostrará un aviso y volverá a "LoggedMenú". Si la fecha si es válida convertirá el

formato de la fecha a “aaaa-MM-dd”, fecha legible para el SGBD y sustituirá el valor antiguo por el nuevo dentro de la tabla users de la base de datos.

9. Eliminar cuenta: El usuario puede seleccionar la opción ofrecida por el sistema de “Eliminar cuenta”. Al seleccionar esta opción el sistema requiere la inserción al usuario sobre el nombre de usuario concreto del que se quiere actuar. Si el usuario no existe el sistema mostrará un aviso y volverá a “LoggedMenu”, por el contrario, el usuario pide los datos al SGBD y lo carga en memoria. Una vez cargado el perfil en memoria el sistema lo primero que hace el sistema es enviar un aviso y preguntar al usuario si desea seguir con el proceso. En caso negativo el sistema vuelve a “LoggedMenu”, en caso afirmativo requiere de la contraseña del usuario actualmente autenticado, es decir, la contraseña del usuario admin. Al introducir el usuario la contraseña el sistema compara la contraseña introducida con la contraseña almacenada en la base de datos. En caso de que las contraseñas no coincidan mostrará un aviso y regresará a “LoggedMenu”, en caso de que coincidan mostrará de nuevo el aviso de si desea continuar con el proceso. En caso negativo el sistema mostrará un aviso y regresará a “LoggedMenu”, en caso afirmativo eliminará los datos del usuario de la base de datos, limpiará los datos del usuario guardados en memoria y volverá a “UnloggedMenu”.
10. Cancelar: El usuario puede seleccionar la opción ofrecida por el sistema de “Cancelar”. Al seleccionar esta opción el sistema volverá a “LoggedMenu”.

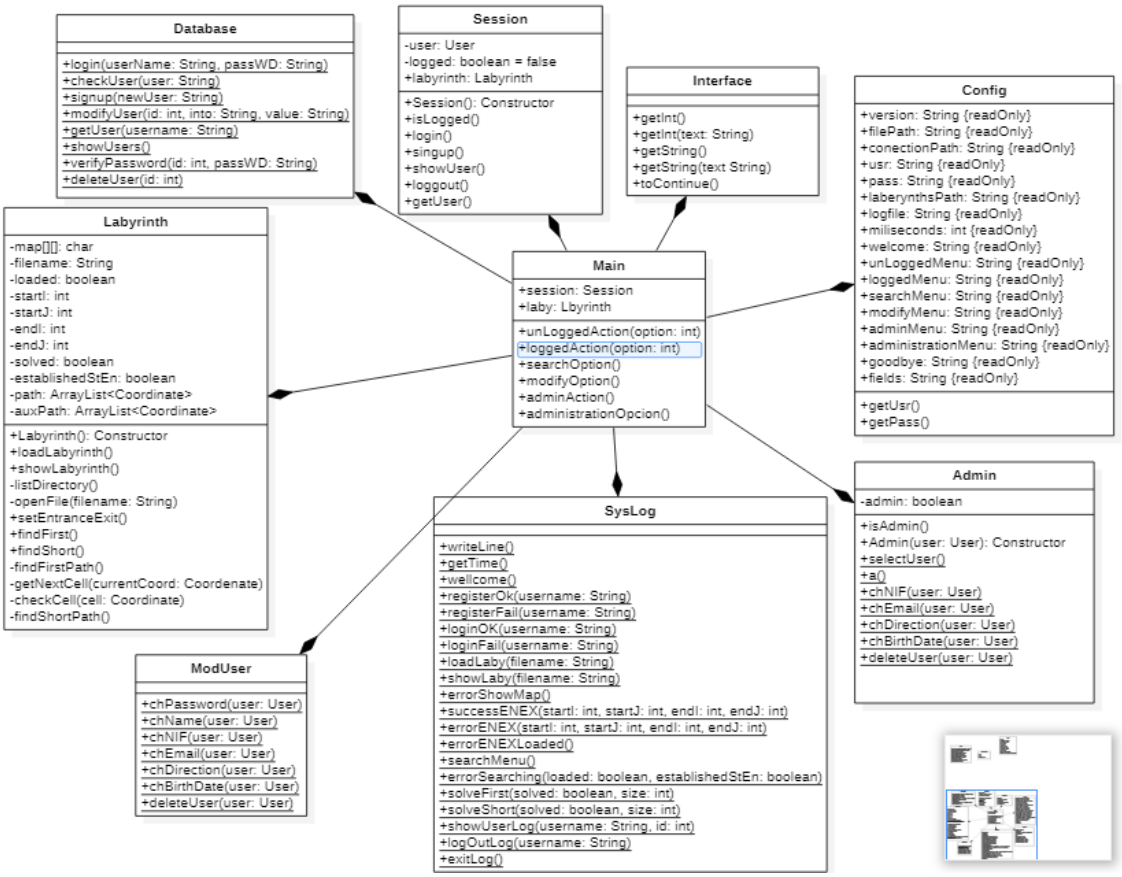
DEFINICION INDIVIDUAL DE CLASE DEL PROGRAMA

DIAGRAMA COMPLETO



Como se puede apreciar en la imagen, el diagrama de clases es imposible de explicar a no ser que lo hagamos por clases individuales, ya que hay bastantes clases y muchas conexiones entre ellas.

MAIN



Esta clase contiene el programa principal y desde la que se implementan todas las clases y métodos declarados necesarios para la función del programa, estas clases a su vez se conectan con otras clases y métodos para la correcta funcionalidad del sistema.

Esta clase se va a valer de atributos y métodos de las clases Database, Session, Interface, Config, Admin, Labyrinth y ModUser para el funcionamiento del programa, así como de los métodos de la clase SysLog para registrar todos los eventos del sistema pertinentes.

Esta clase tiene dos atributos declarados: un atributo laby de tipo Labyrinth y un atributo session de la clase Session.

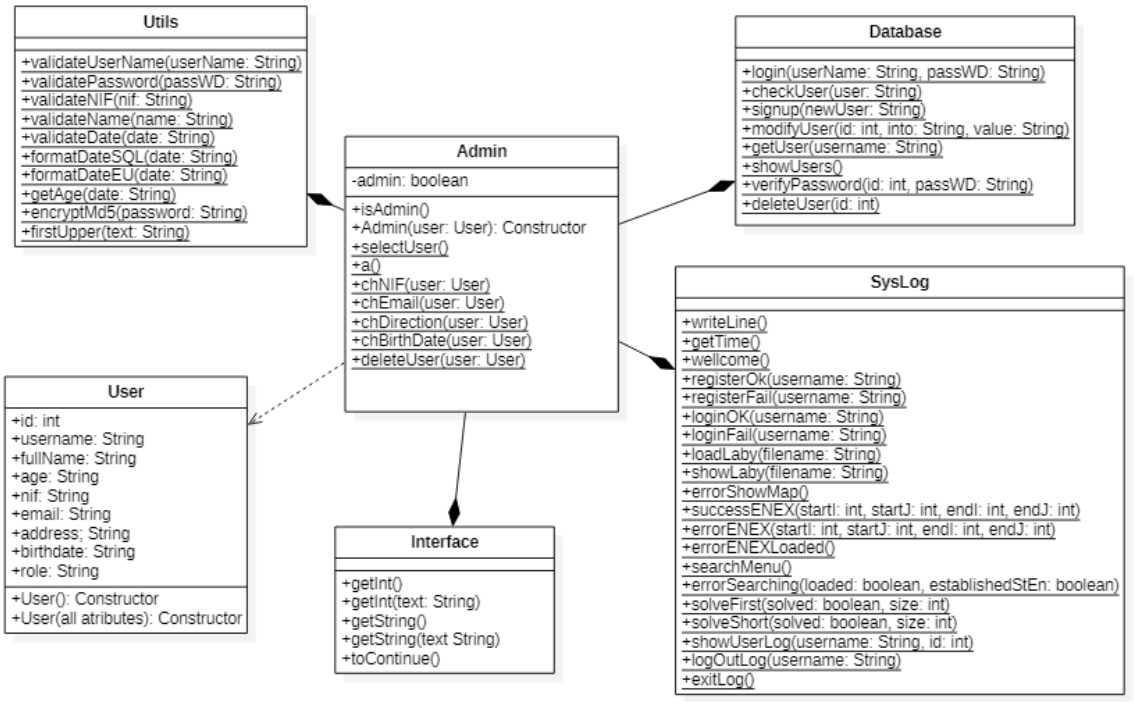
Los métodos de esta clase son:

- unLoggedAction(): Método público que recibe como parámetro una variable de tipo entero y contiene las acciones del menú principal para un usuario no autenticado.
- loggedAction() Método público que recibe como parámetro una variable de tipo entero y contiene las acciones del menú principal para un usuario autenticado con rol “user”.
- SearchOption(): Método público que recibe como parámetro una variable de tipo entero y contiene las acciones del menú para la resolución de un laberinto.

- ModifyOption(): Método público que recibe como parámetro una variable de tipo entero y contiene las acciones del menú para la modificación de un usuario con rol “user”.
- AdminAction(): Método público que recibe como parámetro una variable de tipo entero y contiene las acciones del menú principal para un usuario autenticado con rol “admin”.
- AdministrationOption(): Método público que recibe como parámetro una variable de tipo entero y contiene las acciones del menú para la gestión de usuarios desde un usuario autenticado con rol “admin”.

La clase Main se relaciona con Database, Session, Interface, Config, Admin, SysLog, ModUser y Labyrinth para cumplir con todas las funciones mencionadas.

ADMIN



Esta clase contiene los métodos necesarios para llevar a cabo las funciones principales que se encuentran dentro del menú “administrationMenu”, con el que se permite la administración del sistema de usuarios.

La clase Admin se va a apoyar en los atributos y métodos de las clases Database, Utils, Interface y User para llevar a cabo todas las funciones pertinentes, haciendo uso de los métodos de la clase SysLog para registrar todos los eventos del sistema.

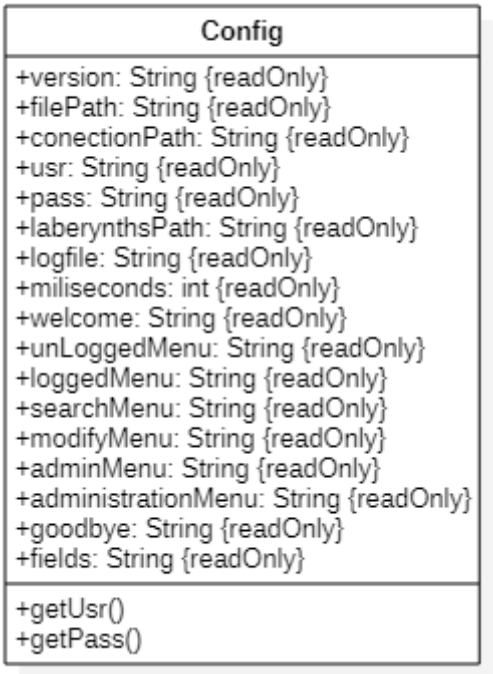
Esta clase contiene un atributo de tipo booleano cuya función es determinar si el usuario actualmente autenticado tiene como rol “admin”.

Métodos:

- IsAdmin(): Método público get del atributo “admin” que devuelve el valor de la variable “admin”.

- Admin(User user): Constructor de la clase que establece el atributo admin a true si el usuario tiene rol “admin” o false en caso contrario.
- selectUser(): Método público estático usado para seleccionar un determinado usuario de la base de datos y guardarlo en memoria.
- chPassword(User user): Método público estático que pide la inserción de una nueva contraseña para un determinado usuario con su repetición para determinar si se escribe en la base de datos.
- ChName(User user): Método público estático que pide la inserción de un nuevo nombre completo para un determinado usuario para determinar si se escribe en la base de datos.
- ChNIF(User user): Método público estático que pide la inserción de un nuevo NIF para un determinado usuario para determinar si se escribe en la base de datos.
- ChEmail(User User): Método público estático que pide la inserción de un nuevo email para un determinado usuario para determinar si se escribe en la base de datos.
- chDirection(User user): Método público estático que pide la inserción de una nueva direccion para un determinado usuario para escribirlo en la base de datos.
- chBirthDate(User user): Método público estático que pide la inserción de una nueva fecha de nacimiento para un determinado usuario para determinar si se escribe en la base de datos.
- deleteUser(User user, int myuser): Método público estático que pide la inserción de una nuevo nombre completo para un determinado usuario para determinar si se escribe en la base de datos.

CONFIG



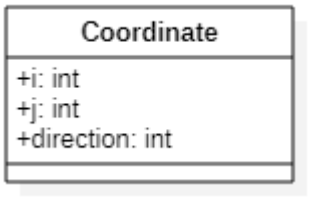
La clase Config contiene mucha información útil para el sistema, siendo una clase que no actúa sobre ninguna otra, sino que son las demás clases las que requieren de la información que contiene la clase, por lo que su principal contenido son atributos

Atributos:

- versión: Atributo público de tipo String de solo lectura que contiene la versión actual del programa (1.4.0 en este caso).
- filePath: Atributo público de tipo String de solo lectura que contiene la información de la ruta donde se ubican los archivos del sistema dentro del SO.
- ConnectionPath: Atributo público de tipo String de solo lectura que contiene la sentencia base de conexión para el SGBD.
- Usr: Atributo privado de tipo String de solo lectura que contiene el nombre de usuario para la conexión con el SGBD.
- Pass: Atributo privado de tipo String de solo lectura que contiene la contraseña usada para la conexión con el SGBD.
- LabyrinthsPath: Atributo público de tipo String de solo lectura que contiene la ruta donde se ubican los ficheros que contienen los laberintos dentro del sistema de ficheros del SO.
- Logfile: Atributo público de tipo String de solo lectura que contiene el nombre del fichero de log del sistema.
- Miliseconds: Atributo público de tipo int de solo lectura que contiene la cantidad en mili segundos que el sistema ha de esperar para la transición para alguno de los menús.
- Welcome: Atributo público de tipo String de solo lectura que contiene un mensaje de bienvenida.
- UnLoggedMenu: Atributo público de tipo String de solo lectura que contiene el texto para las opciones disponibles para el menú unloggedMenu.
- LoggedMenu: Atributo público de tipo String de solo lectura que contiene el texto para las opciones disponibles para el menú LoggedMenu.
- SearchMenu: Atributo público de tipo String de solo lectura que contiene el texto para las opciones disponibles para el menú de buscar caminos.
- ModifyMenu: Atributo público de tipo String de solo lectura que contiene el texto para las opciones disponibles para el menú de modificar usuario.
- AdminMenu: Atributo público de tipo String de solo lectura que contiene el texto para las opciones disponibles para el menú adminMenu.
- AdministrationMenu: Atributo público de tipo String de solo lectura que contiene el texto para las opciones disponibles para el menú Gestionar Cuentas.
- Goodbye: Atributo público de tipo String de solo lectura que contiene un mensaje de despedida.
- fields: Atributo público de tipo String[] de solo lectura que contiene el texto usado en el registro para cada uno de los campos.

Esta clase solamente contiene dos métodos GET para los atributos “usr” y “pass”, ya que son los dos únicos atributos privados.

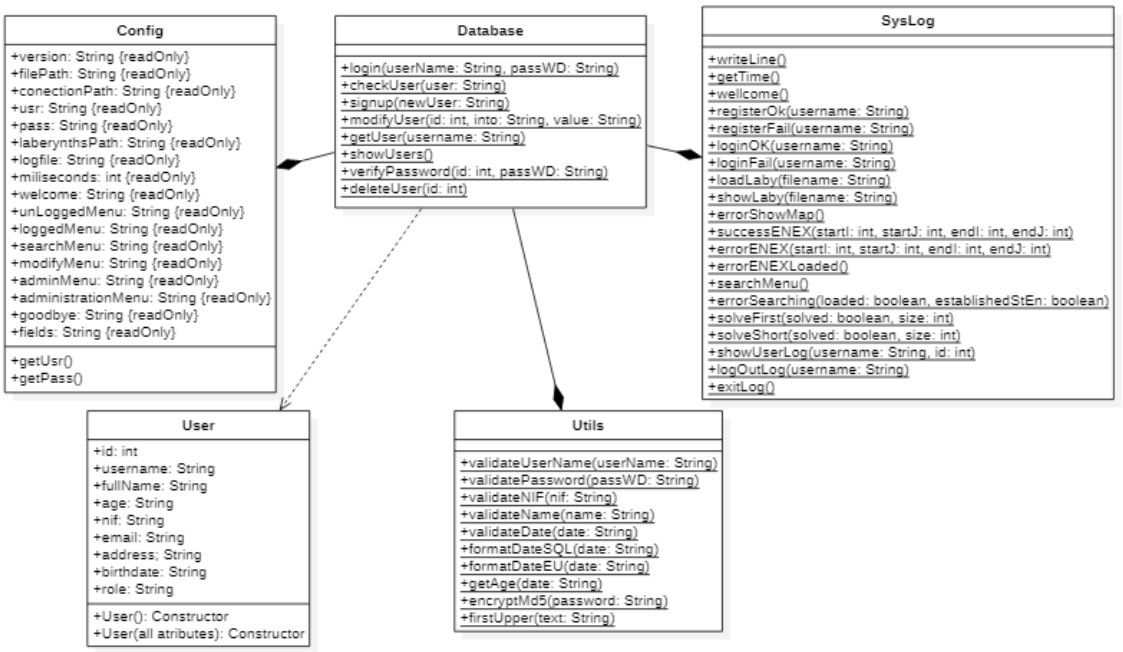
COORDINATE



Esta clase es necesaria para el algoritmo de resolución de laberintos. Contiene los tres atributos necesarios usados por el algortmo:

- i: Atributo público de tipo entero que simboliza el valor de una coordenada I.
- j: Atributo público de tipo entero que simboliza el valor de una coordenada J.
- direction: Atributo público de tipo entero que simboliza una dirección escogida por el algoritmo de resolución de laberintos.

DATABASE



Esta clase contiene los métodos para leer, escribir, eliminar y comparar valores sobre datos existentes en la base de datos.

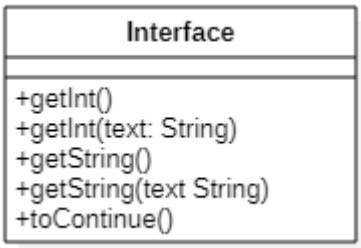
Para llevar a cabo todas las funciones se va a apoyar sobre los métodos y atributos de las clases Config, User y Utils, haciendo uso de los métodos de la clase SysLog para registrar todos los eventos del sistema.

Esta clase no contiene atributos.

Métodos:

- Login(String userName, String passWD): Método público estático que verifica los datos recibidos con los datos de la base de datos para determinar si coinciden y devolver todos los atributos del usuario necesarios.
- CheckUser(String user): Metodo público estático comprueba que el valor recibido existe en algún campo de username, nif o email dentro la base de datos .
- Signup(String[] newUser): Método público estático construye la sentencia para añadir los datos de un nuevo usuario en la base de datos y la lanza.
- ModifyUser(int id, String into, String value): Método público estático que recibe una id, un campo y un valor para formar una sentencia SQL con esos tres valores para modificar una tupla dentro de la base de datos y la lanza.
- getUser(String username): Método público estático que recupera todos los datos de un usuario y los devuelve dentro de un objeto de tipo User.
- ShowUsers(): Método público estático que recupera todos los datos de todas las tuplas de la tabla users de la base de datos y las muestra por pantalla.
- verifyPassword(int id, String passWD): Método público estático que recibe un id junto con una contraseña para verificar que la contraseña recibida coincide con la contraseña de la base de datos para ese id.
- deleteUser(int id): Método público estático que borra un usuario con una determinada id de la base de datos.

INTERFACE

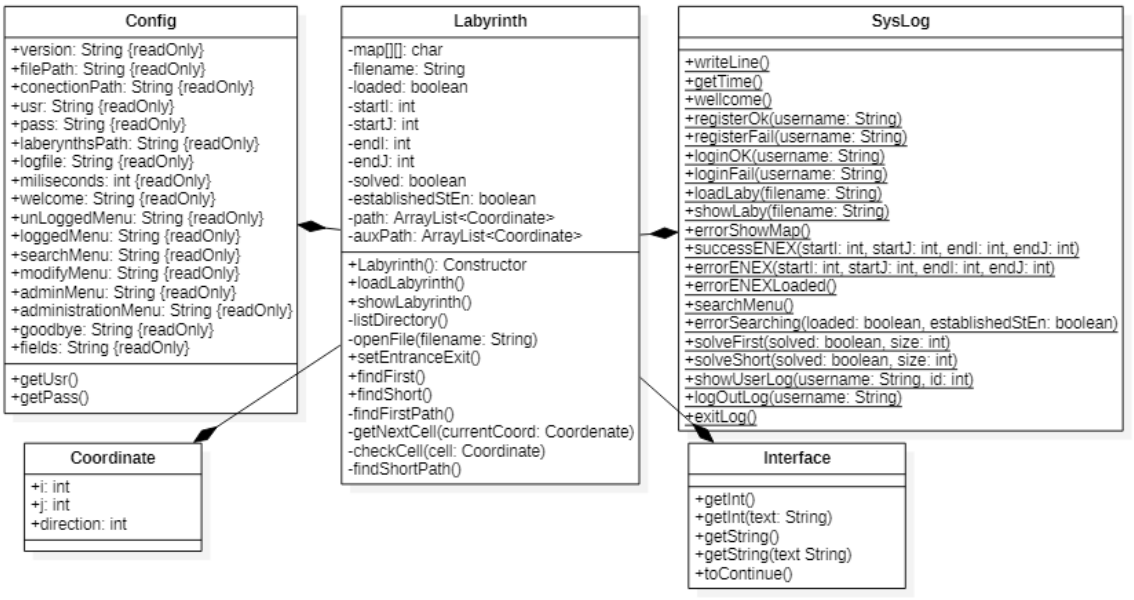


Esta clase es la implementación de la entrada por interfaz de teclado dentro del sistema. Contiene métodos necesarios por otros métodos para la obtención de datos:

- getInt(): Método público estático que no recibe ningún texto de entrada y pide por teclado un número entero para después devolverlo.
- GetiInt(String texto): Método público estático que recibe un texto como parámetro, lo imprime por pantalla para pedir un número entero para después devolverlo.
- GetString(): Método público estático que no recibe ningún texto de entrada y pode por teclado un texto para después devolverlo.
- GetString(String texto): Método público estático que recibe un texto como parámetro, lo imprime por pantalla para pedir un texto para después devolverlo.

- ToContinue(): Método público estático que no recibe ningún texto como parámetro y pide por teclado la inserción de algún dato para continuar con la ejecución del programa.

LABYRINTH



Esta clase es la encargada de gestionar todas las funciones relacionadas con el sistema de resolución de laberintos de nuestro programa.

Se va a apoyar en las clases Config, Interface y Coordinate para cumplir con todas las funciones, así como hacer uso de la clase SysLog para registrar todos los eventos pertinentes producidos en el sistema.

Atributos:

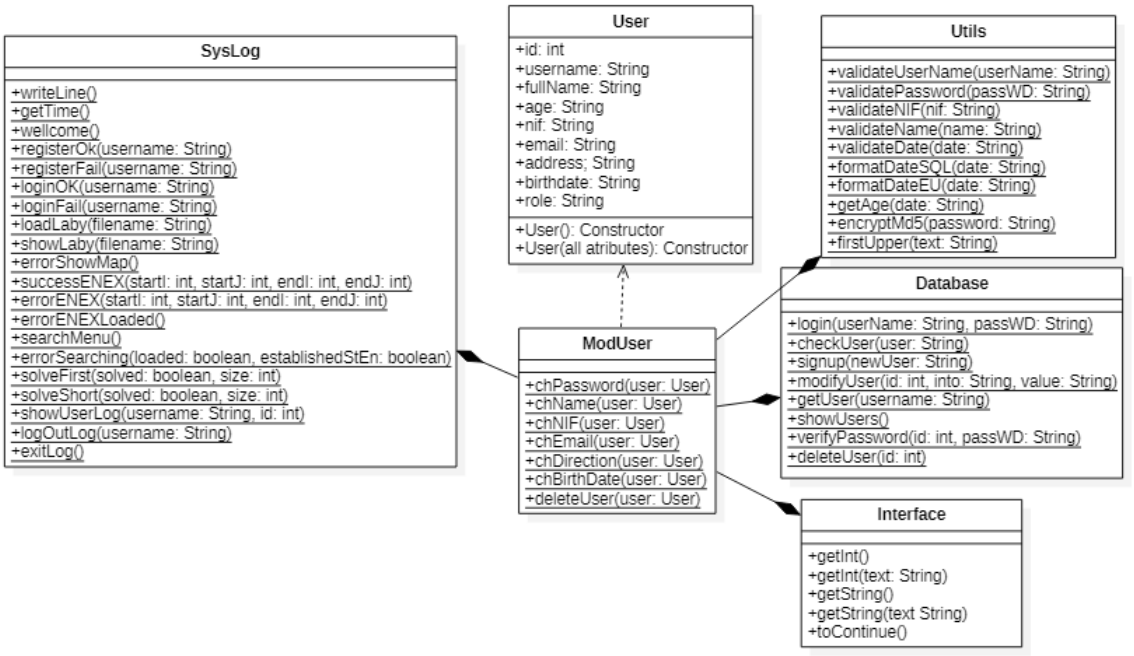
- Map[][]: Atributo privado de tipo char[][] encargado de almacenar el contenido de los laberintos en memoria.
- Filename: Atributo privado de tipo String encargado de almacenar en memoria el nombre del laberinto cargado.
- Loaded: Atributo privado de tipo boolean que contiene el valor booleano de si hay algún laberinto cargado en memoria (true) o no (false).
- StartI, startJ, endI, endJ: Atributos privados de tipo entero que almacenan la información en memoria de los valores de las coordenadas de entrada y salida para un laberinto cargado.
- solved: Atributo privado de tipo boolean que contiene el valor booleano de si se ha logrado la solución al laberinto cargado en memoria.
- establishedStEn: Atributo privado de tipo boolean que contiene el valor booleano de si se ha establecido casillas de entrada y salida para un laberinto cargado en memoria.
- Path: Atributo de tipo ArrayList<Coordinate> encargado de almacenar todos los pasos realizados a la hora de resolver un laberinto.

- AuxPath: Atributo de tipo ArrayList<Coordinate> encargado de almacenar todos los pasos realizados a la hora de resolver un laberinto. Es utilizado para el camino más corto.

Métodos:

- LoadLabyrinth(): Método público que muestra todos los laberintos disponibles en el sistema y carga en memoria el que el usuario escoja.
- ShowLabyrinth(): Método público que muestra el laberinto cargado en memoria así como las casillas de entrada y salida o incluso el dibujo de la solución dentro del laberinto.
- ListDirectory(): Método privado que guarda en memoria el nombre de todos los ficheros donde se encuentra la información de los laberintos.
- OpenFile(String filename): Método privado que guarda la información en bruto de un laberinto en una variable de tipo String y lo devuelve.
- SetEntranceExit():Método público que pide las coordenadas de entrada y salida para el laberinto y las establece en caso de que cumpla con las condiciones válidas.
- FindFirst(): Método que contiene las comprobaciones previas y si se dan las comprobaciones pertinentes llama al método findFirsPath().
- FindShort(): Método público que contiene las comprobaciones previas y si se dan las comprobaciones pertinentes llama al método findFirsPath().
- FindFirstPath(): Método privado que ejecuta el algoritmo de resolución de laberintos para el primer camino posible.
- GetNextCell(Coordinate currentCoord): Método privado que determina la siguiente casilla dentro del algoritmo de resolución de laberintos y devuelve el pertinente objeto de tipo Coordinate.
- CheckCell(Coordinate cell): Método privado que comprueba si la casilla es válida o si ya se encuentra en el path.
- FindShortPath(): Método privado que que ejecuta el algoritmo de resolución de laberintos para el camino más corto.

MODUSER



Esta clase contiene los métodos necesarios para llevar a cabo las funciones para que un usuario con rol “user” pueda modificar sus datos de usuario que permite el sistema.

Se va a apoyar en los atributos y métodos de las clases Database, Utils, Interface y User para llevar a cabo todas las funciones pertinentes, haciendo uso de los métodos de la clase SysLog para registrar todos los eventos del sistema.

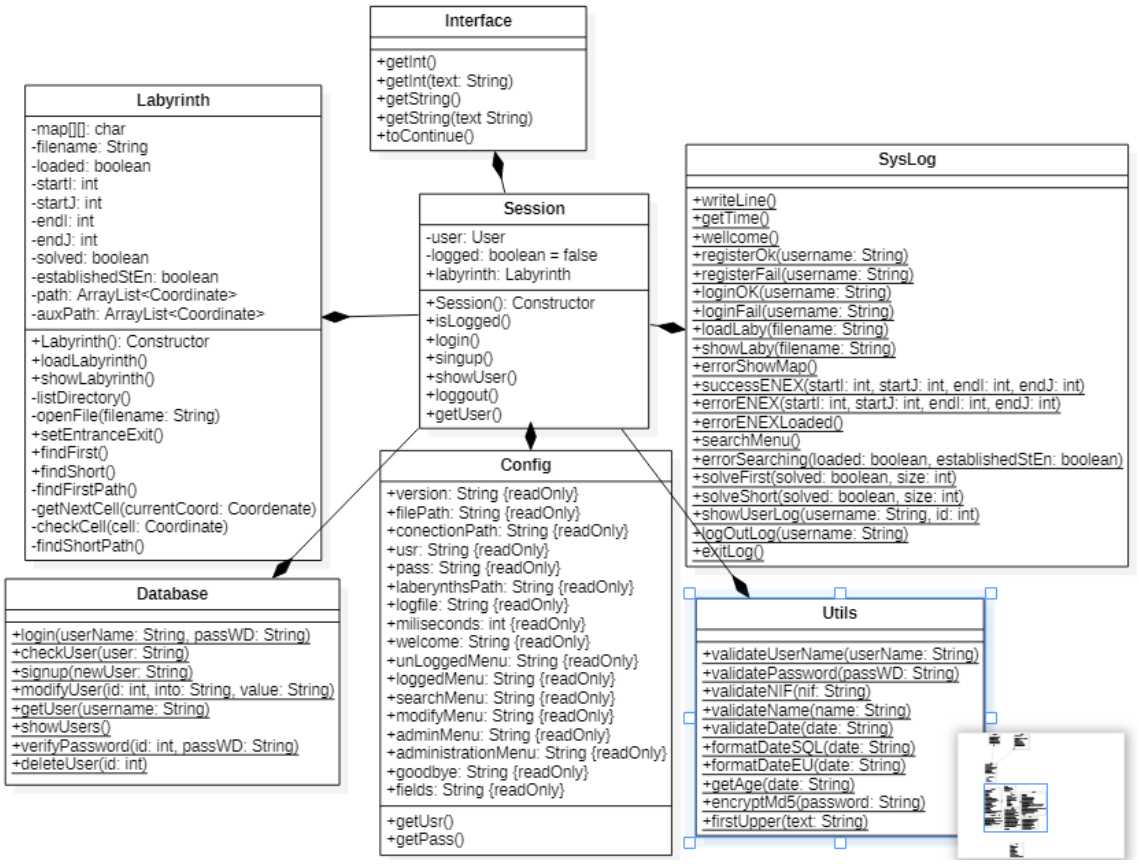
Esta clase no contiene atributos, solamente métodos.

Métodos:

- chPassword(User user): Método público estático que pide la inserción de una nueva contraseña para el usuario autenticado con su repetición para determinar si se escribe en la base de datos.
- ChName(User user): Método público estático que pide la inserción de un nuevo nombre completo para el usuario autenticado para determinar si se escribe en la base de datos.
- ChNIF(User user): Método público estático que pide la inserción de un nuevo NIF para el usuario autenticado para determinar si se escribe en la base de datos.
- ChEmail(User User): Método público estático que pide la inserción de un nuevo email para el usuario autenticado para determinar si se escribe en la base de datos.
- chDirection(User user): Método público estático que pide la inserción de una nueva direccion para el usuario autenticado para escribirlo en la base de datos.
- chBirthDate(User user): Método público estático que pide la inserción de una nueva fecha de nacimiento para el usuario autenticado para determinar si se escribe en la base de datos.

- deleteUser(User user, int myuser): Método público estático que pide la inserción de una nuevo nombre completo para el usuario autenticado para determinar si se escribe en la base de datos.

SESSION



Esta clase contiene los métodos necesarios para llevar a cabo las funciones principales del sistema de usuarios.

Se va a apoyar en los atributos y métodos de las clases Database, Utils, Interface, Labyrinth, Config y User para llevar a cabo todas las funciones pertinentes, haciendo uso de los métodos de la clase SysLog para registrar todos los eventos del sistema.

Atributos:

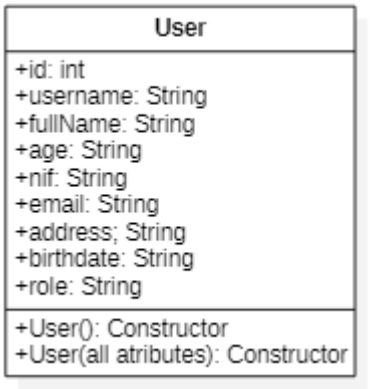
- user: Objeto privado de Tipo User, este objeto es el encargado de almacenar en memoria todos los atributos de un usuario.
- Logged: Atributo privado de tipo booleano que determina si un usuario está autenticado en el sistema (true), o no lo está (false).
- Labyrinth: Objeto de tipo Labyrinth.

Métodos:

- Session(): Constructor de la clase que establece el valor de logged a false.

- IsLogged(): Método público get de la propiedad booleana logged, devuelve el valor de logged.
- Login(): Método público que pide la inserción de usuario y contraseña por parte del usuario y con ayuda del método login de Database determina si son válidos o no para establecer logged a true
- Signup(): Método público que pide por teclado al usuario los datos pertinentes para completar un registro, comprobando cada uno de los datos introducidos para determinar que cumplen con los requisitos del sistema para cada el sistema de usuarios y luego llama al método signup de Database para escribirlo en la base de datos en caso de que todo haya ido bien.
- ShowUser(): Método público que recupera todos los datos de usuario del usuario autenticado en el sistema y los muestra por pantalla incluyendo la edad del mismo, dato que calculará mediante el uso del método getAge(user.birthdate) de la clase Utils.
- Logout(): Método público que establece loaded a false y limpia la información del usuario previamente autenticado de memoria.
- GetUser(): Método público que devuelve el objeto de tipo User declarado en la clase.

USER



Clase que contiene los atributos principales del sistema de usuarios. No contiene el atributo password ni tampoco la edad, que es un valor calculado. Es el pilar fundamental para el sistema de usuarios.

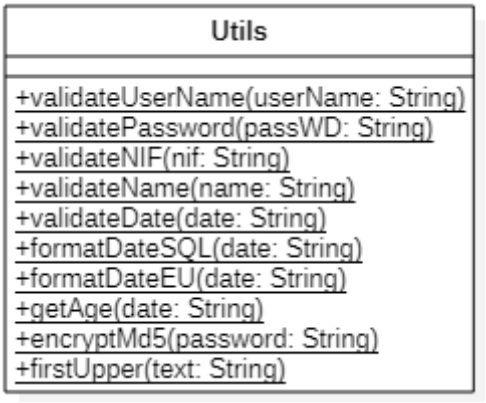
Atributos:

- Id: Atributo público de tipo entero encargado de almacenar el campo “id”.
- Username: Atributo público de tipo String encargado de almacenar el campo “username”.
- FullName: Atributo público de tipo String encargado de almacenar el campo “name”.
- Nif: Atributo público de tipo String encargado de almacenar el campo “nif”.
- Email: Atributo público de tipo String encargado de almacenar el campo “email”.

- Atributo público de tipo String encargado de almacenar el campo “address”.
- Birthdate: Atributo público de tipo String encargado de almacenar el campo “birthdate”.
- Role: Atributo público de tipo String encargado de almacenar el campo “role” (rol de usuario).

Esta clase tiene dos constructores, uno vacío para limpiar el objeto de tipo usuario si se necesita limpiar y otro que recibe los valores para todos los atributos de la clase y establece el valor pertinente a cada atributo.

UTILS



Esta clase contiene los métodos necesarios para verificar la validez de los datos a la hora de gestionar el sistema de usuarios así como evitar el manejo de posibles datos redundantes en el sistema dada la existencia de dichos datos en la base de datos de manera previa.

Métodos:

- ValidateUserName(String userName): Método público estático de tipo boolean que recibe una variable de tipo String que representa el campo username. Este método devuelve el resultado booleano obtenido por la propiedad .matches sobre la variable recibida mediante el uso de un regex.
- ValidatePassword(String passWD): Método público estático de tipo boolean que recibe una variable de tipo String que representa el campo password. Este método devuelve el resultado booleano obtenido por la propiedad .matches sobre la variable recibida mediante el uso de un regex.
- ValidateNif(String nif): Método público estático de tipo boolean que recibe una variable de tipo String que representa el campo nif. Este método devuelve el resultado booleano obtenido por la propiedad .matches sobre la variable recibida mediante el uso de un regex.
- ValidateName(String name): Método público estático de tipo boolean que recibe una variable de tipo String que representa el campo name en la base de datos o fullname dentro de la clase Users. Este método devuelve el resultado booleano obtenido por

la propiedad .matches sobre la variable recibida mediante el uso de un regex.

- ValidateEmail(String email): Método público estático de tipo boolean que recibe una variable de tipo String que representa el campo email. Este método devuelve el resultado booleano obtenido por la propiedad .matches sobre la variable recibida mediante el uso de un regex.
- ValidateDate(String date): Método público estático de tipo boolean que recibe una variable de tipo String que representa el campo date. Este método va comprobando situaciones por las que se puede dar que la fecha de nacimiento no sea válida descartando así todas las posibilidades por las que una fecha puede ser no válida así como determinar las situaciones concretas por las que la fecha puede ser válida. Dependiendo de la comprobación el método puede devolver el valor booleano true si es válida o false si no lo es.
- formatDateSQL(String date): Método público estático de tipo String que recibe una fecha (String date) y comprueba mediante regex si se encuentra en formato Europeo, si es así con ayuda de vectores convierte la fecha de formato europeo (dd/MM/aaaa) a formato SQL (aaaa/MM/dd) para devolverlo. Si no se ha determinado un formato Europeo el sistema comprueba si la fecha introducida ya se encontraba en formato SQL, en caso de que ya estuviera en formato SQL devuelve el valor recibido. Si no se da ninguno de los escenarios la fecha no se encontraba con un formato válido, por lo que devuelve un valor null.
- FormatDateEU(String dateo):): Método público estático de tipo String que recibe una fecha (String date) y comprueba mediante regex si se encuentra en formato SQL, si es así con ayuda de vectores convierte la fecha de formato SQL (aaaa/MM/dd) a formato europeo (dd/MM/aaaa) para devolverlo. Si no se ha determinado un formato SQL el sistema comprueba si la fecha introducida ya se encontraba en formato europeo, en caso de que ya estuviera en formato europeo devuelve el valor recibido. Si no se da ninguno de los escenarios la fecha no se encontraba con un formato válido, por lo que devuelve un valor null.
- GetAge(String date): Método público estático de tipo String que recibe una fecha (String date) y con ayuda de un vector y las funciónes “Period.between”, “LocalDate.of()”, LocalDate.now() y “getYears()” determina la edad a partir de la fecha recibida por el método y devuelve una variable de tipo String que contiene la cantidad de años más un espacio y la palabra años.
- EncryptMD5(String password): Método público estático que recibe una variable de tipo String que hace referencia a una contraseña, la encripta usando un algoritmo MD5 mediante el uso de librerías y métodos y devuelve la contraseña encriptada.
- FirstUpper(String text): Método público estático de tipo String que recibe un texto como parámetro y mediante el uso de las funciones “.substring()”, “.lenght()”, “.toUpperCase()”, y “.toLowerCase()” devuelve el texto recibido con la primera letra en

mayúscula.

DESARROLLOS DE LA IMPLEMENTACION DE CADA FUNCIONALIDAD DEL PROGRAMA

MAIN

Esta clase tiene dos atributos declarados: un atributo laby de tipo Labyrinth y un atributo session de la clase Session.

Los métodos de esta clase son:

- unLoggedAction(): Método público que recibe como parámetro una variable de tipo entero y contiene las acciones del menú principal para un usuario no autenticado.
- loggedAction() Método público que recibe como parámetro una variable de tipo entero y contiene las acciones del menú principal para un usuario autenticado con rol “user”.
- SearchOption(): Método público que recibe como parámetro una variable de tipo entero y contiene las acciones del menú para la resolución de un laberinto.
- ModifyOption(): Método público que recibe como parámetro una variable de tipo entero y contiene las acciones del menú para la modificación de un usuario con rol “user”.
- AdminAction(): Método público que recibe como parámetro una variable de tipo entero y contiene las acciones del menú principal para un usuario autenticado con rol “admin”.
- AdministrationOption(): Método público que recibe como parámetro una variable de tipo entero y contiene las acciones del menú para la gestión de usuarios desde un usuario autenticado con rol “admin”.

ADMIN

Esta clase contiene un atributo de tipo booleano cuya función es determinar si el usuario actualmente autenticado tiene como rol “admin”.

Métodos:

- IsAdmin(): Método público get del atributo “admin” que devuelve el valor de la variable “admin”.
- Admin(User user): Constructor de la clase que recibe como parámetro un objeto de tipo User que representa el usuario actualmente autenticado en el sistema. Comprueba si el contenido del atributo “role” que contiene el objeto coincide con “admin” y establece el valor del atributo “admin” de la clase a true en caso afirmativo o a false en caso negativo.
- selectUser(): Método público estático que no recibe ningún parámetro. El método pide al usuario la inserción de un nombre de

usuario y devuelve un objeto de tipo User procedente del resultado de la llamada al método “Database.*getUser*(username)”.

- chPassword(User user): Método público estático que recibe un objeto de tipo User y comprueba que la propiedad username dentro del objeto user no esté establecido a un valor null. En caso afirmativo devuelve un mensaje y vuelve a “adminMenu”. Si no se da esa situación el método pedirá la entrada por teclado de una nueva contraseña por parte del usuario. En caso de que el usuario deje la inserción en blanco el sistema mostrará un mensaje por pantalla y retrocederá a “adminMenu”. En el caso de que haya introducido una contraseña, el sistema comprobará que la nueva contraseña cumple con los requisitos del sistema. Si no cumple con los requisitos mostrará un aviso por pantalla y regresará a “adminMenu”. En caso contrario el sistema pedirá que el usuario introduzca de nuevo la contraseña y comprobará ambas contraseñas introducidas. En caso de que las contraseñas coincidan el sistema encriptará la contraseña nueva y sustituirá el valor de la contraseña antigua de la base de datos por la nueva para el usuario pertinente. En caso de que las contraseñas no coincidan el sistema mostrará un aviso por pantalla y retrocederá a “adminMenu”.
- ChName(User user): Método público estático que recibe un objeto de tipo User y comprueba que la propiedad username dentro del objeto user no esté establecido a un valor null. En caso afirmativo devuelve un mensaje y vuelve a “adminMenu”. Si no se da esa situación el método pedirá la entrada por teclado de un nuevo valor para el nombre. En caso de que el usuario deje la inserción en blanco el sistema mostrará un mensaje por pantalla y retrocederá a “adminMenu”. En caso de que el usuario haya introducido un nuevo valor, el sistema comprobará que el nuevo valor cumple con los requisitos del sistema para que sea válido. En caso de que no sea válido el sistema mostrará un mensaje por pantalla y retrocederá a “adminMenu”. En caso de que sea válido escribirá en la base de datos el nuevo valor para el usuario pertinente.
- ChNIF(User user): Método público estático que recibe un objeto de tipo User y comprueba que la propiedad username dentro del objeto user no esté establecido a un valor null. En caso afirmativo devuelve un mensaje y vuelve a “adminMenu”. Si no se da esa situación el método pedirá la entrada por teclado de un nuevo valor para el NIF. En caso de que el usuario deje la inserción en blanco el sistema mostrará un mensaje por pantalla y retrocederá a “adminMenu”. En caso de que el usuario haya introducido un nuevo valor, el sistema comprobará si el nuevo valor cumple con los requisitos del sistema para que sea válido, así como que no exista el nuevo valor introducido por el usuario dentro del campo pertinente en la base de datos. Si no cumple con alguno de estos requisitos, el sistema mostrará el mensaje pertinente por pantalla y retrocederá a “adminMenu”. En caso de que cumpla con ambos

requisitos escribirá en la base de datos el nuevo valor para el usuario pertinente.

- ChEmail(User User): Método público estático que recibe un objeto de tipo User y comprueba que la propiedad username dentro del objeto user no esté establecido a un valor null. En caso afirmativo devuelve un mensaje y vuelve a “adminMenu”. Si no se da esa situación el método pedirá la entrada por teclado de un nuevo valor para el email. En caso de que el usuario deje la inserción en blanco el sistema mostrará un mensaje por pantalla y retrocederá a “adminMenu”. En caso de que el usuario haya introducido un nuevo valor, el sistema comprobará si el nuevo valor cumple con los requisitos del sistema para que sea válido, así como que no exista el nuevo valor introducido por el usuario dentro del campo pertinente en la base de datos. Si no cumple con alguno de estos requisitos, el sistema mostrará el mensaje pertinente por pantalla y retrocederá a “adminMenu”. En caso de que cumpla con ambos requisitos escribirá en la base de datos el nuevo valor para el usuario pertinente.
- chDirection(User user): Método público estático que recibe un objeto de tipo User y comprueba que la propiedad username dentro del objeto user no esté establecido a un valor null. En caso afirmativo devuelve un mensaje y vuelve a “adminMenu”. Si no se da esa situación el método pedirá la entrada por teclado de un nuevo valor para el email. En caso de que el usuario deje la inserción en blanco el sistema mostrará un mensaje por pantalla y retrocederá a “adminMenu”. En caso de que el usuario haya introducido un nuevo valor, el sistema, sin realizar comprobación alguna, escribirá en la base de datos el nuevo valor para el usuario pertinente.
- chBirthDate(User user): Método público estático que recibe un objeto de tipo User y comprueba que la propiedad username dentro del objeto user no esté establecido a un valor null. En caso afirmativo devuelve un mensaje y vuelve a “adminMenu”. Si no se da esa situación el método pedirá la entrada por teclado de un nuevo valor para la fecha de nacimiento. En caso de que el usuario deje la inserción en blanco el sistema mostrará un mensaje por pantalla y retrocederá a “adminMenu”. En caso de que el usuario haya introducido un nuevo valor, el sistema comprobará si el nuevo valor cumple con los requisitos del sistema para que sea válido. Si no cumple con los requisitos, el sistema mostrará el mensaje pertinente por pantalla y retrocederá a “adminMenu”. En caso de que cumpla con los requisitos escribirá en la base de datos el nuevo valor para el usuario pertinente.
- deleteUser(User user, int myuser): Método público estático que recibe un objeto de tipo User y comprueba que la propiedad username dentro del objeto user no esté establecido a un valor null. En caso afirmativo devuelve un mensaje junto con un valor booleano false y vuelve a “adminMenu”. Si no se da esa situación el método mostrará un aviso de la acción que se quiere realizar preguntará al usuario si desea continuar con el proceso. En caso

negativo, el método devolverá un valor booleano false y retrocederá a “adminMenu”. En caso positivo el sistema pedirá la inserción de la contraseña del usuario admin y la comparará con la contraseña guardada dentro de la base de datos. Si ambas contraseñas no coinciden el sistema devolverá un valor booleano false y retrocederá a “adminMenu”. En caso de que coincidan el sistema vuelve a mostrar el aviso sobre la acción que se va a realizar y pregunta al usuario si desea continuar. En caso negativo, el sistema devolverá un valor falso y retrocederá a “adminMenu”. En caso de que el usuario desee continuar, el sistema procederá a la eliminación de toda la información que haya en la base de datos para el usuario pertinente. Si el proceso se ha realizado correctamente, el método devolverá un valor booleano true, en caso contrario devolverá false.

CONFIG

Atributos:

- versión: Atributo público de tipo String de solo lectura que contiene la versión actual del programa (1.4.0 en este caso).
- filePath: Atributo público de tipo String de solo lectura que contiene la información de la ruta donde se ubican los archivos del sistema dentro del SO.
- ConnectionPath: Atributo público de tipo String de solo lectura que contiene la sentencia base de conexión para el SGBD.
- Usr: Atributo privado de tipo String de solo lectura que contiene el nombre de usuario para la conexión con el SGBD.
- Pass: Atributo privado de tipo String de solo lectura que contiene la contraseña usada para la conexión con el SGBD.
- LabyrinthsPath: Atributo público de tipo String de solo lectura que contiene la ruta donde se ubican los ficheros que contienen los laberintos dentro del sistema de ficheros del SO.
- Logfile: Atributo público de tipo String de solo lectura que contiene el nombre del fichero de log del sistema.
- Miliseconds: Atributo público de tipo int de solo lectura que contiene la cantidad en mili segundos que el sistema ha de esperar para la transición para alguno de los menús.
- Welcome: Atributo público de tipo String de solo lectura que contiene un mensaje de bienvenida.
- UnLoggedMenu: Atributo público de tipo String de solo lectura que contiene el texto para las opciones disponibles para el menú unloggedMenu.
- LoggedMenu: Atributo público de tipo String de solo lectura que contiene el texto para las opciones disponibles para el menú LoggedMenu.
- SearchMenu: Atributo público de tipo String de solo lectura que contiene el texto para las opciones disponibles para el menú de buscar caminos.

- ModifyMenu: Atributo público de tipo String de solo lectura que contiene el texto para las opciones disponibles para el menú de modificar usuario.
- AdminMenu: Atributo público de tipo String de solo lectura que contiene el texto para las opciones disponibles para el menú adminMenu.
- AdministrationMenu: Atributo público de tipo String de solo lectura que contiene el texto para las opciones disponibles para el menú Gestionar Cuentas.
- Goodbye: Atributo público de tipo String de solo lectura que contiene un mensaje de despedida.
- fields: Atributo público de tipo String[] de solo lectura que contiene el texto usado en el registro para cada uno de los campos.

Esta clase solamente contiene dos métodos GET para los atributos “usr” y “pass”, ya que son los dos únicos atributos privados.

COORDINATE

Atributos:

- i: Atributo público de tipo entero que simboliza el valor de una coordenada I.
- j: Atributo público de tipo entero que simboliza el valor de una coordenada J.
- direction: Atributo público de tipo entero que simboliza una dirección escogida por el algoritmo de resolución de laberintos.

DATABASE

Métodos:

- Login(String userName, String passWD): Método público estático que recibe dos variables de tipo String que representan un nombre de usuario y una contraseña y contiene las acciones a seguir para obtener dichos datos de la base de datos, compararlos con los obtenidos en la declaración del método y si coinciden devolver un objeto de tipo User.
- CheckUser(String user): Metodo público estático que recibe una variable de tipo String. Este método realiza una consulta sobre la base de datos y comprueba que el valor recibido en la variable declarada no coincida con los campos “username”, “nif” o “email” de dicha consulta, devolviendo un valor true si coinciden o false si no coinciden.
- Signup(String[] newUser): Método público estático que recibe una variable de tipo String[] que contiene los datos necesarios para realizar una inserción en la tabla users de la base de datos. El método genera la sentencia SQL pertinente usando los datos que contiene el vector recibido en la declaración del método y la lanza contra la base de datos.
- ModifyUser(int id, String into, String value): Método público estático que recibe una variable de tipo int y dos variables de tipo String que contienen la información pertinente para generar el

código SQL y modificar un dato de un campo en concreto de una tupla indicada. El método devuelve true en caso de que se haya modificado con éxito o false en caso de que haya habido algún tipo de fallo o no hubiera sido posible la modificación.

- getUser(String username): Método público estático que recibe una variable de tipo String que contiene el valor de un campo username necesario para generar la consulta SQL que reciba toda la información de dicho usuario para almacenarla en memoria. Si el proceso ha sido un éxito, el método devuelve true, en caso contrario devolverá false.
- ShowUsers(): Método público estático que no recibe ningún dato. Este método recupera todos los datos de todas las tuplas de la tabla users de la base de datos y las muestra por pantalla.
- verifyPassword(int id, String passWD): Método público estático que recibe una variable de tipo int y otra de tipo String que representan el id de un usuario y una contraseña en concreto. El método recupera los campos de id y contraseña de la base de datos para el id que recibe como parámetro y, tras ello, compara que la contraseña recibida como parámetro coincida con la obtenida de la base de datos. En caso afirmativo devolverá true, en el caso contrario devolverá false.
- deleteUser(int id): Método público estático que recibe una variable de tipo int que representa un id de usuario. El método elimina la tupla completa de la base de datos para dicho id. En caso de éxito devuelve true, en el caso contrario devuelve false.

INTERFACE

Métodos:

- getInt(): Método público estático que no recibe ningún texto de entrada y pide por teclado un número entero para después devolverlo.
- GetiInt(String texto): Método público estático que recibe un texto como parámetro, lo imprime por pantalla para pedir un número entero para después devolverlo.
- GetString(): Método público estático que no recibe ningún texto de entrada y pode por teclado un texto para después devolverlo.
- GetString(String texto): Método público estático que recibe un texto como parámetro, lo imprime por pantalla para pedir un texto para después devolverlo.
- ToContinue(): Método público estático que no recibe ningún texto como parámetro y pide por teclado la inserción de algún dato para continuar con la ejecución del programa.

LABYRINTH

Atributos:

- Map[][]: Atributo privado de tipo char[][] encargado de almacenar el contenido de los laberintos en memoria.

- Filename: Atributo privado de tipo String encargado de almacenar en memoria el nombre del laberinto cargado.
- Loaded: Atributo privado de tipo boolean que contiene el valor booleano de si hay algún laberinto cargado en memoria (true) o no (false).
- StartI, startJ, endI, endJ: Atributos privados de tipo entero que almacenan la información en memoria de los valores de las coordenadas de entrada y salida para un laberinto cargado.
- solved: Atributo privado de tipo boolean que contiene el valor booleano de si se ha logrado la solución al laberinto cargado en memoria.
- establishedStEn: Atributo privado de tipo boolean que contiene el valor booleano de si se ha establecido casillas de entrada y salida para un laberinto cargado en memoria.
- Path: Atributo de tipo ArrayList<Coordinate> encargado de almacenar todos los pasos realizados a la hora de resolver un laberinto.
- AuxPath: Atributo de tipo ArrayList<Coordinate> encargado de almacenar todos los pasos realizados a la hora de resolver un laberinto. Es utilizado para el camino más corto.

Métodos:

- LoadLabyrinth(): Método público que guarda en memoria una lista con todos los laberintos disponibles y resetea el path. Lista los laberintos disponibles y pide al usuario que seleccione un laberinto. Tras seleccionar un laberinto por parte del usuario lo carga en memoria metiéndolo en map[] y establece loaded a true. Si todo ha sido un éxito mostrará un aviso por pantalla. En caso de que en algún momento haya ocurrido algún error muestra el aviso pertinente por pantalla y regresa el menú principal.
- ShowLabyrinth(): Método público que comprueba si existe un mapa cargado en memoria, si existe genera en memoria una versión del mapa mucho más legible para el usuario. Este método tiene en cuenta las casillas de entrada y salida, numeración de coordenadas, y solución del laberinto con los pasos dados en caso de que haya sido resuelto. En caso de que ocurra algún error el sistema mostrará un aviso por pantalla si es pertinente y regresará al menú principal. En caso de que todo vaya bien mostrará por pantalla el laberinto generado en memoria.
- ListDirectory(): Método privado que guarda en memoria l nombre de todos los ficheros donde se encuentra la información de los laberintos.
- OpenFile(String filename): Método privado que recibe como parámetro una variable de tipo String que contiene el nombre de un fichero que contiene la información de un laberinto y lo guarda en memoria en formato de texto (String) y lo devuelve.
- SetEntranceExit():Método público que comprueba si existe un laberinto cargado en memoria y pide por teclado al usuario las coordenadas para las casillas de entrada y salida para dicho laberinto haciendo todas las comprobaciones pertinentes para que

sean casillas válidas. En caso de cumplir con todas las comprobaciones guarda en memoria las casillas, resetea el path, otorga el valor true a establishedExEn y muestra un mensaje de éxito por pantalla. En caso de que alguna comprobación haya sido errónea muestra el mensaje pertinente por pantalla y vuelve al menú principal.

- FindFirst(): Método público que comprueba que exista un laberinto cargado en memoria con las casillas de entrada y salida y en caso afirmativo llama al método findFirstPath() para usar el algoritmo de resolución de laberintos del primer camino posible, guardando el resultado pertinente en la variable solved (true si se ha resuelto, false si no ha sido posible la solución). En caso de que las comprobaciones de laberinto cargado y casillas de entrada y salida establecidas no cumpla con los requisitos mostrará un aviso por pantalla y volverá al menú principal.
- FindShort(): Método público que comprueba que exista un laberinto cargado en memoria con las casillas de entrada y salida y en caso afirmativo llama al método findShortPath() para usar el algoritmo de resolución de laberintos del camino más corto, guardando el resultado pertinente en la variable solved (true si se ha resuelto, false si no ha sido posible la solución). En caso de que las comprobaciones de laberinto cargado y casillas de entrada y salida establecidas no cumpla con los requisitos mostrará un aviso por pantalla y volverá al menú principal.
- FindFirstPath(): Método privado que limpia el path y ejecuta el algoritmo de resolución de laberintos para el primer camino posible, guardando todos los pasos válidos dentro del path apoyándose sobre las propiedades de coordinate. El algoritmo acaba cuando encuentra por primera vez la salida o cuando regresa a la entrada (no teniendo solución posible). Tras determinar el resultado, el método devuelve un valor booleano true si ha resuelto el laberinto o un valor booleano false si no ha encontrado solución posible.
- GetNextCell(Coordinate currentCoord): Método privado que recibe un objeto de tipo Coordinate con la información de la casilla actual junto con la última dirección seguida por el algoritmo de resolución de laberintos. Este método determina un orden establecido a la hora de seleccionar la nueva casilla en función a la nueva dirección en función a la información recibida en el objeto Coordinate obtenido en la declaración del método, devolviendo un objeto de tipo Coordinate con nueva información.
- CheckCell(Coordinate cell): Método privado que recibe un objeto de tipo Coordinate y comprueba si las coordenadas de dicho objeto no coinciden con una pared del laberinto(‘#’). Si coincide con una pared devuelve un valor booleano false. Si no coincide con una pared comprueba si ya ha pasado por esa casilla consultando las casillas seguidas por el path. Si ya ha pasado por esa casilla devolverá un valor booleano false, si no lo ha hecho devolverá un valor booleano true.

- FindShortPath(): Método privado que limpia el path y axuPath. Ejecuta el algoritmo de resolución de laberintos para el camino más corto guardando todos los pasos válidos dentro del path apoyándose sobre las propiedades de coordinate. El algoritmo acaba cuando regresa a la entrada, guardando el contenido de path en axuPath en caso de que sea la primera vez que se encuentra la salida o en caso de que la cantidad de pasos de path sea más pequeña que la de auxPath. Cuando el algorimo termina vuelca todo el auxPath a path para que quede registrado el camino más corto. Tras determinar el resultado, el método devuelve un valor booleano true si ha resuelto el laberinto o un valor booleano false si no ha encontrado solución posible.

MODUSER

Métodos:

- chPassword(User user): Método público estático que recibe un objeto de tipo User con los datos actuales del usuario autenticado en el sistema. El método pide la entrada por teclado de una nueva contraseña por parte del usuario. En caso de que el usuario deje la inserción en blanco el sistema mostrará un mensaje por pantalla y retrocederá a “loggedMenu”. En el caso de que haya introducido una contraseña, el sistema comprobará que la nueva contraseña cumple con los requisitos del sistema. Si no cumple con los requisitos mostrará un aviso por pantalla y regresará a “loggedMenu”. En caso contrario el sistema pedirá que el usuario introduzca de nuevo la contraseña y comprobará ambas contraseñas introducidas. En caso de que las contraseñas coincidan el sistema encriptará la contraseña nueva y sustituirá el valor de la contraseña antigua de la base de datos por la nueva para el usuario pertinente. En caso de que las contraseñas no coincidan el sistema mostrará un aviso por pantalla y retrocederá a “loggedMenu”.
- ChName(User user): Método público estático que recibe un objeto de tipo User con los datos actuales del usuario autenticado en el sistema. El método pide la entrada por teclado por parte del usuario de un nuevo valor para el nombre. En caso de que el usuario deje la inserción en blanco el sistema mostrará un mensaje por pantalla y retrocederá a “loggedMenu”. En caso de que el usuario haya introducido un nuevo valor, el sistema comprobará que el nuevo valor cumple con los requisitos del sistema para que sea válido. En caso de que no sea válido el sistema mostrará un mensaje por pantalla y retrocederá a “loggedMenu”. En caso de que sea válido escribirá en la base de datos el nuevo valor para el usuario pertinente.
- ChNIF(User user): Método público estático que recibe un objeto de tipo User con los datos actuales del usuario autenticado en el sistema. El método pide la entrada por teclado de un nuevo valor para el NIF. En caso de que el usuario deje la inserción en blanco el sistema mostrará un mensaje por pantalla y retrocederá a

"loggedMenu". En caso de que el usuario haya introducido un nuevo valor, el sistema comprobará si el nuevo valor cumple con los requisitos del sistema para que sea válido, así como que no exista el nuevo valor introducido por el usuario dentro del campo pertinente en la base de datos. Si no cumple con alguno de estos requisitos, el sistema mostrará el mensaje pertinente por pantalla y retrocederá a "loggedMenu". En caso de que cumpla con ambos requisitos escribirá en la base de datos el nuevo valor para el usuario pertinente.

- ChEmail(User User): Método público estático que recibe un objeto de tipo User con los datos actuales del usuario autenticado en el sistema. El método pedirá la entrada por teclado de un nuevo valor para el email. En caso de que el usuario deje la inserción en blanco el sistema mostrará un mensaje por pantalla y retrocederá a "loggedMenu". En caso de que el usuario haya introducido un nuevo valor, el sistema comprobará si el nuevo valor cumple con los requisitos del sistema para que sea válido, así como que no exista el nuevo valor introducido por el usuario dentro del campo pertinente en la base de datos. Si no cumple con alguno de estos requisitos, el sistema mostrará el mensaje pertinente por pantalla y retrocederá a "loggedMenu". En caso de que cumpla con ambos requisitos escribirá en la base de datos el nuevo valor para el usuario pertinente.
- chDirection(User user): Método público estático que recibe un objeto de tipo User con los datos actuales del usuario autenticado en el sistema. El método pedirá la entrada por teclado de un nuevo valor para el email. En caso de que el usuario deje la inserción en blanco el sistema mostrará un mensaje por pantalla y retrocederá a "loggedMenu". En caso de que el usuario haya introducido un nuevo valor, el sistema, sin realizar comprobación alguna, escribirá en la base de datos el nuevo valor para el usuario pertinente.
- chBirthDate(User user): Método público estático que recibe un objeto de tipo User con los datos actuales del usuario autenticado en el sistema. El método pedirá la entrada por teclado de un nuevo valor para la fecha de nacimiento. En caso de que el usuario deje la inserción en blanco el sistema mostrará un mensaje por pantalla y retrocederá a "loggedMenu". En caso de que el usuario haya introducido un nuevo valor, el sistema comprobará si el nuevo valor cumple con los requisitos del sistema para que sea válido. Si no cumple con los requisitos, el sistema mostrará el mensaje pertinente por pantalla y retrocederá a "loggedMenu". En caso de que cumpla con los requisitos escribirá en la base de datos el nuevo valor para el usuario pertinente.
- deleteUser(User user, int myuser): Método público estático que recibe un objeto de tipo User con los datos actuales del usuario autenticado en el sistema. El método mostrará un aviso de la acción que se quiere realizar preguntará al usuario si desea continuar con el proceso. En caso negativo, el método retrocederá a "loggedMenu". En caso positivo el sistema pedirá la inserción de

la contraseña del usuario autenticado y la comparará con la contraseña guardada dentro de la base de datos. Si ambas contraseñas no coinciden el sistema retrocederá a “loggedMenu”. En caso de que coincidan el sistema vuelve a mostrar el aviso sobre la acción que se va a realizar y pregunta al usuario si desea continuar. En caso negativo, el sistema retrocederá a “loggedMenu”. En caso de que el usuario desee continuar, el sistema procederá a la eliminación de toda la información que haya en la base de datos para el usuario autenticado, borrará los datos del usuario de memoria y volverá a “unloggedMenu”.

SESSION

Atributos:

- user: Objeto privado de Tipo User, este objeto es el encargado de almacenar en memoria todos los atributos de un usuario.
- Logged: Atributo privado de tipo booleano que determina si un usuario está autenticado en el sistema (true), o no lo está (false).
- Labyrinth: Objeto de tipo Labyrinth.

Métodos:

- Session(): Constructor de la clase que establece el valor de logged a false.
- IsLogged(): Método público get de la propiedad booleana logged, devuelve el valor de logged.
- Login(): Método público que pide la inserción de usuario y contraseña por parte del usuario y usa esos dos atributos para obtener todas las propiedades de user realizando la llamada al método método “Database.login(userName, passWD)”. Tras esto comprueba que los datos obtenidos no son null para establecer logged a true. Tras ello comprueba el valor de logged y si es false mostrará un aviso de Usuario y/o password incorrectos y volverá a “unloggedMenu”.
- Signup(): Método público que declara dos variables con las que va a ir trabajando a la hora de pedir los datos necesarios para el registro por parte del usuario. En total se van a pedir siete datos: nombre de usuario, contraseña, nombre completo, nif, email, dirección y fecha de nacimiento. Para cada uno de los datos que introduzca el usuario, el sistema realizará la comprobación pertinente para determinar si los datos son válidos y cumplen con los requisitos establecidos por el sistema. En caso de que alguna comprobación falle o no cumpla con los requisitos pertinentes, el sistema mostrará un aviso por pantalla y regresará a “unloggedMenu”. En caso de que todos los datos introducidos pasen el control y sean válidos realizará la escritura de los datos sobre la base de datos con todos los datos introducidos y la contraseña encriptada llamando al método sighnup(newUser) de la clase Database pasándole como variable el vector newUser con todos los datos introducidos por el usuario.

- ShowUser(): Método público que recupera todos los datos de usuario del usuario autenticado en el sistema y los muestra por pantalla incluyendo la edad del mismo, dato que calculará mediante el uso del método getAge(user.birthdate) de la clase Utils.
- Logout(): Método público que establece loaded a false y limpia la información del usuario previamente autenticado de memoria.
- GetUser(): Método público que devuelve el objeto de tipo User declarado en la clase.

USER

Atributos:

- Id: Atributo público de tipo entero encargado de almacenar el campo “id”.
- Username: Atributo público de tipo String encargado de almacenar el campo “username”.
- FullName: Atributo público de tipo String encargado de almacenar el campo “name”.
- Nif: Atributo público de tipo String encargado de almacenar el campo “nif”.
- Email: Atributo público de tipo String encargado de almacenar el campo “email”.
- Atributo público de tipo String encargado de almacenar el campo “address”.
- Birthdate: Atributo público de tipo String encargado de almacenar el campo “birthdate”.
- Role: Atributo público de tipo String encargado de almacenar el campo “role” (rol de usuario).

Esta clase tiene dos constructores, uno vacío para limpiar el objeto de tipo usuario si se necesita limpiar y otro que recibe los valores para todos los atributos de la clase y establece el valor pertinente a cada atributo.

UTILS

- ValidateUserName(String userName): Método público estático de tipo boolean que recibe una variable de tipo String que representa el campo username. Este método devuelve el resultado booleano obtenido por la propiedad .matches sobre la variable recibida mediante el uso de un regex.
- ValidatePassword(String passWD): Método público estático de tipo boolean que recibe una variable de tipo String que representa el campo password. Este método devuelve el resultado booleano obtenido por la propiedad .matches sobre la variable recibida mediante el uso de un regex.
- ValidateNif(String nif): Método público estático de tipo boolean que recibe una variable de tipo String que representa el campo nif. Este método devuelve el resultado booleano obtenido por la propiedad .matches sobre la variable recibida mediante el uso de un regex.

- ValidateName(String name): Método público estático de tipo boolean que recibe una variable de tipo String que representa el campo name en la base de datos o fullname dentro de la clase Users. Este método devuelve el resultado booleano obtenido por la propiedad .matches sobre la variable recibida mediante el uso de un regex.
- ValidateEmail(String email): Método público estático de tipo boolean que recibe una variable de tipo String que representa el campo email. Este método devuelve el resultado booleano obtenido por la propiedad .matches sobre la variable recibida mediante el uso de un regex.
- ValidateDate(String date): Método público estático de tipo boolean que recibe una variable de tipo String que representa el campo date. Este método va comprobando situaciones por las que se puede dar que la fecha de nacimiento no sea válida descartando así todas las posibilidades por las que una fecha puede ser no válida así como determinar las situaciones concretas por las que la fecha puede ser válida. Dependiendo de la comprobación el método puede devolver el valor booleano true si es válida o false si no lo es.
- formatDateSQL(String date): Método público estático de tipo String que recibe una fecha (String date) y comprueba mediante regex si se encuentra en formato Europeo, si es así con ayuda de vectores convierte la fecha de formato europeo (dd/MM/aaaa) a formato SQL (aaaa/MM/dd) para devolverlo. Si no se ha determinado un formato Europeo el sistema comprueba si la fecha introducida ya se encontraba en formato SQL, en caso de que ya estuviera en formato SQL devuelve el valor recibido. Si no se da ninguno de los escenarios la fecha no se encontraba con un formato válido, por lo que devuelve un valor null.
- FormatDateEU(String dateo):): Método público estático de tipo String que recibe una fecha (String date) y comprueba mediante regex si se encuentra en formato SQL, si es así con ayuda de vectores convierte la fecha de formato SQL (aaaa/MM/dd) a formato europeo (dd/MM/aaaa) para devolverlo. Si no se ha determinado un formato SQL el sistema comprueba si la fecha introducida ya se encontraba en formato europeo, en caso de que ya estuviera en formato europeo devuelve el valor recibido. Si no se da ninguno de los escenarios la fecha no se encontraba con un formato válido, por lo que devuelve un valor null.
- GetAge(String date): Método público estático de tipo String que recibe una fecha (String date) y con ayuda de un vector y las funciónes “Period.between”, “LocalDate.of()”, LocalDate.now() y “getYears()” determina la edad a partir de la fecha recibida por el método y devuelve una variable de tipo String que contiene la cantidad de años más un espacio y la palabra años.
- EncryptMD5(String password): Método público estático que recibe una variable de tipo String que hace referencia a una contraseña, la encripta usando un algoritmo MD5 mediante el uso de librerías y métodos y devuelve la contraseña encriptada.

- FirstUpper(String text): Método público estático de tipo String que recibe un texto como parámetro y mediante el uso de las funciones “.substring()”, “.lenght()”, “.toUpperCase()”, y “.toLowerCase()” devuelve el texto recibido con la primera letra en mayúscula.

DESARROLLO DE LA IMPLEMENTACION DEL ALGORITMO

Primer camino

El algoritmo comienza con la limpieza del path y la declaración de un objeto Coordinate con los valores de i y j para la casilla de salida y 0 para la dirección(ninguna). Este objeto de tipo Coordinate se agrega como primer paso en el path.

El algoritmo comienza el proceso de resolución mediante un bucle while cuyas condiciones sean que la variable found se encuentre a false y que la longitud del path sea mayor a 0.

Para cada vuelta del bucle el algoritmo lo que hace es coger el último objeto del path y sumarle 1 a dirección.

Mientras que la dirección del último objeto del path sea menor que 5 (solo hay 4 posibles direcciones como ya se verá más adelante) se hará lo siguiente:

Se declara un objeto de tipo Coordinate con nombre nextCell que recibe el valor recibido del método getNextCell, al que se le pasa como parámetro el último objeto de tipo Coordenate que contiene el path.

El método getNextCell(Coordinate currentCoord) declara un objeto de tipo Coordenate llamado nextCoord al que le asigna los valores de I y J de currentCoord respectivamente, pero el valor de dirección lo establece a 0 en la declaración. Tras la declaración del objeto, comprueba la dirección de currentCoord y en función a esa dirección suma o resta un valor a J o a I de nextCoord. Los condicionales LOGICOS que se han seguido para determinar qué decisión tomar son los siguientes:

- Los movimientos horizontales de “J” van de izquierda a derecha, siendo izquierda un menor valor, mientras que derecha un mayor valor.
- Los movimientos de “I” van de arriba hacia abajo, siendo arriba el menor valor mientras que abajo es el mayor valor.
- Direction 1 significa uno a la izquierda.
- Direction 2 significa uno hacia arriba.
- Direction 3 significa uno hacia la derecha.
- Direction 4 significa uno hacia la abajo.

De esta forma, si direction = 1 se resta 1 a “J”. Si direction = 2 se resta 1 a “I”. Si direction = = se suma 1 a “J”. Si direction es 4 se suma 1 a “I”.

Habiendo establecido el nuevo objeto de tipo Coordenate lo devuelve para seguir con el algoritmo esperando la respuesta booleana del método ChechCell() con el objeto de tipo Coordinate nextCell como parámetro.

El método checkCell recibe un objeto de tipo Coordinate llamado cell, el cual comprueba si las propiedades “I” y “J” que contienen son un “#” dentro de map, que contiene los datos del laberinto cargado en memoria. Si es un “#” significa que es una pared y el método devuelve false para que el algoritmo continue.

Si no es un “#” el algoritmo recorre el path comprobando que dicha coordenada no se encuentre ya dentro del path, lo que significaría que ya ha pasado por ahí, devolviendo un valor false.

En caso de que no se dé ninguno de los supuestos el método devuelve true, significando que es un nuevo camino.

Si el método checkCell(nextCell) ha dado como resultado true el objeto nextCell se agrega al path y el algoritmo comprueba si las coordenadas de nextCell concuerda con las coordenadas de salida colocando el valor de found a true, saliendo del bucle y terminando el algoritmo, en caso de que no sea así el bucle vuelve al principio

En caso de que el método checkCell(nextCell) haya dado como resultado false, el algoritmo elimina el último objeto del path y el bucle vuelve al principio.

Camino más corto

El algoritmo comienza con la limpieza del path y la declaración de un objeto Coordinate con los valores de i y j para la casilla de salida y 0 para la dirección(ninguna). Este objeto de tipo Coordinate se agrega como primer paso en el path.

El algoritmo comienza el proceso de resolución mediante un bucle while cuya condición es que la longitud del path sea mayor a 0, de esta forma, sí o sí el bucle recorrerá todos los caminos posibles, no deteniendo el bucle si found se encuentra a true como en el primer camino posible.

Para cada vuelta del bucle el algoritmo lo que hace es coger el último objeto del path y sumarle 1 a dirección.

Mientras que la dirección del último objeto del path sea menor que 5 (solo hay 4 posibles direcciones como ya se verá más adelante) se hará lo siguiente:

Se declara un objeto de tipo Coordinate con nombre nextCell que recibe el valor recibido del método getNextCell, al que se le pasa como parámetro el último objeto de tipo Coordenate que contiene el path.

El método getNextCell(Coordinate currentCoord) declara un objeto de tipo Coordenate llamado nextCoord al que le asigna los valores de I y J de currentCoord respectivamente, pero el valor de dirección lo establece a 0 en la declaración. Tras la declaración del objeto, comprueba la dirección de currentCoord y en función a esa dirección suma o resta un valor a J o a I de

nextCoord. Los condicionales LOGICOS que se han seguido para determinar qué decisión tomar son los siguientes:

- Los movimientos horizontales de “J” van de izquierda a derecha, siendo izquierda un menor valor, mientras que derecha un mayor valor.
- Los movimientos de “I” van de arriba hacia abajo, siendo arriba el menor valor mientras que abajo es el mayor valor.
- Direction 1 significa uno a la izquierda.
- Direction 2 significa uno hacia arriba.
- Direction 3 significa uno hacia la derecha.
- Direction 4 significa uno hacia la abajo.

De esta forma, si direction = 1 se resta 1 a “J”. Si direction = 2 se resta 1 a “I”. Si direction = = se suma 1 a “J”. Si direction es 4 se suma 1 a “I”.

Habiendo establecido el nuevo objeto de tipo Coordenate lo devuelve para seguir con el algoritmo esperando la respuesta booleana del método ChechCell() con el objeto de tipo Coordinate nextCell como parámetro.

El método checkCell recibe un objeto de tipo Coordinate llamado cell, el cual comprueba si las propiedades “I” y “J” que contienen son un “#” dentro de map, que contiene los datos del laberinto cargado en memoria. Si es un “#” significa que es una pared y el método devuelve false para que el algoritmo continue.

Si no es un “#” el algoritmo recorre el path comprobando que dicha coordenada no se encuentre ya dentro del path, lo que significaría que ya ha pasado por ahí, devolviendo un valor false.

En caso de que no se dé ninguno de los supuestos el método devuelve true, significando que es un nuevo camino.

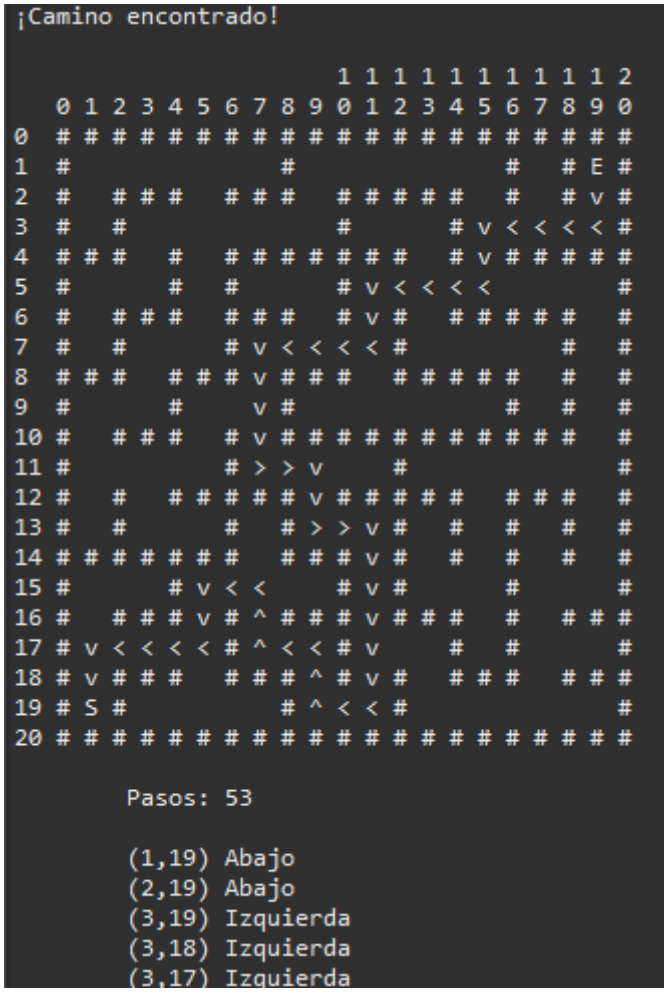
Si el método checkCell(nextCell) ha dado como resultado true el objeto nextCell se agrega al path y el algoritmo comprueba si las coordenadas de nextCell concuerda con las coordenadas de salida colocando el valor de found a true y hace dos comprobaciones:

1. Comprueba que auxPath esté vacío, y si lo está hace una copia del contenido de path sobre auxPath guardando de esta forma la primera vez que encuentra un camino.
2. Comprueba que la longitud de auxPath sea mayor a la de path y si es así limpia auxPath y guarda una copia del contenido de path dentro de auxPath, guardando así un camino más corto a la hora de encontrar la salida.

En cualquier caso, independientemente del resultado el bucle no acabará hasta que path esté vacío.

En caso de que el método checkCell(nextCell) haya dado como resultado false, el algoritmo elimina el último objeto del path y el bucle vuelve al principio.

Una vez ha terminado el bucle se comprueba que auxPath tiene datos. En caso de que tenga datos se guardan todos los datos de auxPath en path.


¡Camino encontrado!
Pasos: 53
(1,19) Abajo
(2,19) Abajo
(3,19) Izquierda
(3,18) Izquierda
(3,17) Izquierda