REPORTE DE ESTADO

## Trabajo Práctico “Laberintos y Fantasmas”

#### Grupo Pixelados

### Cátedra:

Algoritmos y Estructuras de Datos

### Integrantes:

* Ibarra Sánchez, Ludmila Daiana
* Maranzana, Rodrigo Ezequiel
* Ermasi, Franco
* Masino, Carlos Nicolas

Fecha: 07/10/2025

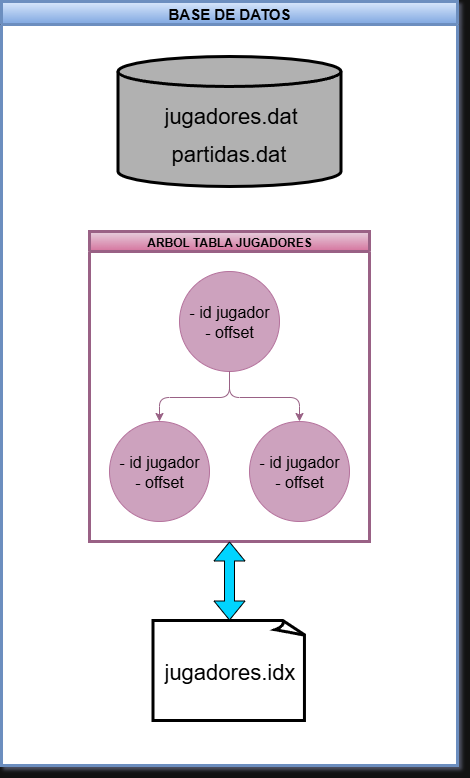
## Funcionalidades actuales del Cliente:

Nuestro grupo tomó la decisión de utilizar la biblioteca gráfica SDL y sus derivadas SDL\_mixer (mezclador de audio), SDL\_ttf (soporte TrueType) y SDL\_image (decodificador de imagenes).

Las siguientes funcionalidades se encuentran completas y funcionando:

* Lectura de configuraciones iniciales desde un archivo de texto.
* Generación del laberinto utilizando el algoritmo DFS (Deep-First Search) y post procesado, para asegurar la jugabilidad y controlar la dificultad.
* Colocación de puerta de entrada y salida, asegurando su conexión con el laberinto.
* Colocación de premios y vidas extra, puestas en lugares estratégicos y su recolección por parte del jugador, junto a su otorgamiento respectivo de punto/vida.
* Utilización de semillas que permiten generar partidas y sus sucesivas rondas exactamente iguales.
* Colocación de hasta la cantidad de fantasmas indicada en el archivo de configuración, variando ronda a ronda entre 1 fantasma y el máximo establecido.
* Cuatro tipos de fantasmas con sus respectivas variaciones de búsqueda del jugador, utilizando intercalación entre el algoritmo DFS (Deep-First Search) y BFS (Breadth-First Search), siendo el último mucho más agresivo aumentando así drásticamente la dificultad del juego. Cada fantasma posee su propio color para su fácil detección.
* Encolado de los movimientos del jugador y los fantasmas, con su posterior función de procesado de movimientos.
* Registro de los movimientos del jugador realizados en cada ronda y su posterior muestra por consola.
* Menú principal con las opciones de Nueva Partida, Continuar, Cambiar Usuario, Estadísticas y Salir (Cambiar Usuario y Estadísticas aún en progreso)
* Ingreso de nombre de Usuario, al iniciar el juego, mediante una ventana con un campo de texto realizado con SDL.
* Servicio de conexión socket tanto en cliente como el servidor.
* Servicio de base de datos en el servidor, que permite la creación de tablas, inserción de registros y selección de los mismos. Cada tabla se guarda en su respectivo archivo .dat e implementa la búsqueda de los registros mediante árboles de búsqueda binaria o escaneo completo de tabla (cuando se busca por un campo que no es clave primaria). Los árboles persisten en sus respectivos archivos índices. El servicio de base de datos acepta comandos de texto similares a los de SQL, y retorna respuestas de texto, y en caso de selección de registros un envío extra en formato binario. El cliente opera con dos tablas, jugadores y partidas.

## Diseño de la Base de Datos del Servidor



## Funcionalidades actuales del Servicio de Base de Datos:

El servicio de base de datos es un sistema que permite la creación de tablas genéricas, según las diferentes solicitudes que se realicen hacia el servidor. Todas las tablas cuentan con un índice implementado con árbol de búsqueda binaria. Este contiene como clave el valor del campo PK. Cuando se solicite una operación que necesite realizar una búsqueda, sobre un campo que no sea clave primaria, el servicio de base de datos realiza un escaneo completo de tabla. El sistema gestiona de manera automática la apertura y cierre de tablas existentes.

La comunicación entre cliente-servidor se lleva a cabo mediante comandos de texto. El servicio de base de datos tiene comandos (CREAR, INSERTAR, ACTUALIZAR, SELECCIONAR), y restricciones (PK, AI) y tipos de datos (ENTERO, TEXTO).

En el caso de SELECCIONAR, si la consulta arroja resultados, los registros se envían (luego de un mensaje de texto convencional) en formato binario con la misma composición y orden que el Cliente haya solicitado al crear la tabla.

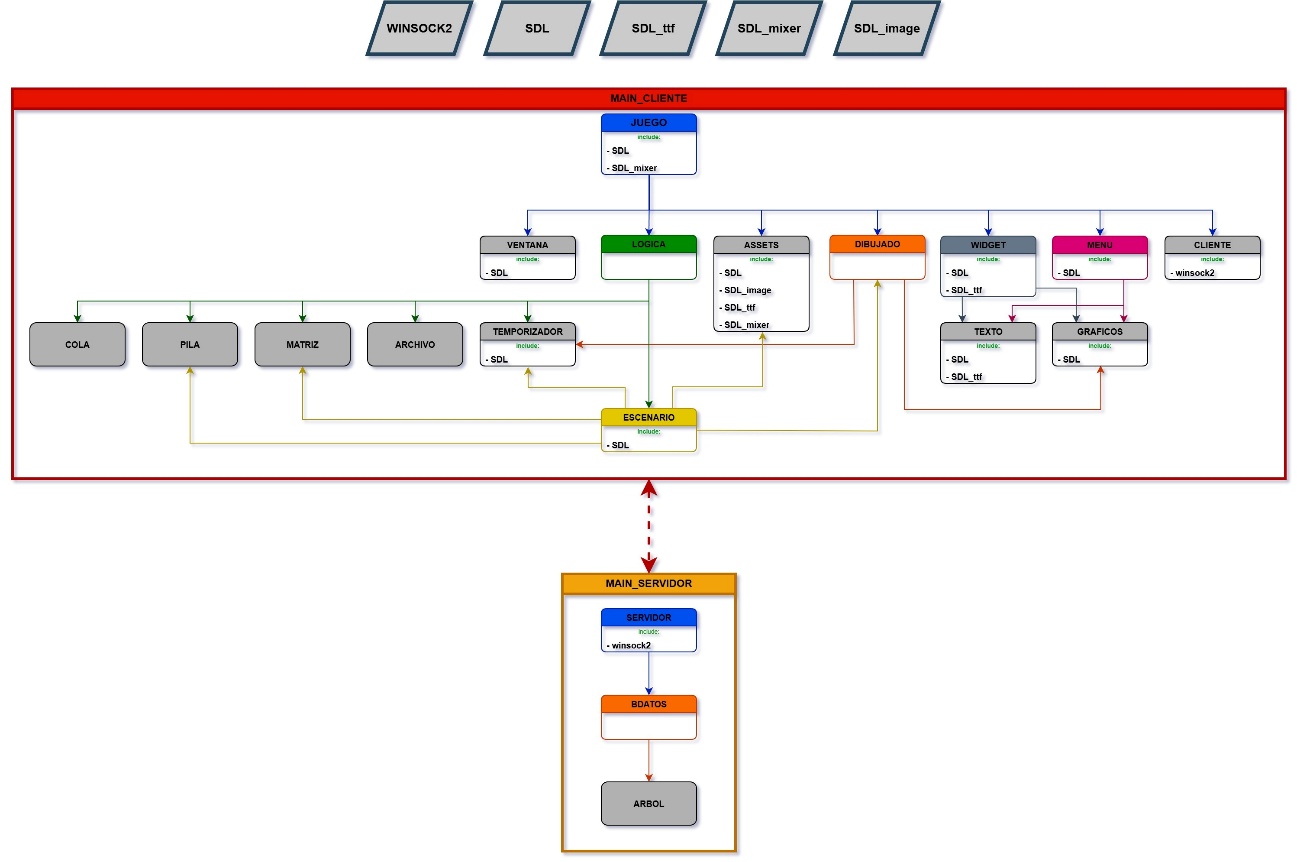
La siguiente lista incluye ejemplos de las solicitudes admitidas por el servidor:

* CREAR:
  + CREAR jugadores (username TEXTO(16) PK, puntajeMax ENTERO, cantPartidas ENTERO)
  + CREAR partidas (idPartida ENTERO PK AI, puntaje ENTERO, cantMovs ENTERO)
* INSERTAR:
  + INSERTAR jugadores (nombre PEPE, puntajeMax 15)
* ACTUALIZAR:
  + ACTUALIZAR jugadores (puntajeMax 25) DONDE idJugador IGUAL 21
* SELECCIONAR:
  + SELECCIONAR jugadores DONDE username IGUAL PEPE
  + SELECCIONAR partidas DONDE username IGUAL PEPE

## Bibliotecas del Cliente:

* **JUEGO**: Controla la creación e inicialización de los componentes de SDL necesarios. Gestiona el game loop y los respectivos llamados a las funciones que conforman tanto a la lógica, detección de inputs, menús, comunicación con el socket, reproducción de efectos de audio y llamado a renderizado.
* **LOGICA**: Controla, como su nombre indica, la lógica que Laberintos y Fantasmas debe seguir. Gestiona los llamados a funciones generadoras de la partida, los movimientos del jugador, de los fantasmas, el otorgamiento de vidas, puntos, mientras regla las interacciones.
* **ASSETS**: Carga desde el disco los archivos de imagen, audio y fuentes TrueType necesarios para la ejecución del juego.
* **DIBUJADO**: Se encarga del Algoritmo del Pintor para la correcta muestra del mundo y las entidades que la componen. Como el juego utiliza perspectiva Top Down, se requiere un dibujado desde la fila menor, columna a columna, hasta la fila mayor, intercalando el dibujado de las entidades para la generación del efecto de 2.5D.
* **ESCENARIO**: Gestiona los algoritmos de generación del laberinto y la inicialización del escenario de juego.
* **CLIENTE**: Gestiona la comunicación con el servidor para la inserción y recupero de los jugadores y sus respectivas partidas utilizando la biblioteca winsock2.
* **VENTANA**: Permite la generación de ventanas graficas de SDL genéricas dentro de la propia ventana del juego.
* **WIDGET**: Permite la creación de utilidades como campos de texto, contadores numéricos gráficos, gráficos de barra, etc. para la muestra y/o recolección de información durante el juego y la partida.
* **MENU**: Permite la creación de menús gráficos interactivos y personalizables.
* **TEXTO**: Gestiona la creación de texturas a partir de strings y fuentes TrueType.
* **GRAFICOS**: Gestiona la escritura de texturas sobre el framebuffer de SDL.
* **TEMPORIZADOR**: Gestiona temporizadores (no son multihilo) que permiten ejecutar acciones de poca precisión como la animación de las entidades y entornos y la espera no bloqueante.
* **ARCHIVO**: Funciones para la lectura, parseo y escritura del archivo de configuración.
* **PILA, COLA, ARBOL**: Funciones de sus respectivos TDA.

# Diagrama de Bloques



Nuestro proyecto se divide en módulos con responsabilidades únicas. Algunos módulos son APIs, otros un compendio de funciones relacionadas, y otros son TDAs vistos en la materia.

Como la cantidad de funciones que hemos desarrollado es significativa, realizar una descripción de cada una haría que este informe sea innecesariamente extenso. Hemos utilizado nombres de funciones autodocumentados y descriptivos por lo que, en conjunto con la descripción de cada módulo, esperamos sea suficientemente explicativo.

DESCRIPCIÓN MODULOS CLIENTE:

### ARCHIVO:

Contiene funciones dedicadas a la manipulación (lectura, parseo, escritura) del archivo de configuración y contingencia.

### ASSETS:

Contiene funciones dedicadas a la carga de assets (recursos) del juego como las imágenes, sonidos y fuentes, así como las funciones de destrucción de los mismos.

### CLIENTE:

Contiene las funciones necesarias para establecer la comunicación con el servidor, desde la creación, conexión y cierre del socket, así como aquellas que emiten solicitudes y recepcionan las respuestas.

### DIBUJADO:

Contiene funciones complementarias que se encargan de la interpretación del escenario del juego y sus entidades para luego renderizarlas según nuestro propio “Algoritmo del Pintor”.

### ESCENARIO:

Contiene las funciones que generan el escenario del juego. Realiza la inicialización de los elementos necesarios para la generación pseudoaleatoria del laberinto, su post procesamiento y la colocación de puertas.

### GRAFICOS:

Contiene las funciones que permiten graficar en el renderer de SDL.

### JUEGO:

Contiene las funciones principales que orquestan y ordenan el juego en sí, similar a un motor de juego. Sus funciones se encargan de determinar desde la inicialización del juego, su continua actualización y recepción de inputs, cambios de estado, así como su posterior destrucción.

### LOGICA:

Contiene las funciones que controlan y dirigen los comportamientos y reglas que “Laberintos y Fantasmas” debe respetar. Se encarga de la inicialización de la lógica, así como el procesamiento de turnos, movimientos del jugador y los fantasmas, inicio de partida, cambio de ronda, y algoritmos de búsqueda por parte de los fantasmas.

### MENU:

Contiene funciones que permiten la creación y manipulación de menús.

### TEMPORIZADOR:

Contiene funciones que permiten la creación, actualización, y manipulación de temporizadores (no multihilos).

### TEXTO:

Contiene funciones que permiten la creación de texturas a partir de fuentes TTF.

### VENTANA:

Contiene funciones que permiten la creación de ventanas contenedoras genéricas, y su posterior manipulación, apertura, y cierre.

### WIDGET:

Contiene funciones que permiten la creación, actualización y manipulación de widgets (utilidades principalmente prácticas para la interfaz de usuario HUD)

FUNCIONES MODULOS CLIENTE:

### ARCHIVO:

##### Funciones Declaradas

* int archivo\_leer\_conf(FILE\* arch, tConf \*conf);
* int archivo\_escribir\_conf(FILE\* arch, const tConf \*conf);
* int archivo\_escribir\_escenario(tEscenario \*escenario, int numRonda, long semillaRonda);

##### Funciones Internas:

* static int \_archivo\_parsear\_linea\_conf(char \*buffer, tParam \*param);

### ASSETS:

##### Funciones Declaradas

##### int assets\_cargar\_imagenes(SDL\_Renderer \*renderer, SDL\_Texture \*\*imagenes);

##### int assets\_cargar\_sonidos(Mix\_Chunk \*\*sonidos);

##### int assets\_cargar\_fuente(TTF\_Font \*\*fuente, int tamFuente);

##### void assets\_destuir\_imagenes(SDL\_Texture \*\*imagenes);

##### void assets\_destuir\_sonidos(Mix\_Chunk \*\*sonidos);

##### void assets\_destruir\_fuente(TTF\_Font \*fuente);

### CLIENTE:

##### Funciones Declaradas

##### int cliente\_inicializar();

##### SOCKET cliente\_conectar\_servidor(const char \*ipServidor, int puerto);

##### int cliente\_enviar\_solicitud(SOCKET sock, const char \*solicitud);

##### void cliente\_cerrar\_conexion(SOCKET sock);

##### int cliente\_recibir\_respuesta(SOCKET sock, char \*respuesta, int tamBuffer);

##### int cliente\_recibir\_datos(SOCKET sock, char \*bufferDatos, int bytesEsperados);

##### int cliente\_ejecutar\_solicitud(SOCKET sock, const char \*solicitud, int\* cantRegistros, char\*\* bufferDatos);

### DIBUJADO:

##### Funciones Declaradas

* void dibujado\_escenario(SDL\_Renderer \*renderer, tEscenario \*escenario, SDL\_Texture \*\*imagenes);

##### Funciones Internas:

* static void \_dibujado\_jugador\_aturdido(SDL\_Texture \*textura);
* static void \_dibujado\_fantasma\_aturdido(SDL\_Texture \*textura);
* static void \_dibujado\_entidad(SDL\_Renderer \*renderer, SDL\_Texture \*\*imagenes, tEntidad \*entidad);
* static void \_dibujado\_jugador\_potenciado(SDL\_Texture \*textura);

### ESCENARIO:

##### Funciones Declaradas

##### int escenario\_crear(tEscenario \*escenario, unsigned columnas, unsigned filas);

##### void escenario\_generar(tEscenario \*escenario);

##### void escenario\_destruir(tEscenario \*escenario);

##### int escenario\_calcular\_mascara(tEscenario \*escenario, int columna, int fila);

##### eParedLimite escenario\_ubic\_es\_pared\_limite(const tEscenario \*escenario, tUbicacion ubic);

##### Funciones Internas:

* static void \_escenario\_init\_tiles(tTile \*tiles);
* static void \_escenario\_generar\_laberinto(tEscenario \*escenario);
* static void \_escenario\_postprocesar(tEscenario \*escenario);
* static void \_escenario\_calcular\_puerta(tCasilla \*puerta, tTile \*tiles, int esEntrada, int pared, int cantFilas, int cantColumnas);
* static void \_escenario\_colocar\_puertas(tEscenario \*escenario);
* static void \_escenario\_despejar\_puerta(tEscenario \*escenario, tUbicacion ubicPuerta, int pared);
* static void \_escenario\_liberar\_casilla(tEscenario \*escenario, int fila, int columna);

### GRAFICOS:

##### Funciones Declaradas

##### SDL\_Texture\* graficos\_crear\_textura(SDL\_Renderer \*renderer, int ancho, int alto, int modoAcceso, unsigned formato);

##### void graficos\_dibujar\_textura(SDL\_Texture \*textura, SDL\_Renderer \*renderer, SDL\_Rect \*rectFuente, SDL\_Rect \*rectDestino, tEfectoGrafico efecto);

##### void graficos\_dibujar\_borde(SDL\_Renderer \*renderer, SDL\_Rect rect, SDL\_Color color);

##### void graficos\_dibujar\_relleno(SDL\_Renderer \*renderer, SDL\_Rect rect, SDL\_Color color);

### JUEGO:

##### Funciones Declaradas

##### int juego\_inicializar(tJuego \*juego, const char \*tituloVentana);

##### int juego\_ejecutar(tJuego \*juego);

##### void juego\_destruir(tJuego \*juego);

##### Funciones Internas:

* static int \_juego\_crear\_ventana(SDL\_Window \*\*ventana, SDL\_Renderer \*\*renderer, unsigned anchoRes, unsigned altoRes, const char \*tituloVentana);
* static void \_juego\_renderizar(SDL\_Renderer \*renderer, SDL\_Texture \*\*imagenes, tLogica \*logica, tVentana \*ventanaMenu, tVentana \*ventanaUsername, tHud \*hud);
* static void \_juego\_iniciar\_partida(void\* datos);
* static void \_juego\_salir\_del\_juego(void\* datos);
* static void \_juego\_continuar\_partida(void\* datos);
* static int \_juego\_actualizar\_hud(tHud \*hud, tLogica \*logica);
* static int \_juego\_crear\_hud(tJuego \*juego);
* static int \_juego\_cargar\_assets(tJuego \*juego);
* static void \_juego\_manejar\_input(tJuego \*juego, SDL\_Keycode tecla);
* static void \_juego\_manejar\_eventos(tJuego \*juego);
* static int \_juego\_ventana\_menu\_crear(void \*datos);
* static void \_juego\_ventana\_menu\_actualizar(SDL\_Event \*evento, void \*datos);
* static void \_juego\_ventana\_menu\_dibujar(void \*datos);
* static void \_juego\_ventana\_menu\_destruir(void \*datos);
* static int \_juego\_ventana\_usuario\_crear(void \*datos);
* static void \_juego\_ventana\_usuario\_actualizar(SDL\_Event \*evento, void \*datos);
* static void \_juego\_ventana\_usuario\_dibujar(void \*datos);
* static void \_juego\_ventana\_usuario\_destruir(void \*datos);

### LOGICA:

##### Funciones Declaradas

##### int logica\_inicializar(tLogica \*logica);

##### void logica\_destruir(tLogica \*logica);

##### void logica\_calc\_min\_res(const tLogica \*logica, unsigned \*anchoRes, unsigned \*altoRes);

##### int logica\_procesar\_turno(tLogica \*logica, SDL\_Keycode tecla);

##### void logica\_procesar\_movimientos(tLogica \*logica);

##### int logica\_iniciar\_juego(tLogica \*logica);

##### int logica\_nueva\_ronda(tLogica \*logica);

##### void logica\_fin\_juego(tLogica \*logica);

##### void logica\_mostrar\_historial\_movs(tCola \*movsJugador);

##### void logica\_actualizar(tLogica \*logica);

##### Funciones Internas:

* static int logica\_ubicacion\_valida(const tEscenario \*escenario, tUbicacion ubic);
* static tUbicacion \_logica\_mover\_fantasma\_bfs(tEscenario \*escenario, tEntidad \*jugador, tEntidad \*fantasma);
* static tUbicacion \_logica\_mover\_fantasma\_dfs(tEscenario \*escenario, tEntidad \*jugador, tEntidad \*fantasma);
* static void \_logica\_colocar\_fantasmas(tLogica \*logica);
* static void \_logica\_inicializar\_jugador(tEntidad \*jugador);
* static void \_logica\_colocar\_jugador(const tEscenario \*escenario, tEntidad \*jugador);
* static int \_logica\_fantasma\_debe\_usar\_bfs(const tEntidad \*fantasma);
* static void \_logica\_actualizar\_entidad(tEntidad \*entidad);
* static int \_logica\_encontrar\_casilla\_libre(tLogica \*logica, tUbicacion \*ubic);

### MENU:

##### Funciones Declaradas

##### tMenu\* menu\_crear(SDL\_Renderer \*renderer, unsigned capOpc, SDL\_Point ubicacion, eMenuTipo menuTipo);

##### void menu\_destruir(tMenu\* menu);

##### int menu\_agregar\_opcion(tMenu \*menu, int id, SDL\_Texture \*textura, unsigned tamAltura, tMenuAccion accion, eOpcionEstado estado);

##### void menu\_siguiente\_opcion(tMenu \*menu);

##### void menu\_anterior\_opcion(tMenu \*menu);

##### tMenuAccion menu\_confirmar\_opcion(tMenu \*menu);

##### void menu\_estado\_opcion(tMenu \*menu, int id, eOpcionEstado nuevoEstado);

##### void menu\_dibujar(tMenu\* menu);

### TEMPORIZADOR:

##### Funciones Declaradas

##### void temporizador\_inicializar(tTempor \*tempor, float duracion);

##### void temporizador\_actualizar(tTempor\* tempor);

##### eTemporEstado temporizador\_estado(const tTempor\* tempor);

##### void temporizador\_iniciar(tTempor\* tempor);

##### void temporizador\_pausar(tTempor\* tempor);

##### void temporizador\_reanudar(tTempor\* tempor);

##### float temporizador\_tiempo\_restante(tTempor\* temporizador);

### TEXTO:

##### Funciones Declaradas

##### SDL\_Texture\* texto\_crear\_textura(SDL\_Renderer \*renderer, TTF\_Font \*fuente, const char\* texto, SDL\_Color color);

##### int texto\_obtener\_tam(TTF\_Font \*fuente, const char \*texto, SDL\_Point \*tamTexto);

### VENTANA:

##### Funciones Declaradas

##### tVentana\* ventana\_crear(SDL\_Renderer \*renderer, tVentanaAccion accion, SDL\_Rect dimensiones, SDL\_Color cFondo, char conSombra);

##### void ventana\_destruir(tVentana \*ventana);

##### void ventana\_abrir(tVentana \*ventana);

##### void ventana\_cerrar(tVentana \*ventana);

##### void ventana\_actualizar(tVentana \*ventana, SDL\_Event \*event);

##### void ventana\_dibujar(tVentana \*ventana);

##### Funciones Internas:

* static float \_interpolacion\_lineal(float valorInicial, float valorFinal, float factor);
* static SDL\_Rect \_ventana\_dibujar(tVentana \*ventana);

### WIDGET:

##### Funciones Declaradas

##### tWidget\* widget\_crear\_campo\_texto(SDL\_Renderer \*renderer, SDL\_Point coords, TTF\_Font \*fuente, SDL\_Color cTexto, SDL\_Color cFondo, char visible);

##### tWidget\* widget\_crear\_boton\_texto(SDL\_Renderer \*renderer, SDL\_Point coords, const char\* texto, TTF\_Font \*fuente, SDL\_Rect rect, SDL\_Color cTexto, SDL\_Color cFondo, char visible);

##### tWidget\* widget\_crear\_contador(SDL\_Renderer \*renderer, SDL\_Point coords, SDL\_Texture \*icono, TTF\_Font \*fuente, SDL\_Color cTexto, int valor, char visible);

##### tWidget\* widget\_crear\_texto(SDL\_Renderer \*renderer, const char \*texto, SDL\_Point coords, TTF\_Font \*fuente, SDL\_Color cTexto, SDL\_Color cFondo, char visible);

##### int widget\_modificar\_valor(tWidget \*widget, void \*valor);

##### void widget\_modificar\_visibilidad(tWidget \*widget, char visible);

##### void widget\_alternar\_visibilidad(tWidget \*widget);

##### void widget\_dibujar(tWidget \*widget);

##### void widget\_destruir(tWidget \*widget);

##### Funciones Internas:

* static tWidget\* \_widget\_crear\_base(SDL\_Renderer \*renderer, SDL\_Point coords, TTF\_Font \*fuente, const char \*texto, SDL\_Color cTexto, eWidgetTipo tipo, char visible);
* static int \_widget\_campo\_texto\_actualizar(tWidget \*widget);
* static void \_widget\_campo\_texto\_dibujar(tWidget \*widget);
* static int \_widget\_boton\_actualizar(tWidget \*widget);
* static void \_widget\_boton\_dibujar(tWidget \*widget);
* static void \_widget\_boton\_destruir(tWidget \*widget);
* static int \_widget\_contador\_actualizar(tWidget \*widget);
* static void \_widget\_contador\_dibujar(tWidget \*widget);
* static int \_widget\_texto\_actualizar(tWidget \*widget);
* static void \_widget\_texto\_dibujar(tWidget \*widget);

DESCRIPCIÓN MÓDULOS SERVIDOR:

### BDATOS:

Contiene las funciones de Base de Datos. Estas son utilizadas por el servidor para proporcionar el servicio, y permiten la creación, inserción, manipulación, actualización y selección en tablas (archivos binarios).

### SERVIDOR:

Contiene las funciones que inicializan el socket y procesan solicitudes, derivando estas al servicio de base de datos interno.

FUNCIONES MÓDULOS SERVIDOR:

### BDATOS:

##### Funciones Declaradas

##### int bdatos\_iniciar(tBDatos \*bDatos);

##### int bdatos\_procesar\_solcitud(tBDatos \*bDatos, const char \*solicitud, tLista \*listaDatos, int \*cantRegistrosDatos, int \*tamRegistroDatos);

##### eSimbolo bdatos\_parsear\_comando(tSecuencia \*secuencia);

##### int bdatos\_apagar(tBDatos \*bDatos);

##### int bdatos\_insertar(tBDatos \*bDatos);

##### int bdatos\_actualizar(tBDatos \*bDatos);

##### int bdatos\_seleccionar(tBDatos \*bDatos);

##### const char\* bdatos\_obtener\_mensaje(eBDRetorno codigoError);

##### int bdatos\_cargar\_idx(tBDatos \*bDatos);

##### Funciones Internas:

* static int \_bdatos\_cerrar\_tabla(tBDatos \*bDatos);
* static int \_bdatos\_insertar(tBDatos \*bDatos);
* static int \_bdatos\_seleccionar(tBDatos \*bDatos, tLista \*listaDatos, int \*cantRegistrosDatos);
* static int \_bdatos\_tabla\_existe(const char \*nombreTabla);
* static int \_bdatos\_crear\_tabla(tBDatos \*bDatos, const char \*nombreTabla);
* static int \_bdatos\_construir\_encabezado(tEncabezado \*encabezado, const char \*nombreTabla, int cantCampos, tCampo \*campos);
* static int \_bdatos\_buscar\_campo(const tCampo \*campos, int cantCampos, tCampo \*campoEncontrado, const char \*nombreCampoLeido);
* static int \_bdatos\_manejar\_apertura\_tabla(tBDatos \*bDatos, const char \*nombreTabla);
* static int \_bdatos\_cmp\_indice(const void \*a, const void \*b);
* static int \_bdatos\_parsear(tSecuencia \*secuencia, tParsear parsear, void \*salida, unsigned tamSalida);
* static int \_bdatos\_parsear\_identificador(tSecuencia \*secuencia, void \*salida, unsigned tamSalida);
* static int \_bdatos\_parsear\_simbolo(tSecuencia \*secuencia, void \*salida, unsigned tamSalida);
* static int \_bdatos\_parsear\_caracter(tSecuencia \*secuencia, void \*salida, unsigned tamSalida);
* static int \_bdatos\_parsear\_numeros(tSecuencia \*secuencia, void \*salida, unsigned tamSalida);
* static int \_bdatos\_parsear\_texto(tSecuencia \*secuencia, void \*salida, unsigned tamSalida);
* static void \_bdatos\_crear\_secuencia(tSecuencia \*secuencia, const char \*buffer);
* static int \_bdatos\_parsear\_declaracion\_campos(tSecuencia \*secuencia, tCampo \*campos, int maxCampos, int \*cantCamposLeidos);
* static int \_bdatos\_leer\_campo\_tipo(tSecuencia \*secuencia, tCampo \*campo);
* static int \_bdatos\_parsear\_valores\_insercion(tSecuencia \*secuencia, const tEncabezado \*encabezado, tDatoParseado \*datosParseados, int \*cantParseados);
* static eSimbolo \_bdatos\_comparar\_simbolo(const char\* simbolo);

### SERVIDOR:

##### Funciones Declaradas

##### int servidor\_inicializar();

##### SOCKET servidor\_crear\_socket();

##### void servidor\_procesar\_solicitud(tBDatos \*bDatos, SOCKET \*sock, const char \*solicitud);

MÓDULOS COMUNES:

### ARBOL:

##### Funciones Declaradas

##### void arbol\_crear(tArbol \*arbol);

##### int arbol\_buscar(const tArbol \*arbol, void \*dato, unsigned tamDato, tCmp cmp);

##### int arbol\_insertar\_rec(tArbol \*arbol, const void \*dato, unsigned tamDato, tCmp cmp);

##### void arbol\_recorrer\_preorden(const tArbol \*arbol, void \*extra, tAccion accion);

##### void arbol\_recorrer\_posorden(const tArbol \*arbol, void \*extra, tAccion accion);

##### int arbol\_cargar\_de\_archivo(FILE \*arch, tArbol \*arbol, unsigned tamReg, tCmp cmp);

##### int arbol\_escribir\_en\_arch(FILE \*arch, tArbol \*arbol);

##### void arbol\_vaciar(tArbol \*arbol);

##### Funciones Internas:

* static tNodoArbol\*\* \_arbol\_buscar\_nodo(const tArbol \*arbol, const void \*dato, tCmp cmp);
* static void \_arbol\_escribir\_preorden(const tArbol \*arbol, FILE \*arch);

### COLA:

##### Funciones Declaradas

##### void cola\_crear(tCola \*cola);

##### int cola\_encolar(tCola \*cola, const void \*dato, unsigned tamDato);

##### int cola\_desencolar(tCola \*cola, void \*dato, unsigned tamDato);

##### int cola\_ver\_primero(const tCola \*cola, void \*dato, unsigned tamDato);

##### int cola\_vacia(const tCola \*cola);

##### int cola\_llena(const tCola \*cola, unsigned tamDato);

##### void cola\_vaciar(tCola \*cola);

### LISTA:

##### Funciones Declaradas

##### void lista\_crear(tLista \*lista);

##### void lista\_recorrer(const tLista \*lista, tAccion accion, void \*extra);

##### int lista\_insertar\_final(tLista \*lista, const void \*dato, unsigned tamDato);

##### int lista\_insertar\_comienzo(tLista \*lista, const void \*dato, unsigned tamDato);

##### void lista\_vaciar(tLista \*lista);

##### int lista\_llena(const tLista \*lista, unsigned tamDato);

##### int lista\_vacia(const tLista \*lista);

##### int lista\_sacar\_primero(tLista \*lista, void \*dato, unsigned tamDato);

##### int lista\_sacar\_ultimo(tLista \*lista, void \*dato, unsigned tamDato);

##### int lista\_ver\_primero(const tLista \*lista, void \*dato, unsigned tamDato);

##### int lista\_ver\_ultimo(const tLista \*lista, void \*dato, unsigned tamDato);

##### int lista\_insertar\_en\_orden(tLista \*lista, const void \*dato, unsigned tamDato, int modo, tCmp cmp);

### MATRIZ:

##### Funciones Declaradas

##### void\*\* matriz\_crear(size\_t columnas, size\_t filas, size\_t tamElem);

##### void matriz\_destruir(void\*\* matriz, size\_t filas);

##### void matriz\_inicializar(void\*\* matriz, void \*dato, size\_t columnas, size\_t filas, size\_t tamElem);

### PILA:

##### Funciones Declaradas

##### void pila\_crear(tPila \*pila);

##### int pila\_llena(const tPila \*pila, unsigned tamDato);

##### int pila\_apilar(tPila \*pila, const void \*dato, unsigned tamDato);

##### int pila\_tope(const tPila \*pila, void \*dato, unsigned tamDato);

##### int pila\_vacia(const tPila \*pila);

##### int pila\_desapilar(tPila \*pila, void \*dato, unsigned tamDato);

##### void pila\_vaciar(tPila \*pila);

### VECTOR:

##### Funciones Declaradas

##### int vector\_crear(tVector\* vector, size\_t tamElem);

##### void vector\_vaciar(tVector \*vector);

##### void vector\_destruir(tVector\* vector);

##### int vector\_cargar\_de\_archivo(tVector\* vector, const char\* nombreArch, size\_t tamElem);

##### void vector\_recorrer(tVector\* vector, Accion accion, void\* extra);

##### void vector\_ordenar(tVector\* vector, int metodo, Cmp cmp);

##### int vector\_ord\_buscar(tVector\* vector, void\* elem, Cmp cmp);

##### int vector\_ord\_insertar(tVector\* vector, void\* elem, Cmp cmp, Actualizar actualizar);

##### int vector\_insertar\_al\_final(tVector\* vector, void\* elem);

##### void vector\_it\_crear(tVectorIterador\* it, tVector\* vector);

##### void\* vector\_it\_primero(tVectorIterador\* it);

##### void\* vector\_it\_siguiente(tVectorIterador\* it);

##### int vector\_it\_fin(tVectorIterador\* it);

##### Funciones Internas:

* static void \_ordenar\_seleccion(tVector\* vector, Cmp cmp);