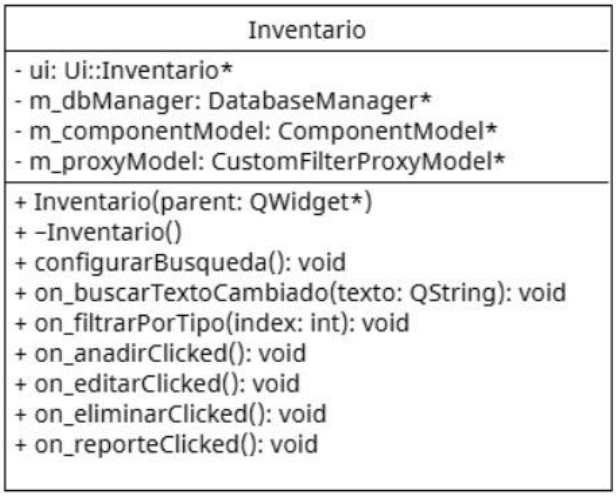
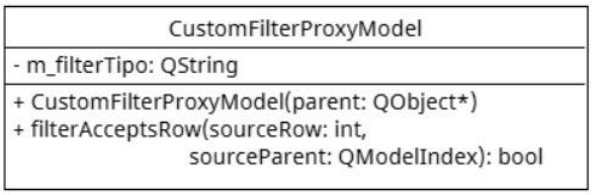
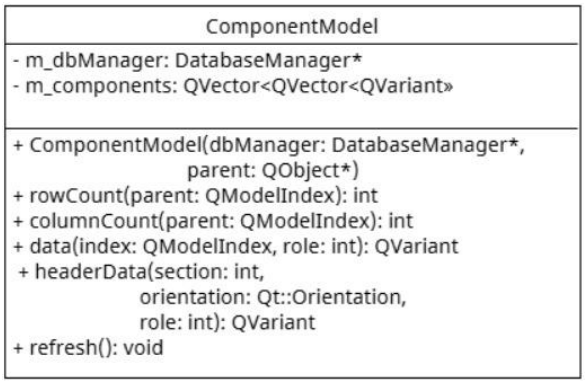
Documentación Gestor-inventario

Diagramas UML:



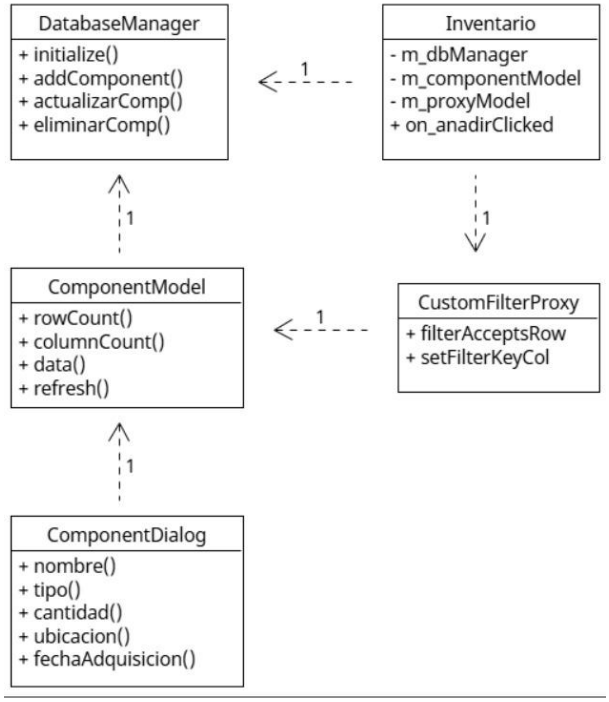




DatabaseManager
-m_db: QSqlDatabase
+ DatabaseManager(parent: QObject*) + initialize(databasePath: QString): bool + createTables(): bool + addComponent(name: QString, type: QString, quantity: int, location: QString, purchaseDate: QDate): bool + actualizarComponente(id: int, nombre: QString, tipo: QString, cantidad: int, ubicacion: QString, fecha: QDate): bool + getAllComponents(): QVector<QStringList> + eliminarComponente(id: QString): bool + lastError(): QSqlError + ~DatabaseManager()

ComponentDialog
- ui: Ui::ComponentDialog*
+ ComponentDialog(parent: QWidget*, nombre: QString, tipo: QString, cantidad: int, ubicacion: QString, fecha: QDate) + ~ComponentDialog() + nombre(): QString + tipo(): QString + cantidad(): int + ubicacion(): QString + fechaAdquisicion(): QDate

Estructura:



Descripción de clases:

- ## Inventario:

Inventario::Inventario(QWidget *parent):

Constructor de la clase Inventario.
Inicializa la interfaz gráfica, la conexión a la base de datos, los modelos de datos y configura la tabla y las conexiones de señales y slots para la interacción de la interfaz.

Inventario::~Inventario()

Destructor de la clase Inventario.
Libera la memoria de la interfaz gráfica y realiza limpieza de recursos.

void Inventario::configurarBusqueda()

Configura el modelo proxy para que la búsqueda no distinga mayúsculas/minúsculas y afecte a todas las columnas.
Conecta el campo de búsqueda de texto con el slot que actualiza el filtro en tiempo real.

void Inventario::on_buscarTextoCambiado(const QString &texto)

Slot que se ejecuta cuando el usuario escribe en el campo de búsqueda.
Actualiza el filtro del modelo proxy para mostrar solo las filas que coincidan con el texto ingresado.

void Inventario::on_filtrarPorTipo(int index)

Slot que se ejecuta cuando el usuario cambia el filtro de tipo de componente.
Aplica el filtro seleccionado al modelo proxy para mostrar solo los componentes del tipo elegido o todos si se selecciona "Todos".

void Inventario::on_anadirClicked()

Slot que se ejecuta al presionar el botón "Añadir".
Abre un diálogo para ingresar los datos de un nuevo componente y, si se confirma, lo agrega a la base de datos y actualiza la tabla.

void Inventario::on_editarClicked()

Slot que se ejecuta al presionar el botón "Editar" o al hacer doble clic en una fila.
Abre un diálogo con los datos del componente seleccionado para editarlo y, si se confirma, actualiza el componente en la base de datos y refresca la tabla.

void Inventario::on_eliminarClicked()

Slot que se ejecuta al presionar el botón "Eliminar".
Elimina el componente seleccionado de la base de datos y actualiza la tabla, mostrando un mensaje de éxito o error según el resultado.

void Inventario::on_reporteClicked()

Slot que se ejecuta al presionar el botón "Reporte".
Genera un archivo CSV y un archivo PDF con todos los componentes del inventario, permitiendo al usuario elegir la ubicación de guardado y mostrando un mensaje de éxito al finalizar.

- ## CustomFilterProxyModel:

CustomFilterProxyModel::CustomFilterProxyModel(QObject *parent)

Constructor de la clase CustomFilterProxyModel.
Inicializa el modelo proxy de filtrado, llamando al constructor base QSortFilterProxyModel. Permite que el modelo sea usado como intermediario entre los datos y la vista, soportando filtrado y ordenamiento.

bool CustomFilterProxyModel::filterAcceptsRow(int sourceRow, const QModelIndex &sourceParent) const

Método principal de filtrado de filas.
Decide si una fila del modelo fuente debe mostrarse en la vista filtrada.

- Primero, verifica si hay un filtro de texto activo (búsqueda) y recorre todas las columnas de la fila para ver si alguna contiene el texto buscado.
- Si ninguna columna coincide con el filtro de texto, la fila es descartada.
- Luego, verifica si hay un filtro de tipo activo (m_filterTipo). Si está definido y el valor de la columna "Tipo" no coincide, la fila es descartada.
- Si pasa ambos filtros, la fila es aceptada y mostrada en la vista.

- ## ComponentModel:

ComponentModel::ComponentModel(DatabaseManager* dbManager, QObject* parent)

Constructor del modelo de componentes.
Inicializa el modelo de tabla, guarda el puntero al gestor de base de datos y carga los datos iniciales desde la base de datos llamando a refresh().

int ComponentModel::rowCount(const QModelIndex&) const

Devuelve el número de filas del modelo, es decir, la cantidad de componentes almacenados.
Se utiliza por la vista para saber cuántas filas mostrar.

int ComponentModel::columnCount(const QModelIndex&) const

Devuelve el número de columnas del modelo, que corresponde a los campos de cada componente (ID, Nombre, Tipo, Cantidad, Ubicación, Fecha).
Se utiliza por la vista para saber cuántas columnas mostrar.

QVariant ComponentModel::data(const QModelIndex& index, int role) const

Devuelve el dato correspondiente a una celda específica, dependiendo del rol solicitado:

- Para Qt::DisplayRole y Qt::EditRole, retorna el valor de la celda.
- Para Qt::BackgroundRole en la columna de cantidad, retorna color rojo si la cantidad es menor a 5.
- Para otros roles o índices inválidos, retorna un valor vacío.

QVariant ComponentModel::headerData(int section, Qt::Orientation orientation, int role) const

Devuelve los encabezados de columna para la vista de tabla cuando el rol es Qt::DisplayRole y la orientación es horizontal.

Retorna el nombre de la columna correspondiente (ID, Nombre, Tipo, etc).

void ComponentModel::refresh()

Recarga todos los datos de componentes desde la base de datos, notificando a la vista que el modelo ha cambiado.
Llama a beginResetModel() y endResetModel() para actualizar correctamente la vista, y emite la señal dataChanged para forzar la actualización visual.

- ## DatabaseManager:

DatabaseManager::DatabaseManager(QObject *parent)

Constructor de la clase DatabaseManager.
Inicializa el objeto gestor de base de datos, permitiendo la gestión de la conexión y operaciones sobre la base de datos SQLite.

bool DatabaseManager::initialize(const QString &databasePath)

Inicializa la conexión a la base de datos SQLite en la ruta especificada.
Crea una conexión nombrada, abre la base de datos y llama a createTables() para asegurarse de que las tablas necesarias existan.

bool DatabaseManager::createTables()

Crea la tabla components en la base de datos si no existe.
Define los campos: id, name, type, quantity, location y purchase_date.

bool DatabaseManager::addComponent(const QString &name, const QString &type, int quantity, const QString &location, const QDate &purchaseDate)

Inserta un nuevo componente en la tabla components usando los valores proporcionados.
Prepara y ejecuta una consulta SQL de inserción.

bool DatabaseManager::actualizarComponente(int id, const QString& nombre, const QString& tipo, int cantidad, const QString& ubicacion, const QDate& fecha)

Actualiza los datos de un componente existente identificado por su ID.
Prepara y ejecuta una consulta SQL de actualización con los nuevos valores.

QVector<QStringList> DatabaseManager::getAllComponents() const

Obtiene todos los componentes almacenados en la base de datos.
Devuelve un vector de listas de cadenas, donde cada lista representa una fila (componente) con sus campos.

bool DatabaseManager::eliminarComponente(const QString &id)

Elimina un componente de la base de datos usando su ID.
Prepara y ejecuta una consulta SQL de eliminación.

QSqlError DatabaseManager::lastError() const

Devuelve el último error registrado por la base de datos.
Permite obtener información detallada sobre fallos en operaciones SQL.

DatabaseManager::~DatabaseManager()

Destructor de la clase DatabaseManager.
Cierra la conexión a la base de datos si está abierta, liberando recursos.

- ## ComponentDialog:

ComponentDialog::ComponentDialog(QWidget *parent, const QString& nombre, const QString& tipo, int cantidad, const QString& ubicacion, const QDate& fecha)

Constructor del diálogo para añadir o editar componentes.
Inicializa la interfaz gráfica, configura los controles del formulario (tipos, límites de cantidad, calendario), establece los valores iniciales de los campos (para edición o valores por defecto) y ajusta el título de la ventana según el contexto (añadir o editar).

ComponentDialog::~ComponentDialog()

Destructor del diálogo.
Libera la memoria asociada a la interfaz gráfica generada por Qt Designer.

QString ComponentDialog::nombre() const

Devuelve el texto ingresado en el campo de nombre, eliminando espacios al inicio y final.
Permite obtener el nombre del componente desde el formulario.

QString ComponentDialog::tipo() const

Devuelve el tipo seleccionado en el combo de tipos.
Permite obtener el tipo de componente elegido por el usuario.

int ComponentDialog::cantidad() const

Devuelve el valor seleccionado en el control de cantidad.
Permite obtener la cantidad ingresada por el usuario.

QString ComponentDialog::ubicacion() const

Devuelve el texto ingresado en el campo de ubicación, eliminando espacios al inicio y final.
Permite obtener la ubicación del componente desde el formulario.

QDate ComponentDialog::fechaAdquisicion() const

Devuelve la fecha seleccionada en el control de fecha.
Permite obtener la fecha de adquisición del componente.

Jonatan García Vásquez

Ivan Daniel Carpentier

Leydi Juliana Solorza