PROYECTO FINAL ALSE

PRESENTADO POR:

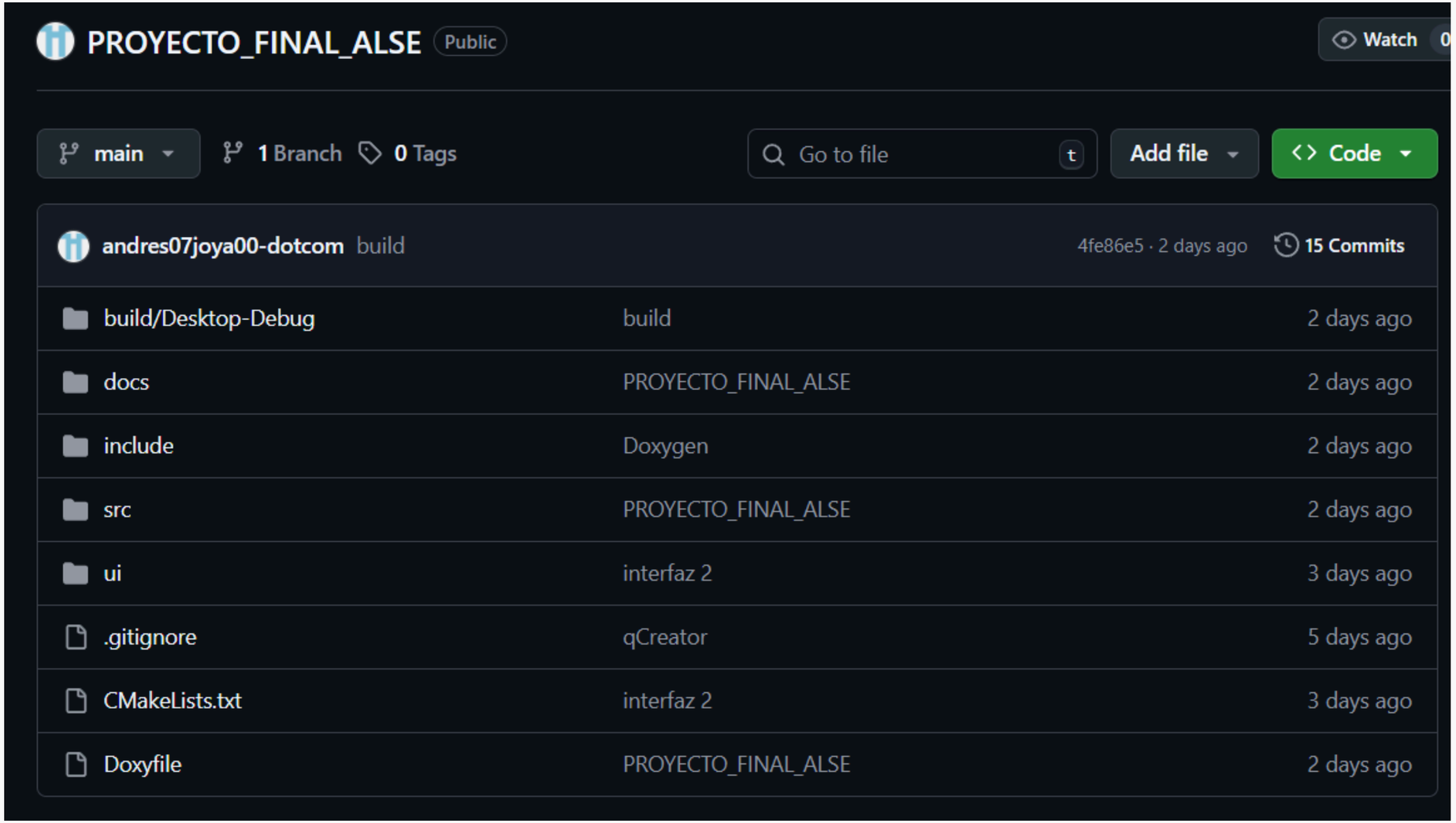
- MIGUEL ANGEL BOHORQUEZ HERNANDEZ
- ANDRES FELIPE JOYA RUIZ

PROYECTO ELEGIDO

3.3 GESTOR DE INVENTARIO PARA HOGAR O LABORATORIO:

SISTEMA DE INVENTARIO PARA ADMINISTRAR COMPONENTES ELECTRONICOS O DISPOSITIVOS DEL HOGAR/LABORATORIO. IMPLEMENTADO EN C++ CON QT Y BASE DE DATOS SQLITE, ORIENTADO A DISPOSITIVOS DE BAJO CONSUMO COMO RASPBERRY PI.

TRABAJO COLABORATIVO USANDO GITHUB



EXPLICACION FUNCION 1

Componente:

Componente.h

include > C component.h > Component

```
#ifndef COMPONENT_H
#define COMPONENT_H

#include <QString>

/**
 * @class Component
 * @brief Representa un componente almacenado en el inventario.
 *
 * Esta clase modela un componente físico dentro del sistema de inventario.
 * Incluye información básica como su identificador, nombre, tipo, cantidad,
 * ubicación y fecha de compra. Provee métodos para obtener y modificar cada
 * uno de estos atributos.
 */
class Component
{
public:
    /**
     * @brief Constructor por defecto.
     *
     * Inicializa todos los campos a valores base (0 o vacío).
     */
    Component();

    /**
     * @brief Constructor con parámetros.
     * @param id Identificador único del componente.
     * @param name Nombre del componente.
     * @param type Tipo o categoría del componente.
     * @param quantity Cantidad disponible en inventario.
     * @param location Ubicación física dentro del almacén.
     * @param purchase_date Fecha de compra del componente.
     */
```

LA CLASE COMPONENT REPRESENTA UN ELEMENTO DEL INVENTARIO Y ALMACENA TODA SU INFORMACIÓN BÁSICA: ID, NOMBRE, TIPO, CANTIDAD, UBICACIÓN Y FECHA DE COMPRA. SU ESTRUCTURA SE BASA EN LA ENCAPSULACIÓN, POR LO QUE CADA DATO SE GESTIONA MEDIANTE MÉTODOS “GET” Y “SET”, GARANTIZANDO UN ACCESO CONTROLADO Y SEGURO. LOS DOS CONSTRUCTORES PERMITEN CREAR OBJETOS VACÍOS O COMPLETAMENTE INICIALIZADOS SEGÚN LA NECESIDAD DEL PROGRAMA. EN CONJUNTO, ESTA CLASE SIRVE COMO MODELO DE DATOS PARA QUE EL SISTEMA PUEDA MANEJAR COMPONENTES DE FORMA ORGANIZADA Y CONSISTENTE.

EXPLICACION FUNCION 2

Database:

databaseManager.h

include > C DatabaseManager.h > DatabaseManager

```
1  #ifndef DATABASEMANAGER_H
2  #define DATABASEMANAGER_H
3
4  #include <QSqlDatabase>
5
6  /**
7   * @class DatabaseManager
8   * @brief Clase encargada de gestionar la conexión con la base de datos.
9   *
10  * Esta clase implementa un patrón similar a Singleton para proporcionar
11  * una única instancia de conexión a la base de datos SQLite utilizada
12  * por la aplicación. El método estático @ref getDatabase retorna una
13  * referencia a dicha conexión evitando crear múltiples instancias.
14  *
15  * Se utiliza QSqlDatabase para manejar la apertura y configuración de
16  * la base de datos según lo requiera Qt.
17  */
18 class DatabaseManager
19 {
20 public:
21     /**
22      * @brief Obtiene la base de datos principal del sistema.
23      *
24      * Si la conexión aún no se ha inicializado, se configura y se abre.
25      * En llamadas posteriores, devuelve la misma instancia ya configurada.
26      *
27      * @return Instancia global de QSqlDatabase utilizada por el sistema.
28      */
29     static QSqlDatabase getDatabase();
```

LA CLASE DATABASEMANAGER CENTRALIZA LA CREACIÓN Y ADMINISTRACIÓN DE LA CONEXIÓN SQLITE. SU MÉTODO ESTÁTICO GETDATABASE() GARANTIZA QUE TODA LA APLICACIÓN UTILICE UNA SOLA INSTANCIA DE QSQLDATABASE, EVITANDO CONEXIONES DUPLICADAS O INCONSISTENCIAS. LA PRIMERA VEZ QUE SE LLAMA, INICIALIZA Y ABRE LA BASE DE DATOS; EN LLAMADAS POSTERIORES, SIMPLEMENTE DEVUELVE LA MISMA INSTANCIA YA CONFIGURADA. ESTO ASEGURA UNA GESTIÓN ORDENADA, EFICIENTE Y SEGURA DE LA CONEXIÓN A LA BASE DE DATOS EN TODO EL SISTEMA.

EXPLICACION FUNCION 3

Inventario:

InventoryManager.h

include > C InventoryManager.h > InventoryManager

```h
#ifndef INVENTORYMANAGER_H
#define INVENTORYMANAGER_H

#include <QObject>
#include <QList>
#include <QSqlDatabase>

/*
 * Estructura que representa un ítem dentro del inventario.
 * Contiene toda la información necesaria para leer o escribir
 * un registro desde la base de datos.
 */
struct InventoryItem {
    int id;                     // Identificador único del ítem (PRIMARY KEY)
    QString nombre;             // Nombre del componente o elemento
    QString tipo;               // Tipo o categoría del ítem
    int cantidad;               // Cantidad disponible en inventario
    QString ubicacion;          // Ubicación física dentro del almacén
    QString fechaAdquisicion;   // Fecha en que se adquirió el ítem
};

/*
 * Clase InventoryManager
 * ----------------------
 * Administra la comunicación entre la aplicación y la base de datos SQLite.
 * Permite crear la tabla, agregar, editar, eliminar y consultar ítems.
 * Se utiliza como capa intermedia entre la lógica del programa y el motor SQL.
 */
class InventoryManager : public QObject
{
```

LA CLASE INVENTORYMANAGER ACTÚA COMO LA CAPA QUE CONECTA LA APLICACIÓN CON LA BASE DE DATOS SQLITE, PERMITIENDO GESTIONAR TODOS LOS REGISTROS DEL INVENTARIO. SU OBJETIVO ES CENTRALIZAR LAS OPERACIONES PRINCIPALES: CREAR LA TABLA, AGREGAR ÍTEMS, ACTUALIZAR DATOS, ELIMINARLOS Y CONSULTARLOS. CADA MÉTODO CORRESPONDE A UNA OPERACIÓN SQL ESPECÍFICA, Y AL ENCAPSULAR ESTA LÓGICA EN UNA SOLA CLASE, SE MANTIENE EL CÓDIGO ORGANIZADO, LIMPIO Y SEPARADO DE LA INTERFAZ GRÁFICA. GRACIAS A ESTO, EL RESTO DEL PROGRAMA PUEDE MANIPULAR EL INVENTARIO DE FORMA SENCILLA Y SEGURA, SIN PREOCUPARSE POR DETALLES DE SQL O CONEXIONES A LA BASE DE DATOS.

EXPLICACION FUNCION 4

Reporte:

Report.h

include > C report.h > ...

```
/**
 * @file CSVReport.h
 * @brief Declaración de la clase responsable de generar reportes en formato CSV.
 *
 * La clase CSVReport permite crear un archivo CSV a partir de una lista
 * de objetos InventoryItem, exportando la información del inventario
 * para análisis o respaldo externo.
 */

#ifndef CSVREPORT_H
#define CSVREPORT_H

#include <QString>
#include <QList>

/**
 * @struct InventoryItem
 * @brief Estructura que representa un elemento del inventario.
 *
 * Esta estructura es declarada en otro archivo (InventoryManager.h),
 * y se utiliza aquí solo como referencia para exportar datos.
 */
struct InventoryItem;
```

LA CLASE CSVREPORT GENERA UN ARCHIVO CSV RECORRIENDO LA LISTA DE INVENTORYITEM Y ESCRIBIENDO CADA CAMPO EN FORMATO TEXTO USANDO CLASES DE QT COMO QFILE Y QTEXTSTREAM. CUANDO SE LLAMA AL MÉTODO GENERATE(), PRIMERO INTENTA ABRIR EL ARCHIVO INDICADO POR FILEPATH EN MODO DE ESCRITURA. SI LA APERTURA ES EXITOSA, CREA UN QTEXTSTREAM ASOCIADO AL ARCHIVO, LO QUE PERMITE ESCRIBIR LÍNEAS DE TEXTO DE MANERA SENCILLA. LUEGO ESCRIBE UNA FILA DE ENCABEZADOS FIJA (POR EJEMPLO: "ID,NOMBRE,TIPO,CANTIDAD,UBICACIÓN,FECHA"). DESPUÉS, RECORRE LA LISTA RECIBIDA EN EL PARÁMETRO ITEMS Y, PARA CADA ELEMENTO, ESCRIBE UNA LÍNEA NUEVA CON SUS DATOS SEPARADOS POR COMAS. QT CONVIERTE AUTOMÁTICAMENTE LOS VALORES INT Y QSTRING A TEXTO, POR LO QUE EL PROCESO ES DIRECTO Y SEGURO. AL FINALIZAR, CIERRA EL ARCHIVO Y DEVUELVE TRUE, INDICANDO QUE EL REPORTE SE GENERÓ CORRECTAMENTE. SI EL ARCHIVO NO PUEDE ABRIRSE, RETORNA FALSE.

EXPLICACION FUNCION 5

Interfaz Visual:

mainwindow.h

include > C mainwindow.h > ...

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QWidget>
#include <QLineEdit>
#include <QSpinBox>
#include <QDateEdit>
#include <QTableView>
#include <QSqlQueryModel>
#include <QSortFilterProxyModel>
#include <QPushButton>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QLabel>
#include <QMessageBox>

#include "InventoryManager.h"
#include "component.h"
#include "report.h"

/*
 * Clase AddDialog
 * ---------------
 * Ventana pequeña para agregar un nuevo ítem al inventario.
 * Contiene campos de entrada para nombre, tipo, cantidad, ubicación y fecha.
 * Emite señales accepted() y cancelled() según la acción del usuario.
 */
class AddDialog : public QWidget {
    Q_OBJECT
public:
    explicit AddDialog(QWidget *parent = nullptr);
```

ESTE ARCHIVO DEFINE LA ESTRUCTURA DE LA INTERFAZ PRINCIPAL DEL SISTEMA DE INVENTARIO EN QT. LA CLASE ADDDIALOG CREA UNA VENTANA PEQUEÑA PARA QUE EL USUARIO INGRESE LOS DATOS DE UN NUEVO ÍTEM, MIENTRAS QUE LA CLASE MAINWINDOW CONTIENE LA VENTANA PRINCIPAL DONDE SE MUESTRAN, FILTRAN Y ADMINISTRAN LOS PRODUCTOS DEL INVENTARIO. MAINWINDOW RECIBE UNA BASE DE DATOS SQLITE YA ABIERTA DESDE EL MAIN.CPP Y, MEDIANTE UN INVENTORYMANAGER, EJECUTA LAS OPERACIONES SQL (AGREGAR, EDITAR, ELIMINAR Y CONSULTAR). LOS DATOS SE MUESTRAN EN PANTALLA USANDO UN QSQLQUERYMODEL, Y UN QSORTFILTERPROXYMODEL PERMITE FILTRAR LOS RESULTADOS MIENTRAS SE ESCRIBE EN LA BARRA DE BÚSQUEDA. CADA BOTÓN DE LA INTERFAZ LLAMA UN SLOT (ONADD, ONEDIT, ONDELETE, ETC.) QUE ACTUALIZA TANTO LA BASE DE DATOS COMO LA TABLA VISUAL USANDO REFRESHMODEL().

INICIALIZACION

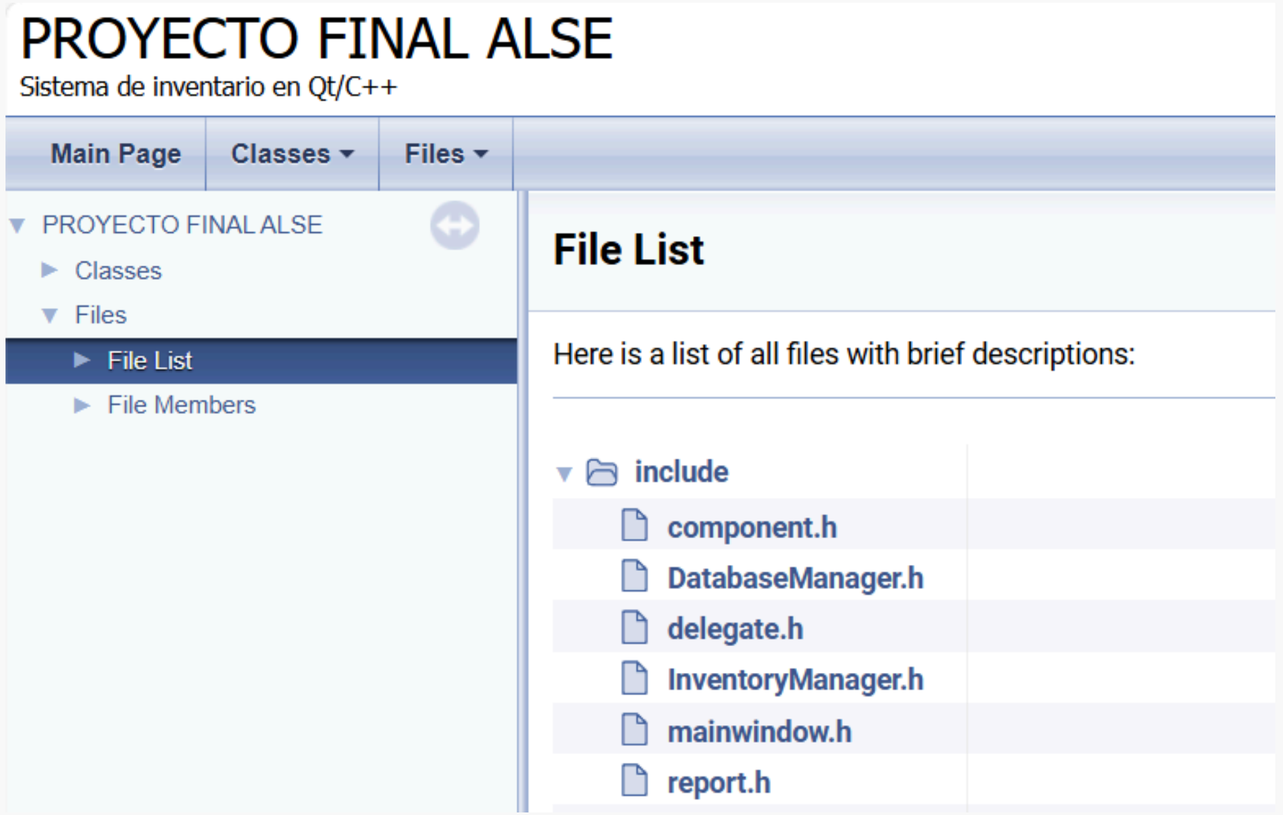
Main.cpp

src > main.cpp > ...

```cpp
/**
 * @file main.cpp
 * @brief Punto de entrada principal de la aplicación de inventario.
 *
 * Este archivo inicializa el entorno Qt, establece la conexión con la base
 * de datos mediante DatabaseManager y crea la ventana principal de la aplicación.
 */

#include <QApplication>
#include <QMessageBox>
#include "DatabaseManager.h"
#include "mainwindow.h"

/**
 * @brief Función principal de la aplicación.
 *
 * Inicializa QApplication, intenta abrir la base de datos y crea la ventana
 * principal si la conexión es exitosa.
 *
 * @param argc Número de argumentos de línea de comandos.
 * @param argv Arreglo con los argumentos de línea de comandos.
 * @return int Código de salida de la aplicación.
 */
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    // Obtener la conexión a la base de datos desde DatabaseManager
    QSqlDatabase db = DatabaseManager::getDatabase();

    // Verificar si la base de datos se abrió correctamente
    if (!db.isValid() || !db.isOpen()) {
        QMessageBox::critical(nullptr, "Error", "No se pudo abrir la base de datos.");
        return -1;
    }
```

DOCUMENTACION

Doxygen



MUCHAS GRACIAS