

Proyecto final - Entrega preliminar

Datos del alumno:

Salvador Esaú Rodríguez González

código: 223386151

Datos de la materia:

Materia: Estructuras de datos

Profesor: Alfredo Gutiérrez Hernández

NRC: 210901

Sección: D02

Fecha de elaboración:

16 de octubre de 2024

Autoevaluación			
Concepto	Sí	No	Acumulación
Bajé el trabajo de internet o alguien me lo pasó (aunque sea de forma parcial)	-100 pts	0 pts	0
Incluí el código fuente en formato de texto (sólo si funciona cumpliendo todos los requerimientos)	+25 pts	0 pts	25
Incluí las impresiones de pantalla (sólo si funciona cumpliendo todos los requerimientos)	+25 pts	0 pts	25
Incluí una portada que identifica mi trabajo (nombre, código, materia, fecha, título)	+25 pts	0 pts	25
Incluí una descripción y conclusiones de mi trabajo	+25 pts	0 pts	25
Suma:			100

Introducción

El proyecto **Recetario Digital** se desarrollará en esta entrega como una **entrega preliminar** para la materia de Estructuras de Datos. Su objetivo es proporcionar un software que facilite a los chefs la captura, almacenamiento y gestión de recetas de los platillos que le ofrece a todos sus clientes, permitiendo una organización eficiente de la información. A continuación, se verá algunas de las características principales:

En esta entrega preliminar del proyecto **Recetario Digital**, se han definido las siguientes clases para su desarrollo:

1. **Receta**: Esta clase representa una receta y almacenará los atributos necesarios, tales como el nombre del platillo, el autor, el tiempo de preparación, el procedimiento y una lista de ingredientes. La relación entre la receta y sus ingredientes se gestionará mediante la inclusión de una lista de ingredientes como un atributo de composición.
2. **Ingrediente**: Esta clase permitirá representar cada ingrediente que forma parte de una receta, incluyendo sus características como el nombre, la cantidad y la unidad de medida.
3. **ListaRecetas**: Esta clase gestionará una colección de objetos de tipo **Receta**. La implementación de esta lista para esta entrega se implementará con arreglos dinámicos que almacenan punteros a objetos de tipo Receta, lo que proporcionará un acceso eficiente y permitirá realizar operaciones como inserciones y eliminaciones de recetas de manera eficaz.
4. **ListaIngredientes**: En contraste con **ListaRecetas**, esta clase se encargará de manejar una colección de objetos de tipo **Ingrediente**. La implementación de **ListaIngredientes** se realizará utilizando **arreglos dinámicos** que almacenan

punteros a objetos de tipo Ingrediente, lo que permitirá la gestión eficiente de los ingredientes asociados a cada receta.

Funcionalidades del Recetario Digital

El Recetario Digital ofrecerá diversas funcionalidades, entre las que se destacan:

- Visualización de la lista de recetas almacenadas, con la opción de filtrarlas por categoría (Desayuno, Comida, Cena o Navideño).
- Agregar, modificar y eliminar recetas o ingredientes.
- Buscar y mostrar una receta específica a través del nombre del platillo o su categoría, mostrando todos los atributos relevantes (categoría, nombre, autor, ingredientes, tiempo de preparación y procedimiento).
- Ordenar las recetas por nombre o por tiempo de preparación.
- Implementar métodos para manejar los ingredientes de cada receta, asegurando que la inserción de ingredientes se realice de forma ordenada.
- Almacenar y leer el recetario en el disco para mantener la información permanentemente y se carguen las recetas previamente almacenadas de forma automática

Interfaz Gráfica

Además, el desarrollo del Recetario Digital incluirá una interfaz gráfica creada con **Qt**, que proporcionará un entorno visual intuitivo y atractivo para el usuario. La interfaz permitirá una interacción sencilla con las funcionalidades del recetario digital, facilitando la gestión de recetas y la visualización de los platillos ofrecidos.

Para la gestión de la visualización de algunos elementos en la interfaz se implementaron 2 clases Adicionales:

1. **RecipeCard:** Esta clase representa una tarjeta visual que muestra la información principal de una receta en la interfaz gráfica del usuario. Una imagen del platillo, el tiempo de preparación, nombre de la receta, categoría y 2 iconos, uno para modificar y el otro para eliminar, esto para que sea de forma mas directa y sencilla, al dar clic en la imagen abrirá la información detallada mostrando todos los demás atributos.
2. **IngredientWidget:** Esta clase representa un componente visual para mostrar y gestionar la información de un ingrediente al añadir o modificar una receta.

Código Fuente

Main:

```
#include "mainwindow.hpp"

#include <QApplication>
#include <QFile>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    a.setStyle("Fusion");
    QFile file(":/style.qss");
    if (file.open(QFile::ReadOnly)) {
        QString styleSheet = file.readAll();
        a.setStyleSheet(styleSheet);
        file.close();
    } else {
        qWarning("No se pudo abrir el archivo de estilo.");
    }
    MainWindow w;
    w.show();
    return a.exec();
}
```

Clase Receta:

```
#ifndef RECETA_HPP
#define RECETA_HPP

#include <QString>
#include <QTextStream>
#include <iostream>
#include "nombre.hpp"
#include "listaIngredientes.hpp"

class Receta{
public:
    enum class Categoria {
        DESAYUNO,
        COMIDA,
        CENA,
    };
};
```

```

        NAVIDEÑO
    };

private:
    int id;
    QString nombre;
    QString preparacion;
    QString rutaImagen;
    Categoria categoria;
    Nombre autor;
    int tiempoPreparacion;
    ListaIngredientes listaIngredientes;

public:
    Receta();
    Receta(const Receta&);
    ~Receta();

    void copiarTodo(const Receta&);

    void setNombre(const QString&);
    void setPreparacion(const QString&);
    void setCategoria(const Categoria&);
    void setAutor(const Nombre&);
    void setTiempoPreparacion(const int&);
    void setId(const int&);
    void setRutaImagen(const QString &newRutaImagen);

    void agregarIngrediente(Ingrediente *ingrediente);
    void eliminarTodosLosIngredientes();

    QString getNombre() const;
    QString getPreparacion() const;
    Categoria getCategoria() const;
    Nombre getAutor() const;
    int getTiempoPreparacion() const;
    int getId() const;
    ListaIngredientes& getListaIngredientes();
    QString getRutaImagen() const;

    Receta& operator=(const Receta&);

    bool operator==(const Receta& other) const;
    bool operator!=(const Receta& other) const;
    bool operator<(const Receta& other) const;

```

```

    bool operator>(const Receta& other) const;
    bool operator<=(const Receta& other) const;
    bool operator>=(const Receta& other) const;

    friend QTextStream& operator<<(QTextStream& , const Receta&);
    friend QTextStream& operator>>(QTextStream& , Receta& );

    QString getCategoriaToQString() const;
    int getCategoriaToInt() const;

    static int compararPorNombre(const Receta&,const Receta&);
    static int compararPorTiempoPreparacion(const Receta&,const Receta&);

};

#endif // RECETA_HPP

#include "receta.hpp"

Receta::Receta():listaIngredientes() {}

Receta::Receta(const Receta& otra){
    copiarTodo(otra);
}

Receta::~~Receta() {
}

void Receta::copiarTodo(const Receta& otra){
    this->nombre=otra.nombre;
    this->preparacion=otra.preparacion;
    this->categoria=otra.categoria;
    this->autor=otra.autor;
    this->tiempoPreparacion=otra.tiempoPreparacion;
    this->rutaImagen=otra.rutaImagen;
    this->id=otra.id;

    listaIngredientes = otra.listaIngredientes;
}

void Receta::setNombre(const QString & nuevoNombre){
    nombre=nuevoNombre;
}

```



```

void Receta::setPreparacion(const QString & nuevaPreparacion){
    preparacion=nuevaPreparacion;
}

void Receta::setRutaImagen(const QString &newRutaImagen){
    rutaImagen = newRutaImagen;
}

void Receta::setCategoria(const Categoria & nuevaCategoria){
    categoria=nuevaCategoria;
}

void Receta::setAutor(const Nombre & nuevoAutor){
    autor=nuevoAutor;
}

void Receta::setTiempoPreparacion(const int & nuevoTiempo){
    tiempoPreparacion=nuevoTiempo;
}

void Receta::setId(const int & nuevoId){
    id=nuevoId;
}

void Receta::agregarIngrediente(Ingrediente *ingrediente){
    listaIngredientes.insertarIngrediente(ingrediente);
}

void Receta::eliminarTodosLosIngredientes(){
    listaIngredientes.anular();
}

QString Receta::getNombre() const{
    return nombre;
}

QString Receta::getPreparacion() const{
    return preparacion;
}

QString Receta::getRutaImagen() const{
    return rutaImagen;
}

```

```

Receta::Categoria Receta::getCategoria() const{
    return categoria;
}

Nombre Receta::getAutor() const{
    return autor;
}

int Receta::getTiempoPreparacion() const{
    return tiempoPreparacion;
}

int Receta::getId() const{
    return id;
}

ListaIngredientes& Receta::getListIngredientes() {
    return listaIngredientes;
}

Receta& Receta::operator=(const Receta& otra){
    if (this != &otra) {
        copiarTodo(otra);
    }
    return *this;
}

bool Receta::operator==(const Receta& other) const {
    return (id == other.id) &&
        (nombre == other.nombre) &&
        (preparacion == other.preparacion) &&
        (rutaImagen == other.rutaImagen) &&
        (categoria == other.categoria) &&
        (autor == other.autor) &&
        (tiempoPreparacion == other.tiempoPreparacion);
}

bool Receta::operator!=(const Receta& other) const {
    return !(*this == other);
}

bool Receta::operator<(const Receta& other) const {
    return id < other.id;
}

```

```

bool Receta::operator>(const Receta& other) const {
    return other < *this;
}

bool Receta::operator<=(const Receta& other) const {
    return !(*this > other);
}

bool Receta::operator>=(const Receta& other) const {
    return !(*this < other);
}

QTextStream& operator<<(QTextStream& os, const Receta& receta) {
    os << receta.getNombre() << "|"
        << receta.getCategoriaToInt() << "|"
        << receta.getRutaImagen() << "|" //
        << receta.getTiempoPreparacion() << "|"
        << receta.getAutor().getNombre() << "|"
        << receta.getAutor().getApellido() << "|"
        << receta.getPreparacion() << "*";
    return os;
}

QTextStream& operator>>(QTextStream& is, Receta& receta) {
    QString nombre, rutaImagen, autorNombre, autorApellido, preparacion;
    int tiempoPreparacion, categoria;

    nombre = is.readLine('|');
    is >> categoria;
    is.skipWhiteSpace();
    rutaImagen = is.readLine('|');
    is >> tiempoPreparacion;
    is.skipWhiteSpace();
    autorNombre = is.readLine('|');
    autorApellido = is.readLine('|');
    preparacion = is.readLine();

    receta.setNombre(nombre);
    receta.setCategoria(static_cast<Receta::Categoria>(categoria));
    receta.setRutaImagen(rutaImagen);
    receta.setTiempoPreparacion(tiempoPreparacion);

    Nombre autor;
    autor.setNombre(autorNombre);

```

```

        autor.setApellido(autorApellido);
        receta.setAutor(autor);

        receta.setPreparacion(preparacion);

        return is;
    }

QString Receta::getCategoriaToQString() const{
    switch (categoria) {
        case Categoria::DESAYUNO:    return "Desayuno";
        case Categoria::COMIDA:       return "Comida";
        case Categoria::CENA:         return "Cena";
        case Categoria::NAVIDEÑO:     return "Navideño";
        default:                      return "Desconocida";
    }
}

int Receta::getCategoriaToInt() const{
    if (categoria >= Categoria::DESAYUNO && categoria <=
        Categoria::NAVIDEÑO) {
        return static_cast<int>(categoria);
    }

    return -1;
}

int Receta::compararPorNombre(const Receta & a, const Receta & b){
    return QString::compare(a.getNombre(), b.getNombre(),
        Qt::CaseInsensitive);
}

int Receta::compararPorTiempoPreparacion(const Receta & a, const Receta &
b){
    return a.getTiempoPreparacion()-b.getTiempoPreparacion();
}

```

Clase Ingrediente:

```

#ifndef INGREDIENTE_HPP
#define INGREDIENTE_HPP

#include <QString>
#include <QTextStream>

```

```

#include <QString>

class Ingrediente{
public:
    enum class UnidadMedida {
        GRAMOS,
        KILOGRAMOS,
        MILILITROS,
        LITROS,
        CUCHARADA,
        CUCHARADITA,
        TAZA,
        PIZCA,
        ONZA,
        LIBRA,
        LATA
    };

private:
    QString nombre;
    float cantidad;
    UnidadMedida unidad;

public:
    Ingrediente();
    Ingrediente(const Ingrediente&);
    Ingrediente(const QString&,const float&,const UnidadMedida&);

    void copiarTodo(const Ingrediente&);

    void setNombre(const QString&);
    void setCantidad(const float&);
    void setUnidad(const UnidadMedida&);

    QString getNombre() const;
    float getCantidad() const;
    UnidadMedida getUnidad() const;

    Ingrediente& operator=(const Ingrediente&);

    bool operator==(const Ingrediente& other) const;
    bool operator!=(const Ingrediente& other) const;
    bool operator<(const Ingrediente& other) const;
    bool operator>(const Ingrediente& other) const;

```

```

    bool operator<=(const Ingrediente& other) const;
    bool operator>=(const Ingrediente& other) const;

    friend QTextStream& operator<<(QTextStream& , const Ingrediente& );
    friend QTextStream& operator>>(QTextStream& , Ingrediente& e);

    QString unidadMedidaToQString() const;
    QString toQString();
    int unidadMedidaToInt() const;
};

#endif // INGREDIENTE_HPP

#include "ingrediente.hpp"

Ingrediente::Ingrediente() {}

Ingrediente::Ingrediente(const Ingrediente& otro){
    copiarTodo(otro);
}

Ingrediente::Ingrediente(const QString& nombre,const float& cantidad,const
UnidadMedida& unidad){
    this->nombre=nombre;
    this->cantidad=cantidad;
    this->unidad=unidad;
}

void Ingrediente::copiarTodo(const Ingrediente& otro){
    this->nombre=otro.nombre;
    this->cantidad=otro.cantidad;
    this->unidad=otro.unidad;
}

void Ingrediente::setNombre(const QString & nuevoNombre){
    nombre=nuevoNombre;
}

void Ingrediente::setCantidad(const float & nuevaCantidad){
    cantidad=nuevaCantidad;
}

void Ingrediente::setUnidad(const UnidadMedida & nuevaUnidad){
    unidad=nuevaUnidad;
}

```

```

QString Ingrediente::getNombre() const{
    return nombre;
}

float Ingrediente::getCantidad() const{
    return cantidad;
}

Ingrediente::UnidadMedida Ingrediente::getUnidad() const{
    return unidad;
}

Ingrediente& Ingrediente::operator=(const Ingrediente& otro){
    copiarTodo(otro);

    return *this;
}

bool Ingrediente::operator==(const Ingrediente& other) const {
    return (nombre == other.nombre) &&
           (cantidad == other.cantidad) &&
           (unidad == other.unidad);
}

bool Ingrediente::operator!=(const Ingrediente& other) const {
    return !(*this == other);
}

bool Ingrediente::operator<(const Ingrediente& other) const {
    return nombre < other.nombre;
}

bool Ingrediente::operator>(const Ingrediente& other) const {
    return other < *this;
}

bool Ingrediente::operator<=(const Ingrediente& other) const {
    return !(*this > other);
}

bool Ingrediente::operator>=(const Ingrediente& other) const {
    return !(*this < other);
}

```



```

QString Ingrediente::toQString(){
    return QString::number(cantidad, 'f', 2) + " " +
    unidadMedidaToQString()+" de " +nombre;
}

int Ingrediente::unidadMedidaToint() const{
    if (unidad >= UnidadMedida::GRAMOS && unidad <= UnidadMedida::LATA) {
        return static_cast<int>(unidad);
    }

    return -1;
}

```

Clase Nombre:

```

#ifndef NOMBRE_HPP
#define NOMBRE_HPP
#include <QString>

class Nombre {
private:
    QString nombre;
    QString apellido;

public:
    Nombre();
    Nombre(const Nombre&);
    Nombre(const QString&, const QString&);

    void copiarTodo(const Nombre&);

    void setNombre(const QString&);
    void setApellido(const QString&);

    QString getNombre() const;
    QString getApellido() const;

    Nombre& operator=(const Nombre&);

    bool operator==(const Nombre&) const;
    bool operator!=(const Nombre&) const;
    bool operator<(const Nombre&) const;
    bool operator<=(const Nombre&) const;
    bool operator>(const Nombre&) const;
    bool operator>=(const Nombre&) const;

```

```

    QString toQString() const;
};

#endif // NOMBRE_HPP

#include "nombre.hpp"

Nombre::Nombre() {}

Nombre::Nombre(const Nombre& otro) {
    copiarTodo(otro);
}

Nombre::Nombre(const QString& nombre, const QString& apellido) {
    this->nombre = nombre;
    this->apellido = apellido;
}

void Nombre::copiarTodo(const Nombre& otro) {
    this->nombre = otro.nombre;
    this->apellido = otro.apellido;
}

void Nombre::setNombre(const QString& nuevoNombre) {
    nombre = nuevoNombre;
}

void Nombre::setApellido(const QString& nuevoApellido) {
    apellido = nuevoApellido;
}

QString Nombre::getNombre() const {
    return nombre;
}

QString Nombre::getApellido() const {
    return apellido;
}

Nombre& Nombre::operator=(const Nombre& otro) {
    copiarTodo(otro);

    return *this;
}

```

```

bool Nombre::operator==(const Nombre& otro) const {
    return (nombre == otro.nombre && apellido == otro.apellido);
}

bool Nombre::operator!=(const Nombre& otro) const {
    return !(*this == otro);
}

bool Nombre::operator<(const Nombre& otro) const {
    if (apellido == otro.apellido) {
        return nombre < otro.nombre;
    }
    return apellido < otro.apellido;
}

bool Nombre::operator<=(const Nombre& otro) const {
    return (*this < otro || *this == otro);
}

bool Nombre::operator>(const Nombre& otro) const {
    return !(*this <= otro);
}

bool Nombre::operator>=(const Nombre& otro) const {
    return !(*this < otro);
}

QString Nombre::toString() const {
    return nombre + " " + apellido;
}

```

Clase ListaRecetas:

```

#ifndef LISTARECETAS_H
#define LISTARECETAS_H

#include "receta.hpp"
#include "exception.hpp"

class ListaRecetas{
private:
    Receta** recetas;
    int ultimaPosicion;
    int capacidad;

```

```

    void copiarTodo(const ListaRecetas& );
    void aumentarCapacidad();

public:
    ListaRecetas();
    ListaRecetas(const ListaRecetas& );
    ~ListaRecetas();

    ListaRecetas& operator=(const ListaRecetas& );

    void agregarReceta(Receta* );
    void eliminarReceta(int& );
    Receta* recuperarReceta(const int&) const;
    void anular();

    bool vacia() const;
    bool llena() const;
    bool posicionValida(const int& ) const;
    int size() const;

    int getPrimeraPosicion() const;
    int getUltimaPosicion() const;
    int getPosicionPrevia(const int&) const;
    int getSiguientePosicion(const int& ) const;
    int localiza(Receta*);

    int busquedaLineal(const Receta* , int (*cmp)(const Receta&, const
Receta&)) const;
    int busquedaBinaria(const Receta* , int (*cmp)(const Receta&, const
Receta&)) const;

    void quickSort(int (*cmp)(const Receta&, const Receta&));
    void quickSort(const int& ,const int&,int (*cmp)(const Receta&, const
Receta&));

    void swap(const int&, const int&);
};

#endif // LISTARECETAS_H

#include "listaRecetas.hpp"

```

```

ListaRecetas::ListaRecetas() : recetas(nullptr), ultimaPosicion(-1),
capacidad(50) {
    recetas = new Receta*[capacidad];
}

ListaRecetas::ListaRecetas(const ListaRecetas& otra){
    copiarTodo(otra);
}

ListaRecetas::~~ListaRecetas(){
    anular();
    delete [] recetas;
}

void ListaRecetas::copiarTodo(const ListaRecetas & otra){
    this->ultimaPosicion = otra.ultimaPosicion;
    this->capacidad = otra.capacidad;
    this->recetas = new Receta*[capacidad];

    for (int i = 0; i <= ultimaPosicion; i++) {
        this->recetas[i] = new Receta(*otra.recetas[i]);
    }
}

ListaRecetas& ListaRecetas::operator=(const ListaRecetas& otra){
    copiarTodo(otra);
    return *this;
}

void ListaRecetas::aumentarCapacidad(){
    capacidad *= 2;
    Receta** nuevoArreglo = new Receta*[capacidad];

    for (int i = 0; i <= ultimaPosicion; i++) {
        nuevoArreglo[i] = recetas[i];
    }

    delete[] recetas;
    recetas = nuevoArreglo;
}

bool ListaRecetas::vacia() const {
    return ultimaPosicion == -1;
}

```

```

bool ListaRecetas::llena() const {
    return ultimaPosicion == capacidad - 1;
}

int ListaRecetas::size() const {
    return ultimaPosicion + 1;
}

void ListaRecetas::agregarReceta(Receta *receta) {
    if (llena()) {
        aumentarCapacidad();
    }

    ultimaPosicion++;
    recetas[ultimaPosicion] = receta;
}

void ListaRecetas::eliminarReceta(int& posicion) {
    if (vacía()) {
        throw Exception("La lista está vacía, no hay recetas para
eliminar.");
    }

    if (!posicionValida(posicion)) {
        throw Exception("Posicion invalida para eliminar la receta.");
    }

    delete recetas[posicion];

    for (int i = posicion; i < ultimaPosicion; i++) {
        recetas[i] = recetas[i + 1];
    }

    recetas[ultimaPosicion] = nullptr;
    ultimaPosicion--;
}

bool ListaRecetas::posicionValida(const int& posicion) const {
    return posicion >= 0 && posicion <= ultimaPosicion;
}

int ListaRecetas::getPrimeraPosicion() const {
    if (vacía()) {
        return -1;
    }
}

```

```

        return 0;
    }

    int ListaRecetas::getUltimaPosicion() const {
        return ultimaPosicion;
    }

    int ListaRecetas::getPosicionPrevia(const int& posicion) const {
        if (posicion == getPrimeraPosicion() || !posicionValida(posicion)) {
            return -1;
        }
        return posicion - 1;
    }

    int ListaRecetas::getSiguientePosicion(const int& posicion) const {
        if (posicion == getUltimaPosicion() || !posicionValida(posicion)) {
            return -1;
        }
        return posicion + 1;
    }

    int ListaRecetas::localiza(Receta * receta){
        for (int i = 0; i <= ultimaPosicion; ++i) {
            if (recetas[i] == receta) {
                return i;
            }
        }
        return -1;
    }

    Receta* ListaRecetas::recuperarReceta(const int& posicion) const {
        if (!posicionValida(posicion)) {
            throw Exception("Posicion invalida para recuperar la receta.");
        }
        return recetas[posicion];
    }

    int ListaRecetas::busquedaLineal(const Receta* objetivo, int (*cmp)(const
Receta&, const Receta&)) const {
        for (int i = 0; i <= ultimaPosicion; i++) {
            if (cmp(*objetivo, *recetas[i]) == 0) {
                return i;
            }
        }
        return -1;
    }

```

```

}

int ListaRecetas::busquedaBinaria(const Receta* objetivo, int (*cmp)(const
Receta&, const Receta&)) const {
    if(vacia()){return -1;}

    int izquierda = 0;
    int derecha = ultimaPosicion;

    while (izquierda <= derecha) {
        int mitad = (izquierda + derecha) / 2;
        int resultado = cmp(*objetivo, *recetas[mitad]);

        if (resultado == 0) {
            return mitad;
        }
        else if (resultado < 0) {
            derecha = mitad - 1;
        }
        else {
            izquierda = mitad + 1;
        }
    }
    return -1;
}

void ListaRecetas::quickSort(int (*cmp)(const Receta &, const Receta &)){
    quickSort(0, ultimaPosicion, cmp);
}

void ListaRecetas::quickSort(const int& left, const int & right, int
(*cmp)(const Receta &, const Receta &)){
    if (left >= right) {
        return;
    }

    int i = left;
    int j = right - 1;
    Receta pivot = *recetas[right]; // El pivote está en right

    while (i <= j) {
        while (i <= j && cmp(*recetas[i], pivot) <= 0) {
            i++;
        }
        while (i <= j && cmp(*recetas[j], pivot) >= 0) {

```



```

        j--;
    }
    if (i < j) {
        swap(i, j);
    }
}

swap(i, right);

// Divide y vencerás
quickSort(left, i - 1, cmp);
quickSort(i + 1, right, cmp);
}

void ListaRecetas::swap(const int& i, const int& j){
    Receta *aux=recetas[i];
    recetas[i]=recetas[j];
    recetas[j] = aux;
}

void ListaRecetas::anular() {
    for (int i = 0; i <= ultimaPosicion; i++) {
        delete recetas[i];
    }
    ultimaPosicion = -1;
}

```

Clase ListaIngredientes:

```

#ifndef LISTAINGREDIENTES_H
#define LISTAINGREDIENTES_H

#include "ingrediente.hpp"
#include "exception.hpp"

class ListaIngredientes{
private:
    Ingrediente **ingredientes;
    int ultimaPosicion;
    int capacidad;

    void copiarTodo(const ListaIngredientes&);
    void aumentarCapacidad();

```

```

public:
    ListaIngredientes();
    ListaIngredientes(const ListaIngredientes&);
    ~ListaIngredientes();

    ListaIngredientes& operator=(const ListaIngredientes&);

    bool vacia();
    bool llena();
    bool posicionValida(const int&);

    void insertarIngrediente(Ingrediente*);
    void eliminarIngrediente(int&);

    int getPrimeraPosicion();
    int getUltimaPosicion();
    int getPosicionPrevia(const int&);
    int getSiguientePosicion(const int&);

    Ingrediente* recuperarIngrediente(const int&);

    int busquedaLineal(const Ingrediente*,int (*cmp)(const Ingrediente&,
const Ingrediente&)) const;
    int busquedaBinaria(const Ingrediente*,int (*cmp)(const Ingrediente&,
const Ingrediente&)) const;
    int buscarPosicionOrdenada(const Ingrediente*) const;

    QString toQString() ;
    void anular();

    int size() const;
};

#endif // LISTAINGREDIENTES_H

#include "listaIngredientes.hpp"

ListaIngredientes::ListaIngredientes() : ingredientes(nullptr),
ultimaPosicion(-1), capacidad(15) {
    // Inicialización
    ingredientes = new Ingrediente*[capacidad];
}

```

```

ListaIngredientes::ListaIngredientes(const ListaIngredientes& otra){
    copiarTodo(otra);
}

ListaIngredientes::~~ListaIngredientes(){
    anular();

    delete [] ingredientes;
}

void ListaIngredientes::copiarTodo(const ListaIngredientes & otra){

    this->ultimaPosicion=otra.ultimaPosicion;
    this->capacidad=otra.capacidad;

    this->ingredientes = new Ingrediente*[capacidad];

    for(int i=0;i<=ultimaPosicion;i++){
        this->ingredientes[i]=new Ingrediente(*otra.ingredientes[i]);
    }
}

ListaIngredientes& ListaIngredientes:: operator=(const ListaIngredientes&
otra){
    copiarTodo(otra);

    return *this;
}

void ListaIngredientes::aumentarCapacidad(){
    capacidad *= 2;

    Ingrediente** nuevoArreglo = new Ingrediente*[capacidad];

    for (int i = 0; i <=ultimaPosicion; i++) {
        nuevoArreglo[i] = ingredientes[i];
    }

    delete[] ingredientes;
    ingredientes = nuevoArreglo;
}

bool ListaIngredientes::vacia(){
    return ultimaPosicion==-1;
}

```

```

bool ListaIngredientes::llena(){
    return ultimaPosicion == capacidad-1;
}

bool ListaIngredientes::posicionValida(const int& posicion){
    return posicion >= 0 && posicion <= ultimaPosicion;
}

int ListaIngredientes::size() const {
    return ultimaPosicion + 1;
}

void ListaIngredientes::insertarIngrediente(Ingrediente *ingrediente) {
    if (llena()) {
        aumentarCapacidad();
    }

    int posicion = buscarPosicionOrdenada(ingrediente);

    int i = ultimaPosicion;

    while (i >= posicion) {
        ingredientes[i + 1] = ingredientes[i];
        i--;
    }

    ingredientes[posicion] =ingrediente;
    ultimaPosicion++;
}

void ListaIngredientes::eliminarIngrediente(int & posicion){
    if (vacía()) {
        throw Exception("La lista está vacía, no hay ingredientes para
eliminar.");
    }

    if (!posicionValida(posicion)) {
        throw Exception("Posicion invalida para eliminar el ingrediente.");
    }

    delete ingredientes[posicion];

    int i = posicion;
    while (i < ultimaPosicion) {

```

```

        ingredientes[i] = ingredientes[i + 1];
        i++;
    }

    ingredientes[ultimaPosicion] = nullptr;
    ultimaPosicion--;
}

int ListaIngredientes::getPrimeraPosicion(){
    if (vacía()) {
        return -1;
    }

    return 0;
}

int ListaIngredientes::getUltimaPosicion(){
    return ultimaPosicion;
}

int ListaIngredientes::getPosicionPrevia(const int & posicion){
    if (posicion == getPrimeraPosicion() || !posicionValida(posicion)) {
        return -1;
    }

    return posicion - 1;
}

int ListaIngredientes::getSiguientePosicion(const int & posicion){
    if (posicion == getUltimaPosicion() || !posicionValida(posicion)) {
        return -1;
    }

    return posicion + 1;
}

Ingrediente *ListaIngredientes::recuperarIngrediente(const int & posicion){
    if (!posicionValida(posicion)) {
        throw Exception("Posicion invalida para recuperar el elemento.");
    }

    return ingredientes[posicion];
}

```

```

int ListaIngredientes::busquedaLineal(const Ingrediente * objetivo, int
(*cmp)(const Ingrediente &, const Ingrediente &)) const{

    for(int i=0;i<=ultimaPosicion;i++){
        if(cmp(*objetivo,*ingredientes[i])==0){
            return i;
        }
    }

    return -1;
}

int ListaIngredientes::busquedaBinaria(const Ingrediente * objetivo, int
(*cmp)(const Ingrediente &, const Ingrediente &)) const{
    int izquierda=0;
    int derecha=ultimaPosicion;

    while(izquierda<=derecha){
        int mitad=(izquierda+derecha)/2;
        int resultado=cmp(*objetivo,*ingredientes[mitad]);

        if(resultado==0){
            return mitad;
        }
        else if(resultado<0){
            derecha=mitad-1;
        }
        else{
            derecha=mitad+1;
        }
    }

    return -1;
}

int ListaIngredientes::buscarPosicionOrdenada(const Ingrediente *
ingrediente) const{
    int posicion=0;

    while(posicion<=ultimaPosicion && (ingrediente->getNombre() >
ingredientes[posicion]->getNombre())){
        posicion++;
    }

    return posicion;
}

```

```

}

QString ListaIngredientes::toQString() {
    QString resultado;

    resultado+="---INGREDIENTES---\n\n";
    for(int i=0;i<=ultimaPosicion;i++){
        resultado+=QString::number(i+1)+". "+ingredientes[i]-
>toQString()+"\n";

        if (i < ultimaPosicion) {
            resultado += "-----\n";
        }
    }

    return resultado;
}

void ListaIngredientes::anular(){

    for(int i=0;i<=ultimaPosicion;i++){
        delete ingredientes[i];
    }

    ultimaPosicion=-1;
}

```

Clase IngredientWidget:

```

#ifndef INGREDIENTWIDGET_H
#define INGREDIENTWIDGET_H

#include <QWidget>
#include <QLineEdit>
#include <QPushButton>
#include <QComboBox>
#include <QLabel>
#include <QVBoxLayout>
#include <QFile>
#include <QUiLoader>
#include <QPixmap>
#include <QDoubleValidator>
#include "ingrediente.hpp"

```

```

class IngredientWidget : public QWidget
{
    Q_OBJECT
public:
    explicit IngredientWidget(QWidget *parent = nullptr);
    explicit IngredientWidget(const Ingrediente&, QWidget *parent =
nullptr);

    void setIngredient(const Ingrediente&);

    QString getNombre() const;
    double getCantidad() const;
    Ingrediente::UnidadMedida getUnidadMedida();

signals:
    void deleteClicked();

private:
    QLabel *itemIconLabel;
    QLineEdit *lineEditNameIngredient;
    QLineEdit *quantityLineEdit;
    QComboBox *comboBoxUnit;
    QPushButton *deleteButton;

    void setupUi();
};

#endif // INGREDIENTWIDGET_H

#include "ingredientWidget.hpp"

IngredientWidget::IngredientWidget(QWidget* parent) : QWidget{parent} {
    setupUi();
}

IngredientWidget::IngredientWidget(const Ingrediente& ingredient,
                                   QWidget* parent)
    : QWidget{parent} {
    setupUi();
    setIngredient(ingredient);
}

```



```

void IngredientWidget::setupUi() {
    setMinimumSize(595, 66);

    QFile file(":/ingredientWidget.ui");
    file.open(QFile::ReadOnly);
    QUiLoader loader;
    QWidget* widget = loader.load(&file, this);
    file.close();

    // Configurar el layout de IngredientCard
    QVBoxLayout* layout = new QVBoxLayout(this);
    layout->addWidget(widget);

    // Acceder a los widgets
    itemIconLabel = widget->findChild<QLabel*>("itemIcon");
    lineEditNameIngredient =
        widget->findChild<QLineEdit*>("lineEditNameIngredient");
    quantityLineEdit = widget->findChild<QLineEdit*>("quantitylineEdit");
    comboBoxUnit = widget->findChild<QComboBox*>("comboBoxUnit");
    deleteButton = widget->findChild<QPushButton*>("deleteIngredientButton");

    deleteButton->setCursor(Qt::PointingHandCursor);

    QDoubleValidator* validator = new QDoubleValidator(0.01, 9999.99, 2,
this);
    validator->setNotation(QDoubleValidator::StandardNotation);
    quantityLineEdit->setValidator(validator);

    connect(deleteButton, &QPushButton::clicked, this,
        &IngredientWidget::deleteClicked);
}

void IngredientWidget::setIngredient(const Ingrediente& ingredient) {
    lineEditNameIngredient->setText(ingredient.getNombre());
    quantityLineEdit->setText(QString::number(ingredient.getCantidad(), 'f',
2));
    int unitIndex = ingredient.unidadMedidaToint();
    if (unitIndex != -1) {
        comboBoxUnit->setCurrentIndex(unitIndex);
    }
}

QString IngredientWidget::getNombre() const {
    return lineEditNameIngredient->text();
}

```

```

double IngredientWidget::getCantidad() const {
    return quantityLineEdit->text().toDouble();
}

Ingrediente::UnidadMedida IngredientWidget::getUnidadMedida() {
    return static_cast<Ingrediente::UnidadMedida>(comboBoxUnit-
>currentIndex());
}

```

Clase RecipeCard:

```

#ifndef RECIPECARD_H
#define RECIPECARD_H

#include <QFile>
#include <QLabel>
#include <QMouseEvent>
#include <QPixmap>
#include <QPushButton>
#include <QPainter>
#include <QVBoxLayout>
#include <QWidget>
#include "receta.hpp"

class RecipeCard : public QWidget {
    Q_OBJECT

public:
    explicit RecipeCard(Receta*, QWidget* parent = nullptr);
    void setReceta(const Receta& receta);
    void actualizarVista();
    void setRecetaAsociada(Receta*);

    Receta* getRecetaAsociada();

signals:
    void modifyClicked();
    void deleteClicked();
    void imageClicked();

protected:
    void mousePressEvent(QMouseEvent* event) override;

```

```

private:
    QLabel* titleLabel;
    QLabel* categoryLabel;
    QLabel* timeLabel;
    QLabel* imageLabel;
    QPushButton* deleteButton;
    QPushButton* modifyButton;
    Receta* receta;

    void setupUi();
};

#endif // RECIPECARD_H

#include "recipeCard.hpp"

RecipeCard::RecipeCard(Receta* receta, QWidget* parent) : QWidget{parent} {
    setMinimumSize(321, 265);
    setupUi();
    this->receta = receta;
    setReceta(*receta);

    connect(modifyButton, &QPushButton::clicked, this,
            &RecipeCard::modifyClicked);
    connect(deleteButton, &QPushButton::clicked, this,
            &RecipeCard::deleteClicked);

    modifyButton->setCursor(Qt::PointingHandCursor);
    deleteButton->setCursor(Qt::PointingHandCursor);
    imageLabel->setCursor(Qt::PointingHandCursor);
}

void RecipeCard::setupUi() {
    QFile file(":/recipeCard.ui");
    file.open(QFile::ReadOnly);
    QUiLoader loader;
    QWidget* widget = loader.load(&file, this);
    file.close();

    QVBoxLayout* layout = new QVBoxLayout(this);
    layout->addWidget(widget);

    titleLabel = widget->findChild<QLabel*>("tittle");
    categoryLabel = widget->findChild<QLabel*>("category");
    timeLabel = widget->findChild<QLabel*>("time");
}

```

```

        imageLabel = widget->findChild<QLabel*>("image");
        deleteButton = widget->findChild<QPushButton*>("deleteRecipeButton");
        modifyButton = widget->findChild<QPushButton*>("modifyRecipeButton");
    }

    void RecipeCard::setReceta(const Receta& receta) {
        QString nombreReceta = receta.getNombre();
        if (nombreReceta.length() > 19) {
            titleLabel->setText(nombreReceta.left(16) + "...");
        } else {
            titleLabel->setText(nombreReceta);
        }

        categoryLabel->setText(receta.getCategoriaToString());
        timeLabel->setText(QString::number(receta.getTiempoPreparacion()) + "
min");
        QPixmap pixmap(receta.getRutaImagen());
        if (!pixmap.isNull()) {
            imageLabel->setScaledContents(true);
            imageLabel->setPixmap(pixmap);
        } else {
            imageLabel->setText("Sin imagen");
        }
    }

    void RecipeCard::actualizarVista() {
        setReceta(*receta);
    }

    void RecipeCard::setRecetaAsociada(Receta* receta) {
        this->receta = receta;
    }

    Receta* RecipeCard::getRecetaAsociada() {
        return receta;
    }

    void RecipeCard::mousePressEvent(QMouseEvent* event) {
        if (event->button() == Qt::LeftButton &&
            imageLabel->geometry().contains(event->pos())) {
            emit imageClicked();
        }
        QWidget::mousePressEvent(event);
    }

```

Clase MainWindow (Interfaz grafica):

```
#ifndef MAINWINDOW_HPP
#define MAINWINDOW_HPP

#include <QMainWindow>
#include <QStackedWidget>
#include <QPushButton>
#include <QVBoxLayout>
#include <QWidget>
#include <QMessageBox>
#include <QIntValidator>
#include <QFileDialog>
#include <QTimer>
#include <QJsonDocument>
#include <QJsonObject>
#include <QJsonArray>
#include <iostream>
#include <sstream>
#include <fstream>
#include <QBuffer>

#include "ingredientWidget.hpp"
#include "ingrediente.hpp"
#include "listaRecetas.hpp"
#include "receta.hpp"
#include "recipeCard.hpp"

QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE

class MainWindow : public QMainWindow {
    Q_OBJECT

public:
    MainWindow(QWidget* parent = nullptr);
    ~MainWindow();

private:
    Ui::MainWindow* ui;
```

```

int lastId;
ListaRecetas listaRecetas;
QVBoxLayout* ingredientsLayout;
QGridLayout* recipesCardsLayout;
int row = 0;
int col = 0;
const int maxColumns = 3;

const int homePage = 0;
const int recipesPage = 1;
const int addRecipePage = 2;
const int viewRecipePage = 3;

// Inicializacion
void initializeImagesDirectory();
void setupRecipesCardsSection();
void setupIngredientsSection();
void setupValidators();
void setupConnections();

// Manejo de las recetas
void agregarReceta();
void modificarReceta();
void agregarIngredientesReceta(Receta*);

// Manejo de widgets para mostrar recetas e ingredientes en la interfaz
void clearRecipeCards();
void addRecipeCardWidget(Receta*);
void addIngredientWidget();
void clearIngredientWidgets();
void actualizarRecipeCardWidgets();
void filtrarPorCategoria(QString);

// Utilidades
void desactivarBotonesNavBar();
void activarBotonesNavBar();
void setCursorPointerForAllButtons();

bool validarCamposAddRecipePage();
void limpiarCamposAddRecipePage();
void llenarCamposAddRecipePage(Receta&);

void llenarCamposViewRecipePage(Receta*);

// Cargar y guardar las recetas al disco

```

```

    void guardarRecetas();
    void cargarRecetas();
protected:
    void closeEvent(QCloseEvent *event) override;

    // Manejo de eventos de la interfaz
public slots:
    void onModifyRecipeClicked(RecipeCard*);
    void onDeleteRecipeClicked(RecipeCard*);
    void onViewRecipe(RecipeCard*);
    void onSortChanged(int);
    void onFilterChanged(int);
    void onSearchButtonClicked();
    void onTextChanged(const QString&);
    void onClearButtonClicked();
    void onReturnButtonClicked();

private slots:
    void onHomeButtonClicked();
    void onRecipesButtonClicked();
    void onAddRecipeButtonClicked();
    void onDeleteAllRecipesButtonClicked();
    void onCancelButtonClicked();
    void onConfirmButtonClicked();
    void onUploadPhotoButtonClicked();
    void onDeleteAllIngredientsButtonClicked();
};
#endif // MAINWINDOW_HPP

#include "mainwindow.hpp"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget* parent)
    : QMainWindow(parent), ui(new Ui::MainWindow) {
    ui->setupUi(this);
    this->setWindowTitle("Recetario Digital");
    setFixedSize(1080, 760);
    ui->homeButton->setChecked(true);
    this->lastId = 0;
    ui->messageWidget->setVisible(false);

    initializeImagesDirectory();
    setupIngredientsSection();
    setupRecipesCardsSection();

```

```

        setCursorPointerForAllButtons();
        setupValidators();
        setupConnections();

        ui->pages->setCurrentIndex(homePage);
    }

MainWindow::~MainWindow() {
    delete ui;
}

void MainWindow::initializeImagesDirectory() {
    QString imagesDir = QDir::currentPath() + "/imagenes";
    QDir dir(imagesDir);
    if (!dir.exists()) {
        dir.mkpath(".");
        qDebug() << "Carpeta de imágenes creada en:" << imagesDir;
    } else {
        qDebug() << "La carpeta de imágenes ya existe en:" << imagesDir;
    }
}

void MainWindow::setupRecipesCardsSection() {
    QWidget* recipesCardsContainer = new QWidget();
    recipesCardsLayout = new QGridLayout(recipesCardsContainer);
    ui->recipesArea->setWidget(recipesCardsContainer);
    ui->recipesArea->setWidgetResizable(true);
}

void MainWindow::setupIngredientsSection() {
    QWidget* ingredientsContainer = new QWidget();
    ingredientsLayout = new QVBoxLayout(ingredientsContainer);
    ingredientsContainer->setLayout(ingredientsLayout);
    ui->ingredientsArea->setWidget(ingredientsContainer);
    ui->ingredientsArea->setWidgetResizable(true);

    connect(ui->addIngredientButton, &QPushButton::clicked, this,
            [this]() { addIngredientWidget(); });
}

void MainWindow::setupValidators() {
    QIntValidator* validator =
        new QIntValidator(1, std::numeric_limits<int>::max(), this);
    ui->lineEditDuration->setValidator(validator);
}

```



```

void MainWindow::setupConnections() {
    connect(ui->homeButton, &QPushButton::clicked, this,
            &MainWindow::onHomeButtonClicked);
    connect(ui->recipesButton, &QPushButton::clicked, this,
            &MainWindow::onRecipesButtonClicked);

    connect(ui->addRecipeButton, &QPushButton::clicked, this,
            &MainWindow::onAddRecipeButtonClicked);
    connect(ui->deleteAllRecipesButton, &QPushButton::clicked, this,
            &MainWindow::onDeleteAllRecipesButtonClicked);

    connect(ui->cancelButton, &QPushButton::clicked, this,
            &MainWindow::onCancelButtonClicked);
    connect(ui->confirmButton, &QPushButton::clicked, this,
            &MainWindow::onConfirmButtonClicked);

    connect(ui->uploadPhotoButton, &QPushButton::clicked, this,
            &MainWindow::onUploadPhotoButtonClicked);
    connect(ui->deleteAllIngredientsButton, &QPushButton::clicked, this,
            &MainWindow::onDeleteAllIngredientsButtonClicked);

    ui->comboBoxSort->addItem("--Selecciona--");
    ui->comboBoxSort->setItemData(0, true, Qt::UserRole - 1);

    // Agregar las opciones de ordenamiento
    ui->comboBoxSort->addItem("Nombre");
    ui->comboBoxSort->addItem("Tiempo de Preparacion");
    connect(ui->comboBoxSort, &QComboBox::currentIndexChanged, this,
            &MainWindow::onSortChanged);
    ui->comboBoxSort->setCurrentIndex(0);

    // Agregar las opciones de filtrado
    ui->comboBoxFilter->addItem("Todas");
    ui->comboBoxFilter->addItem("Desayuno");
    ui->comboBoxFilter->addItem("Comida");
    ui->comboBoxFilter->addItem("Cena");
    ui->comboBoxFilter->addItem("Navideño");
    connect(ui->comboBoxFilter, &QComboBox::currentIndexChanged, this,
            &MainWindow::onFilterChanged);
    ui->comboBoxFilter->setCurrentIndex(0);

    connect(ui->searchRecipeLineEdit, &QLineEdit::returnPressed, this,
            &MainWindow::onSearchButtonClicked);

```

```

connect(ui->searchRecipeButton, &QPushButton::clicked, this,
        &MainWindow::onSearchButtonClicked);

connect(ui->searchRecipeLineEdit, &QLineEdit::textChanged, this,
        &MainWindow::onTextChanged);
connect(ui->clearSearchLineEditButton, &QPushButton::clicked, this,
        &MainWindow::onClearButtonClicked);

ui->clearSearchLineEditButton->setVisible(false);

connect(ui->exploreRecipesButton, &QPushButton::clicked, this,
        &MainWindow::onRecipesButtonClicked);
connect(ui->returnButton, &QPushButton::clicked, this,
        &MainWindow::onReturnButtonClicked);
}

void MainWindow::agregarReceta() {
    Receta* nuevaReceta = new Receta;

    if (nuevaReceta == nullptr) {
        QMessageBox::critical(
            this, "Error",
            "Error de memoria: no se pudo reservar memoria para la receta.",
            QMessageBox::Ok);
    }

    nuevaReceta->setNombre(ui->lineEditName->text());
    nuevaReceta->setCategoria(
        static_cast<Receta::Categoria>(ui->comboBoxCategory->currentIndex()));
    nuevaReceta->setTiempoPreparacion(ui->lineEditDuration->text().toInt());
    Nombre autorReceta(ui->lineEditAutorFirstName->text(),
        ui->lineEditAutorLastName->text());
    nuevaReceta->setAutor(autorReceta);
    QString sourcePath = ui->uploadPhotoButton-
>property("imagePath").toString();
    QString imagesDir = QDir::currentPath() + "/imagenes";
    QString destinationPath =
        imagesDir + "/" + QFile::FileInfo(sourcePath).fileName(); // Nueva ruta

    // Copiar la imagen a la carpeta de imágenes
    if (QFile::copy(sourcePath, destinationPath)) {
        nuevaReceta->setRutaImagen(destinationPath);
        qDebug() << "Imagen copiada a:" << destinationPath;
    } else {
        qDebug() << "Error al copiar la imagen.";
    }
}

```

```

    }
    nuevaReceta->setPreparacion(ui->instructionsText->toPlainText());

    agregarIngredientesReceta(nuevaReceta);

    try {
        listaRecetas.agregarReceta(nuevaReceta);
    } catch (const std::exception& e) {
        QMessageBox::warning(this, "Error", e.what());
    }

    addRecipeCardWidget(nuevaReceta);

    qDebug() << listaRecetas.getUltimaPosicion();
    qDebug() << listaRecetas.size();
    qDebug() << "El tamaño del puntero a receta es: " << sizeof(nuevaReceta)
        << " bytes.";
    qDebug() << "El tamaño del objeto receta al que apunta es: "
        << sizeof(*nuevaReceta) << " bytes.";
}

void MainWindow::modificarReceta() {
    RecipeCard* recipeCard =
        ui->confirmButton->property("recipePointer").value<RecipeCard*>();
    Receta* receta = recipeCard->getRecetaAsociada();

    if (receta->getNombre() != ui->lineEditName->text()) {
        receta->setNombre(ui->lineEditName->text());
    }

    int nuevaCategoria = ui->comboBoxCategory->currentIndex();
    if (receta->getCategoriaToInt() != nuevaCategoria) {
        receta->setCategoria(static_cast<Receta::Categoria>(nuevaCategoria));
    }

    int nuevoTiempoPreparacion = ui->lineEditDuration->text().toInt();
    if (receta->getTiempoPreparacion() != nuevoTiempoPreparacion) {
        receta->setTiempoPreparacion(nuevoTiempoPreparacion);
    }

    Nombre nuevoAutor(ui->lineEditAutorFirstName->text(),
        ui->lineEditAutorLastName->text());
    if (receta->getAutor() != nuevoAutor) {
        receta->setAutor(nuevoAutor);
    }
}

```

```

// Verificar si se ha cambiado la imagen
QString sourcePath = ui->uploadPhotoButton-
>property("imagePath").toString();
QString currentImagePath = receta->getRutaImagen();

if (sourcePath != currentImagePath) {
    QString imagesDir = QDir::currentPath() + "/imagenes";
    QString destinationPath =
        imagesDir + "/" + QFile::FileInfo(sourcePath).fileName();

    // Eliminar la imagen antigua si existe
    if (!currentImagePath.isEmpty() && QFile::exists(currentImagePath)) {
        QFile::remove(currentImagePath); // Eliminar archivo
        qDebug() << "Imagen antigua eliminada:" << currentImagePath;
    }

    if (QFile::copy(sourcePath, destinationPath)) {
        receta->setRutaImagen(destinationPath);
        qDebug() << "Imagen copiada a:" << destinationPath;
    } else {
        qDebug() << "Error al copiar la nueva imagen.";
    }
}

if (receta->getPreparacion() != ui->instructionsText->toPlainText()) {
    receta->setPreparacion(ui->instructionsText->toPlainText());
}

receta->eliminarTodosLosIngredientes();
agregarIngredientesReceta(receta);

recipeCard->actualizarVista();
}

void MainWindow::agregarIngredientesReceta(Receta* receta) {
    for (int i = 0; i < ingredientsLayout->count(); i++) {
        QWidget* widget = ingredientsLayout->itemAt(i)->widget();

        IngredientWidget* ingredientWidget =
            qobject_cast<IngredientWidget*>(widget);
        if (ingredientWidget) {
            QString nombre = ingredientWidget->getNombre();
            float cantidad = ingredientWidget->getCantidad();

```

```

        Ingrediente::UnidadMedida unidad = ingredientWidget-
>getUnidadMedida();

        Ingrediente* nuevoIngrediente = new Ingrediente(nombre, cantidad,
unidad);
        receta->agregarIngrediente(nuevoIngrediente);

        qDebug() << "El tamaño del puntero a Ingrediente es: "
                << sizeof(nuevoIngrediente) << " bytes.";
        qDebug() << "El tamaño del objeto Ingrediente al que apunta es: "
                << sizeof(*nuevoIngrediente) << " bytes.";
    }
}
}

void MainWindow::clearRecipeCards() {
    while (QLayoutItem* item = recipesCardsLayout->takeAt(0)) {
        if (QWidget* widget = item->widget()) {
            widget->deleteLater();
        }
        delete item;
    }
}

void MainWindow::addRecipeCardWidget(Receta* receta) {
    // Crear una nueva RecipeCard con la receta
    RecipeCard* recetaCard = new RecipeCard(receta);

    recipesCardsLayout->addWidget(recetaCard, row, col);
    col++;
    if (col >= maxColumns) {
        col = 0;
        row++;
    }

    connect(recetaCard, &RecipeCard::modifyClicked, this,
            [this, recetaCard]() { onModifyRecipeClicked(recetaCard); });
    connect(recetaCard, &RecipeCard::deleteClicked, this,
            [this, recetaCard]() { onDeleteRecipeClicked(recetaCard); });
    connect(recetaCard, &RecipeCard::imageClicked, this,
            [this, recetaCard]() { onViewRecipe(recetaCard); });

    qDebug() << "El tamaño del puntero a recetaCard es: " <<
sizeof(recetaCard)
        << " bytes.";
}

```

```

    qDebug() << "El tamaño del objeto recetaCard al que apunta es: "
        << sizeof(*recetaCard) << " bytes.";
}

void MainWindow::addIngredientWidget() {
    IngredientWidget* newIngredientWidget = new IngredientWidget;
    connect(newIngredientWidget, &IngredientWidget::deleteClicked, this,
        [newIngredientWidget, this]() {
            ingredientsLayout->removeWidget(
                newIngredientWidget);          // Elimina el widget del
layout
            newIngredientWidget->deleteLater(); // Destruye el widget
        });
    ingredientsLayout->addWidget(newIngredientWidget);
    qDebug() << "El tamaño del puntero a IngredientWidget vacío es: "
        << sizeof(newIngredientWidget) << " bytes.";
    qDebug() << "El tamaño del objeto IngredientWidget vacío al que apunta es: "
        << sizeof(*newIngredientWidget) << " bytes.";
}

void MainWindow::clearIngredientWidgets() {
    while (QLayoutItem* item = ingredientsLayout->takeAt(0)) {
        if (QWidget* widget = item->widget()) {
            widget->deleteLater();
        }
        delete item;
    }
}

void MainWindow::actualizarRecipeCardWidgets() {
    for (int i = 0; i < recipesCardsLayout->count(); i++) {
        QWidget* widget = recipesCardsLayout->itemAt(i)->widget();

        RecipeCard* recipeCardWidget = qobject_cast<RecipeCard*>(widget);
        if (recipeCardWidget) {
            recipeCardWidget->setRecetaAsociada(listaRecetas.recuperarReceta(i));
            recipeCardWidget->actualizarVista();
        }
    }
}

void MainWindow::desactivarBotonesNavBar() {
    ui->homeButton->setEnabled(false);
    ui->recipesButton->setEnabled(false);
}

```

```

}

void MainWindow::activarBotonesNavBar() {
    ui->homeButton->setEnabled(true);
    ui->recipesButton->setEnabled(true);
}

void MainWindow::setCursorPointerForAllButtons() {
    QList<QWidget*> widgets = this->findChildren<QWidget*>();
    for (QWidget* widget : widgets) {
        // Verifica si el widget es un QPushButton
        QPushButton* button = qobject_cast<QPushButton*>(widget);
        if (button) {
            button->setCursor(
                Qt::PointingHandCursor); // Cambia el cursor a "pointer hand"
        }
    }
}

bool MainWindow::validarCamposAddRecipePage() {
    QList<QLineEdit*> camposTexto{ui->lineEditName, ui->lineEditDuration,
                                   ui->lineEditAutorFirstName,
                                   ui->lineEditAutorLastName};

    QStringList camposFaltantes;
    bool camposVacios = false;

    for (auto campo : camposTexto) {
        if (campo->text().isEmpty()) {
            camposVacios = true;
            break;
        }
    }

    if (camposVacios) {
        camposFaltantes << "Hay campos de texto vacios en la informacion
general";
    }

    if (ui->uploadPhotoImage->pixmap().isNull()) {
        camposFaltantes << "Debes agregar una imagen del platillo";
    }

    if (ui->instructionsText->toPlainText().isEmpty()) {
        camposFaltantes << "Debes agregar el procedimiento";
    }
}

```

```

    }

    bool ingredientesValidos = true;
    for (int i = 0; i < ingredientsLayout->count(); i++) {
        QWidget* widget = ingredientsLayout->itemAt(i)->widget();

        if (IngredientWidget* ingredientWidget =
            qobject_cast<IngredientWidget*>(widget)) {
            if (ingredientWidget->getNombre().isEmpty() ||
                ingredientWidget->getCantidad() < 0.001) {
                ingredientesValidos = false;
                break;
            }
        }
    }

    if (!ingredientesValidos) {
        camposFaltantes << "Debes llenar todos los campos de los ingredientes";
    }

    if (!camposFaltantes.isEmpty()) {
        QString mensaje = camposFaltantes.join("\n");
        QMessageBox::warning(this, "Campos Vacíos", mensaje);
        return false;
    }

    return true;
}

void MainWindow::limpiarCamposAddRecipePage() {
    ui->lineEditName->clear();
    ui->lineEditDuration->clear();
    ui->comboBoxCategory->setCurrentIndex(0);
    ui->lineEditAutorFirstName->clear();
    ui->lineEditAutorLastName->clear();
    ui->instructionsText->clear();
    ui->uploadPhotoImage->clear();
    clearIngredientWidgets();
    ui->confirmButton->setProperty("recipePointer", QVariant());
    ui->uploadPhotoButton->setProperty("imagePath", QVariant());
}

void MainWindow::llenarCamposAddRecipePage(Receta& recipe) {
    ui->lineEditName->setText(recipe.getNombre());

```



```

    ui->lineEditDuration->
>setText(QString::number(recipe.getTiempoPreparacion()));
    ui->comboBoxCategory->setCurrentIndex(recipe.getCategoriaToInt());
    ui->lineEditAutorFirstName->setText(recipe.getAutor().getNombre());
    ui->lineEditAutorLastName->setText(recipe.getAutor().getApellido());

    QString fileName = recipe.getRutaImagen();
    ui->uploadPhotoButton->setProperty("imagePath", fileName);

    if (!fileName.isEmpty()) {
        QPixmap image(fileName);
        ui->uploadPhotoImage->setPixmap(image);
        ui->uploadPhotoImage->setScaledContents(true);
    }

    ui->instructionsText->setText(recipe.getPreparacion());

    ListaIngredientes& lista = recipe.getListIngredientes();

    int i = 0;
    if (!lista.vacia()) {
        while (i <= lista.getUltimaPosicion()) {
            Ingrediente* in = lista.recuperarIngrediente(i);
            IngredientWidget* newIngredientWidget = new IngredientWidget(*in);
            connect(newIngredientWidget, &IngredientWidget::deleteClicked, this,
                [newIngredientWidget, this]() {
                    ingredientsLayout->removeWidget(
                        newIngredientWidget); // Elimina el widget del layout
                    newIngredientWidget->deleteLater(); // Destruye el widget
                });
            ingredientsLayout->addWidget(newIngredientWidget);
            i++;
        }
    }
}

void MainWindow::llenarCamposViewRecipePage(Receta* receta) {
    ui->tittleViewRecipe->setText(receta->getNombre());
    ui->imageViewRecipe->setPixmap(QPixmap(receta->getRutaImagen()));
    ui->Procedimiento->setText(receta->getPreparacion());
    ui->Ingredientes->setText(receta->getListIngredientes().toString());
    ui->tiempoPreparacion->setText(
        QString::number(receta->getTiempoPreparacion()) + " Minutos");
    ui->autor->setText(receta->getAutor().toString());
    ui->categoria->setText(receta->getCategoriaToString());
}

```

```

}

void MainWindow::onModifyRecipeClicked(RecipeCard* recipeCard) {
    Receta* recipe = recipeCard->getRecetaAsociada();

    ui->recipesButton->setChecked(false);
    ui->pages->setCurrentIndex(addRecipePage);
    ui->confirmButton->setText("Guardar Cambios");
    ui->confirmButton->setProperty("recipePointer",
                                   QVariant::fromValue(recipeCard));

    ui->uploadPhotoButton->setText("Cambiar Imagen");
    ui->uploadPhotoButton->setStyleSheet(
        "color: black; border-bottom: 2px solid blue;");

    desactivarBotonesNavBar();
    llenarCamposAddRecipePage(*recipe);
}

void MainWindow::onDeleteRecipeClicked(RecipeCard* recipeCard) {
    Receta* recipe = recipeCard->getRecetaAsociada();
    int pos = listaRecetas.localiza(recipe);

    if (pos != -1) {
        try {
            listaRecetas.eliminarReceta(pos);
            recipesCardsLayout->removeWidget(recipeCard);
            recipeCard->deleteLater();
            qDebug() << "Eliminada Correctamente";
            qDebug() << listaRecetas.getUltimaPosicion();
            qDebug() << listaRecetas.size();
        } catch (const std::exception& e) {
            QMessageBox::warning(this, "Error", e.what());
        }
    } else {
        qDebug() << "No se pudo localizar la receta";
    }
}

void MainWindow::onViewRecipe(RecipeCard* recipeCard) {
    Receta* recipe = recipeCard->getRecetaAsociada();
    ui->pages->setCurrentIndex(viewRecipePage);
    llenarCamposViewRecipePage(recipe);
}

```

```

void MainWindow::onSortChanged(int index) {
    if (listaRecetas.vacia()) {
        return;
    }

    if (index == 1) {
        listaRecetas.quickSort(Receta::compararPorNombre);
    } else {
        listaRecetas.quickSort(Receta::compararPorTiempoPreparacion);
    }

    actualizarRecipeCardWidgets();
}

void MainWindow::onFilterChanged(int) {
    if (recipesCardsLayout->count() > 0) {
        QString categoriaSeleccionada = ui->comboBoxFilter->currentText();
        filtrarPorCategoria(categoriaSeleccionada);
        recipesCardsLayout->update();
    }
}

void MainWindow::onSearchButtonClicked() {
    qDebug() << "Busqueda realizada";

    QString search = ui->searchRecipeLineEdit->text().trimmed();
    if (!search.isEmpty()) {
        Receta* objetivo = new Receta;
        objetivo->setNombre(search);

        int pos = listaRecetas.busquedaBinaria(objetivo,
Receta::compararPorNombre);
        if (pos != -1) {
            Receta* receta = listaRecetas.recuperarReceta(pos);
            llenarCamposViewRecipePage(receta);
            ui->pages->setCurrentIndex(viewRecipePage);
        } else {
            ui->messageWidget->setVisible(true);

            QTimer::singleShot(2000, this,
                [this]() { ui->messageWidget->setVisible(false);
});
        }
        delete objetivo;
    }
}

```

```

}

void MainWindow::onTextChanged(const QString& text) {
    ui->clearSearchLineEditButton->setVisible(!text.isEmpty());
}

void MainWindow::onClearButtonClicked() {
    ui->searchRecipeLineEdit->clear();
    ui->clearSearchLineEditButton->setVisible(false);
}

void MainWindow::filtrarPorCategoria(QString categoriaSeleccionada) {
    for (int i = 0; i < recipesCardsLayout->count(); i++) {
        QWidget* widget = recipesCardsLayout->itemAt(i)->widget();
        RecipeCard* recipeCardWidget = qobject_cast<RecipeCard*>(widget);

        if (recipeCardWidget) {
            // Recupera la receta asociada a la RecipeCard
            Receta* receta = recipeCardWidget->getRecetaAsociada();

            // Verifica si la receta tiene la categoría seleccionada
            if (receta->getCategoriaToQString() == categoriaSeleccionada ||
                categoriaSeleccionada == "Todas") {
                recipeCardWidget->setVisible(true); // Mostrar si coincide
            } else {
                recipeCardWidget->setVisible(false); // Ocultar si no coincide
            }
        }
    }
}

void MainWindow::onHomeButtonClicked() {
    ui->pages->setCurrentIndex(homePage);
    ui->recipesButton->setChecked(false);
    ui->homeButton->setChecked(true);
}

void MainWindow::onRecipesButtonClicked() {
    ui->pages->setCurrentIndex(recipesPage);
    ui->homeButton->setChecked(false);
    ui->recipesButton->setChecked(true);
}

void MainWindow::onAddRecipeButtonClicked() {
    ui->recipesButton->setChecked(false);
    desactivarBotonesNavBar();
}

```

```

    ui->pages->setCurrentIndex(addRecipePage);
    ui->confirmButton->setText("Agregar Receta");
}

void MainWindow::onDeleteAllRecipesButtonClicked() {
    if (recipesCardsLayout->count() == 0) {
        return;
    }

    QMessageBox::StandardButton reply;

    reply = QMessageBox::question(
        this, "Eliminar todas las recetas",
        "¿Estás seguro de que deseas eliminar todas las recetas?",
        QMessageBox::Yes | QMessageBox::No);

    if (reply == QMessageBox::Yes) {
        listaRecetas.anular();
        clearRecipeCards();
    }
}

void MainWindow::onCancelButtonClicked() {
    activarBotonesNavBar();
    limpiarCamposAddRecipePage();
    ui->pages->setCurrentIndex(recipesPage);
    ui->recipesButton->setChecked(true);
    ui->uploadPhotoButton->setStyleSheet(
        "color: rgb(240, 128, 0); border-bottom: 2px solid rgb(240, 128,
0);");
}

void MainWindow::onConfirmButtonClicked() {
    if (!validarCamposAddRecipePage()) {
        return;
    }

    if (ui->confirmButton->text() == "Agregar Receta") {
        agregarReceta();
    } else {
        modificarReceta();
        ui->uploadPhotoButton->setStyleSheet(
            "color: rgb(240, 128, 0); border-bottom: 2px solid rgb(240, 128,
0);");
    }
}

```

```

    limpiarCamposAddRecipePage();
    activarBotonesNavBar();
    ui->pages->setCurrentIndex(recipesPage);
    ui->recipesButton->setChecked(true);
}

void MainWindow::onUploadPhotoButtonClicked() {
    QString fileName = QFileDialog::getOpenFileName(
        this, tr("Seleccionar Imagen"), "",
        tr("Imágenes (*.png *.jpg *.jpeg *.bmp *.gif);;Todos los archivos (*)"));

    if (!fileName.isEmpty()) {
        QPixmap image(fileName);
        ui->uploadPhotoImage->setPixmap(image);
        ui->uploadPhotoImage->setScaledContents(true);
        if (image.isNull()) {
            QMessageBox::warning(this, tr("Error"),
                                tr("No se pudo cargar la imagen."));
            return;
        }
    }

    ui->uploadPhotoButton->setProperty("imagePath", fileName);
}

void MainWindow::onDeleteAllIngredientsButtonClicked() {
    if (ingredientsLayout->count() == 0) {
        return;
    }

    QMessageBox::StandardButton reply;

    reply = QMessageBox::question(
        this, "Eliminar todos los ingredientes",
        "¿Estás seguro de que deseas eliminar todos los ingredientes?",
        QMessageBox::Yes | QMessageBox::No);

    if (reply == QMessageBox::Yes) {
        clearIngredientWidgets();
    }
}

void MainWindow::onReturnButtonClicked() {

```

```

        ui->pages->setCurrentIndex(recipesPage);
    }

void MainWindow::guardarRecetas() {
    QString datosRecetas = QDir::currentPath() + "/recetas.json";
    QFile file(datosRecetas);

    if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {
        QMessageBox::warning(this, "Error", "No se pudo abrir el archivo
para guardar las recetas.");
        return;
    }

    QJsonArray recetasArray;

    int cantidadRecetas = listaRecetas.size();
    for (int i = 0; i < cantidadRecetas; i++) {
        Receta *receta = listaRecetas.recuperarReceta(i);
        if (receta) {
            QJsonObject recetaObject;
            recetaObject["nombre"] = receta->getNombre();
            recetaObject["categoria"] = receta->getCategoriaToInt();
            recetaObject["rutaImagen"] = receta->getRutaImagen();
            recetaObject["tiempoPreparacion"] = receta-
>getTiempoPreparacion();

            // Procesar autor
            Nombre autor = receta->getAutor();
            QJsonObject autorObject;
            autorObject["nombre"] = autor.getNombre();
            autorObject["apellido"] = autor.getApellido();
            recetaObject["autor"] = autorObject;

            // Preparación
            recetaObject["preparacion"] = receta->getPreparacion();

            // Ingredientes
            ListaIngredientes& ingredientes = receta-
>getListIngredientes();
            QJsonArray ingredientesArray;

            int cantidadIngredientes = ingredientes.size();
            for (int j = 0; j < cantidadIngredientes; j++) {
                Ingrediente *ingrediente =
ingredientes.recuperarIngrediente(j);

```

```

        if (ingrediente) {
            QJsonObject ingredienteObject;

            ingredienteObject["nombre"] = ingrediente->getNombre();
            ingredienteObject["cantidad"] = ingrediente-
>getCantidad();
            ingredienteObject["unidad"] = ingrediente-
>unidadMedidaToint();
            ingredientesArray.append(ingredienteObject);
        }
    }
    recetaObject["ingredientes"] = ingredientesArray;

    recetasArray.append(recetaObject);
}

}

QJsonDocument jsonDoc(recetasArray);
file.write(jsonDoc.toJson());
file.close();

QMessageBox::information(this, "Guardado", "Las recetas han sido
guardadas exitosamente.");
}

void MainWindow::cargarRecetas() {
    QString datosRecetas = QDir::currentPath() + "/recetas.json";
    QFile file(datosRecetas);

    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QMessageBox::warning(this, "Error", "No se pudo abrir el archivo
para cargar las recetas.");
        return;
    }

    QByteArray fileData = file.readAll();
    file.close();

    QJsonDocument jsonDoc(QJsonDocument::fromJson(fileData));
    QJsonArray recetasArray = jsonDoc.array();

    for (const QJsonValue &value : recetasArray) {
        QJsonObject recetaObject = value.toObject();
    }
}

```



```

        Receta *receta = new Receta;
        receta->setNombre(recetaObject["nombre"].toString());
        receta-
>setCategoria(static_cast<Receta::Categoria>(recetaObject["categoria"].toInt(
)));
        receta->setRutaImagen(recetaObject["rutaImagen"].toString());
        receta-
>setTiempoPreparacion(recetaObject["tiempoPreparacion"].toInt());

        QJsonObject autorObject = recetaObject["autor"].toObject();
        Nombre autor;
        autor.setNombre(autorObject["nombre"].toString());
        autor.setApellido(autorObject["apellido"].toString());
        receta->setAutor(autor);
        receta->setPreparacion(recetaObject["preparacion"].toString());

        // Leer ingredientes
        QJsonArray ingredientesArray =
recetaObject["ingredientes"].toArray();
        for (const QJsonValue &ingValue : ingredientesArray) {
            QJsonObject ingredienteObject = ingValue.toObject();
            Ingrediente *ingrediente = new Ingrediente;
            ingrediente->setNombre(ingredienteObject["nombre"].toString());
            ingrediente-
>setCantidad(ingredienteObject["cantidad"].toDouble());
            int unidadInt = ingredienteObject["unidad"].toInt();
            ingrediente-
>setUnidad(static_cast<Ingrediente::UnidadMedida>(unidadInt));
            receta->agregarIngrediente(ingrediente);
        }

        listaRecetas.agregarReceta(receta);
    }

    QMessageBox::information(this, "Cargado", "Las recetas han sido cargadas
exitosamente.");
}
void MainWindow::closeEvent(QCloseEvent *event){
    QMessageBox::StandardButton reply;
    reply = QMessageBox::question(this, "Confirmar", "¿Deseas guardar las
recetas antes de salir?",
                                QMessageBox::Yes | QMessageBox::No |
QMessageBox::Cancel);

```

```
    if (reply == QMessageBox::Yes) {  
        guardarRecetas();  
    } else if (reply == QMessageBox::Cancel) {  
        event->ignore();  
        return;  
    }  
  
    event->accept();  
}
```

Ejecución del programa:

Pagina home:



Pagina Recetas:

Recetario Digital

Home

Recetas

Agregar Una Receta

Eliminar Las Recetas

Buscar Recetas por Nombre o Categoria

Filtar por:

Todas

Ordenar por:

--Selecciona--

Pagina Agregar Receta:

Recetario Digital

Home

Recetas

INFORMACIÓN GENERAL DE LA RECETA

Subir Foto

Nombre de la receta

Ejemplo: Pollo a la naranja

Tiempo de preparacion

30Minutos

Categoria

Desayuno

Nombre del autor

Ejemplo: Juan Pablo

Apellido del autor

Ejemplo: Viramontes Cruz

DETALLES DE LA RECETA

Eliminar los ingredientes

Ingredientes

Añadir Ingrediente

Procedimiento

Escribe aqui el procedimiento de la receta

Cancelar

Agregar Receta

-Agregar foto y seleccionar categoría

Recetario Digital

— □ ×

Home

Recetas

INFORMACIÓN GENERAL DE LA RECETA



Nombre de la receta

Pollo a la naranja

Tiempo de preparacion

45

Minutos

Categoría

Desayuno

Comida

Cena

Navideño

Apellido del autor

Ejemplo: Viramontes Cruz

DETALLES DE LA RECETA

Eliminar los ingredientes

Ingredientes

+ Añadir Ingrediente

Procedimiento

Escribe aquí el procedimiento de la receta

× Cancelar

✓ Agregar Receta

-Agregar ingredientes

Recetario Digital

— □ ×

Home

Recetas

INFORMACIÓN GENERAL DE LA RECETA



Nombre de la receta

Pollo a la naranja

Tiempo de preparacion

45

Minutos

Categoria

Comida

Nombre del autor

Salvador Esaú

Apellido del autor

Rodríguez González

DETALLES DE LA RECETA

Eliminar los ingredientes

Ingredientes



1.5

g

kg

ml

l

cda

cdta

taza

pizca

oz

lb

lata

Pechuga de pollo



+ Añadir Ingrediente

Procedimiento

Escribe aqui el procedimiento de la receta

✕ Cancelar

✓ Agregar Receta


-Agregar procedimiento y guardar

Recetario Digital

Home

Recetas

INFORMACIÓN GENERAL DE LA RECETA



Nombre de la receta

Pollo a la naranja

Tiempo de preparacion

45

Minutos

Categoria

Comida

Nombre del autor

Salvador Esaú

Apellido del autor

Rodríguez González

DETALLES DE LA RECETA

Eliminar los ingredientes

Ingredientes

0.5

cda

Jengibre fresco

1

taza

Jugo de naranja

0.25

taza

Salsa de soya

+ Añadir Ingrediente

Procedimiento

1. Para el pollo, mezcla la sal con ajo, la pimienta, 1 taza de fécula, la harina, los huevos y la Leche Evaporada CARNATION® CLAVEL®, bate perfectamente hasta integrar por completo. Añade el pollo y mezcla; refrigera por 1 hora. Fríe el pollo en el aceite caliente y coloca en papel absorbente para retirar el exceso de grasa.

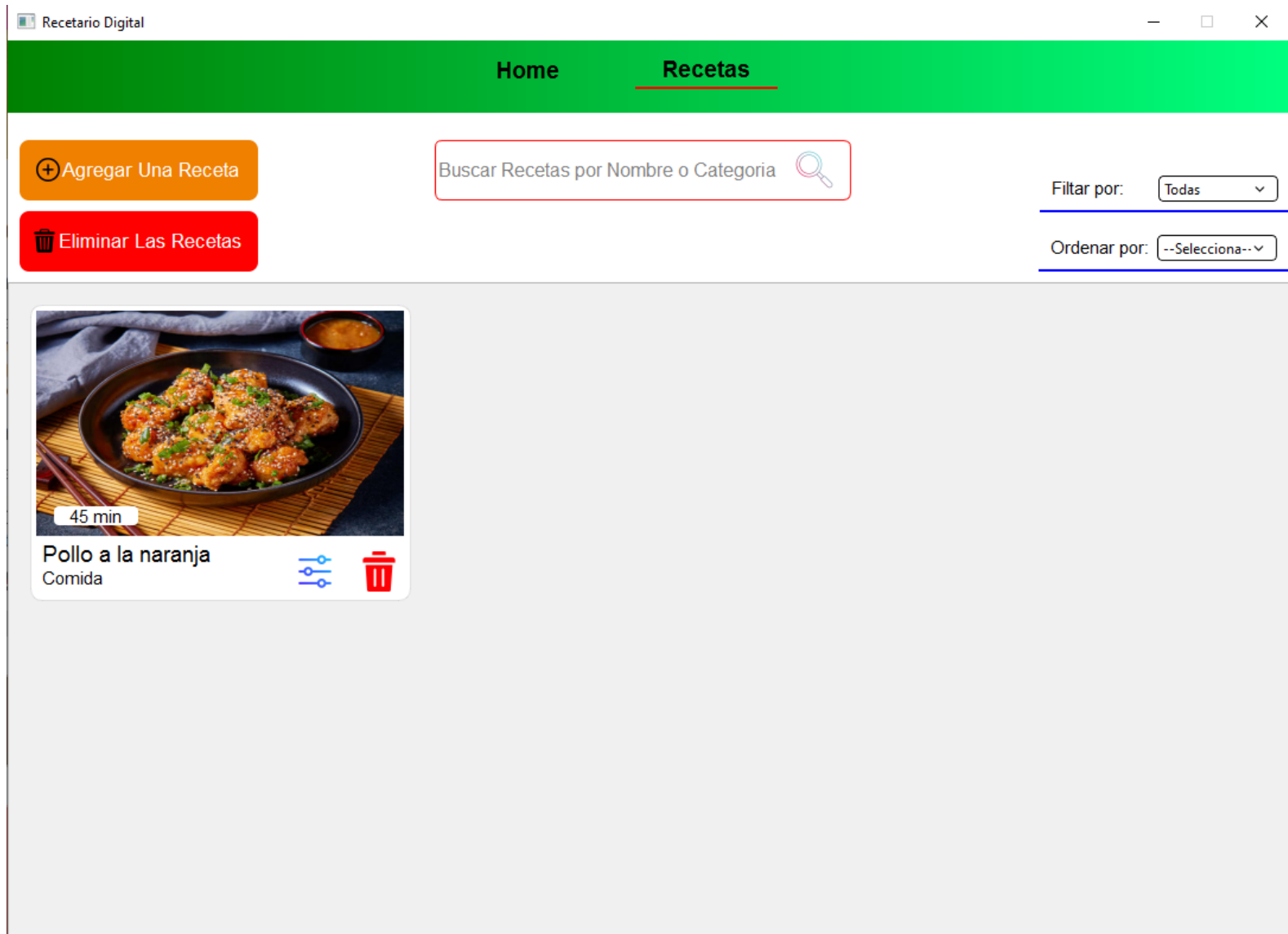
2. Para la salsa de naranja, calienta el aceite, fríe ligeramente el jengibre, el ajo y las hojuelas de chile. Añade el azúcar, el vinagre, el jugo de naranja y la Salsa de Soya MAGGI®; calienta hasta que hierva, agrega la fécula de maíz previamente disuelta y el aceite de ajonjolí; cocina hasta que espese ligeramente.

3. Sirve el pollo con la salsa, decora con el ajonjolí y el cebollín, ofrece.

Cancelar

Agregar Receta

-Se crea su representación en la interfaz




Modificar Receta:

-Para modificar una receta se reutiliza la pagina de agregar, al presionar el icono de modificar en una receta se llenan automáticamente los campos permitiendo modificar lo necesario

Recetario Digital

HomeRecetas

INFORMACIÓN GENERAL DE LA RECETA



Cambiar Imagen

Nombre de la receta

Pollo a la naranja

Tiempo de preparacion

45Minutos

Categoria

Comida

Nombre del autor

Salvador Esaú

Apellido del autor

Rodríguez González

DETALLES DE LA RECETA

Eliminar los ingredientes

Ingredientes

250.00

ml

Aceite de maiz

1.00

cda

Aceite vegetal

1.00

taza

Fécula de maíz

+ Añadir Ingrediente

Procedimiento

1.Para el pollo, mezcla la sal con ajo, la pimienta, 1 taza de fécula, la harina, los huevos y la Leche Evaporada CARNATION® CLAVEL®, bate perfectamente hasta integrar por completo. Añade el pollo y mezcla; refrigera por 1 hora. Fríe el pollo en el aceite caliente y coloca en papel absorbente para retirar el exceso de grasa.

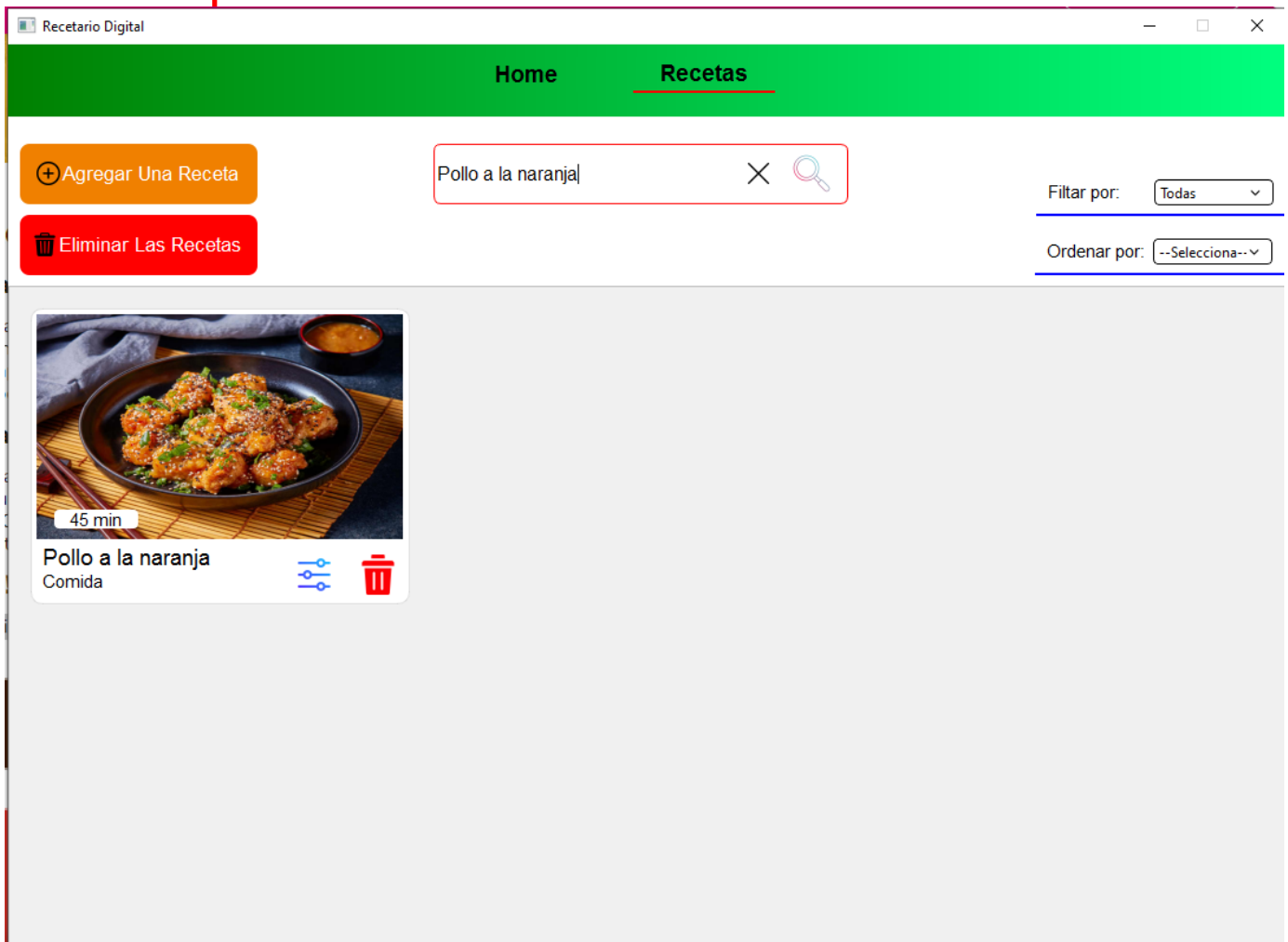
2.Para la salsa de naranja, calienta el aceite, fríe ligeramente el jengibre, el ajo y las hojuelas de chile. Añade el azúcar, el vinagre, el jugo de naranja y la Salsa de Soya MAGGI®; calienta hasta que hierva, agrega la fécula de maíz previamente disuelta y el aceite de ajonjolí; cocina hasta que espese ligeramente.

3.Sirve el pollo con la salsa, decora con el ajonjolí y el cebollín, ofrece.

Cancelar

Guardar Cambios

Búsqueda de recetas:



-Al presionar el icono de la lupa o presionar enter se realizará la búsqueda en la lista, en caso de No encontrarse solamente mostrará un mensaje indicándolo, caso contrario se redigirá automáticamente a la vista detallada de esa receta:

[← Regresar](#)

Tiempo de Preparacion

45 Minutos

Autor

**Salvador Esaú
Rodríguez González**

Categoría

Comida

Pollo a la naranja



---INGREDIENTES---

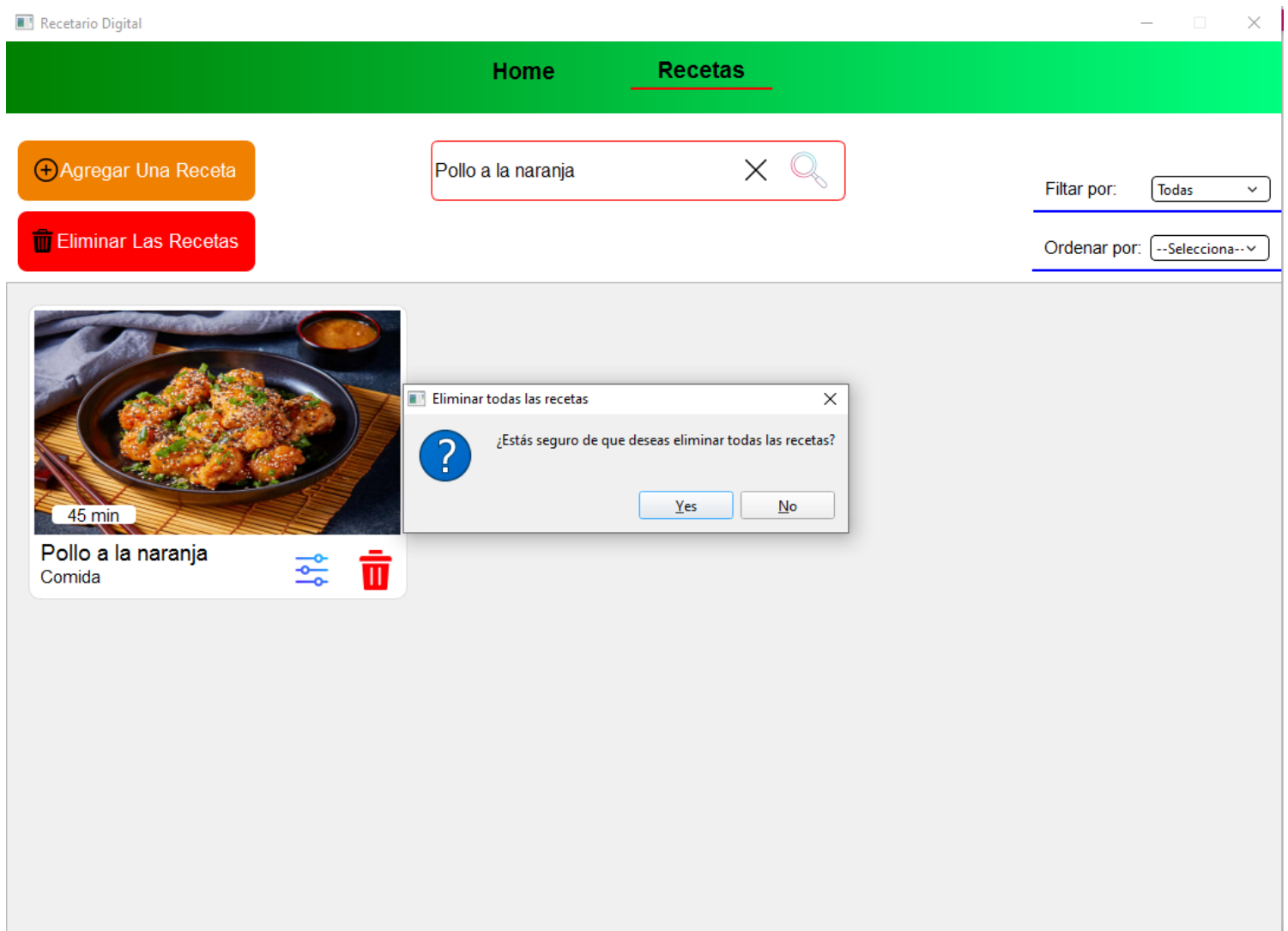
1. 250.00 Mililitros de Aceite de maiz
2. 1.00 Cucharada de Aceite vegetal
3. 1.00 Taza de Fécula de maíz

1. Para el pollo, mezcla la sal con ajo, la pimienta, 1 taza de fécula, la harina, los huevos y la Leche Evaporada CARNATION® CLAVEL®, bate perfectamente hasta integrar por completo. Añade el pollo y mezcla; refrigera por 1 hora. Fríe el pollo en el aceite caliente y coloca en papel absorbente para retirar el exceso de grasa.

2. Para la salsa de naranja, calienta el aceite, fríe ligeramente

Eliminar todas las recetas:

-Al presionar el botón de eliminar las recetas pedirá la confirmación antes de eliminar todas las recetas en la lista:



Eliminar todos los ingredientes de una receta:

-Al presionar el botón de eliminar los ingredientes estando modificando una receta pedirá la confirmación antes de eliminar todas las recetas en la lista que de igual forma la eliminación de la lista se hará hasta que presione guardar cambios:


Recetario Digital

Home Recetas

INFORMACIÓN GENERAL DE LA RECETA

DETALLES DE LA RECETA

Eliminar los ingredientes

 Cambiar Imagen

Nombre de la receta
Pollo a la naranja

Tiempo de preparacion
45 Minutos

Categoria
Comida

Nombre del autor
Salvador Esaú

Apellido del autor
Rodríguez González

Ingredientes

250.00 ml Aceite de maiz

1.00 cda Aceite vegetal

Eliminar todos los ingredientes

¿Estás seguro de que deseas eliminar todos los ingredientes?

Yes No

Procedimiento

1. Para el pollo, mezcla la sal con ajo, la pimienta, 1 taza de fécula, la harina, los huevos y la Leche Evaporada CARNATION® CLAVEL®, bate perfectamente hasta integrar por completo. Añade el pollo y mezcla; refrigera por 1 hora. Fríe el pollo en el aceite caliente y coloca en papel absorbente para retirar el exceso de grasa.

2. Para la salsa de naranja, calienta el aceite, fríe ligeramente el jengibre, el ajo y las hojuelas de chile. Añade el azúcar, el vinagre, el jugo de naranja y la Salsa de Soya MAGGI®; calienta hasta que hierva, agrega la fécula de maíz previamente disuelta y el aceite de ajonjolí; cocina hasta que espese ligeramente.

3. Sirve el pollo con la salsa, decora con el ajonjolí y el cebollín, ofrece.

Cancelar Guardar Cambios


-Se elimina la representación de los ingredientes en la interfaz, si presiona guardar cambios se elimina la información de la lista, si presiona cancelar simplemente regresa a la pagina de recetas y no realiza ningún cambio a la receta:

Recetario Digital

Home

Recetas

INFORMACIÓN GENERAL DE LA RECETA



Nombre de la receta

Pollo a la naranja

Tiempo de preparacion

45

Minutos

Categoria

Comida

Nombre del autor

Salvador Esaú

Apellido del autor

Rodríguez González

DETALLES DE LA RECETA

Eliminar los ingredientes

Ingredientes

+ Añadir Ingrediente

Procedimiento

1. Para el pollo, mezcla la sal con ajo, la pimienta, 1 taza de fécula, la harina, los huevos y la Leche Evaporada CARNATION® CLAVEL®, bate perfectamente hasta integrar por completo. Añade el pollo y mezcla; refrigera por 1 hora. Fríe el pollo en el aceite caliente y coloca en papel absorbente para retirar el exceso de grasa.

2. Para la salsa de naranja, calienta el aceite, fríe ligeramente el jengibre, el ajo y las hojuelas de chile. Añade el azúcar, el vinagre, el jugo de naranja y la Salsa de Soya MAGGI®; calienta hasta que hierva, agrega la fécula de maíz previamente disuelta y el aceite de ajonjolí; cocina hasta que espese ligeramente.

3. Sirve el pollo con la salsa, decora con el ajonjolí y el cebollín, ofrece.

Cancelar

Guardar Cambios

Recetario Digital

Home

Recetas

+ Agregar Una Receta

Eliminar Las Recetas


Buscar Recetas por Nombre o Categoría

Filtrar por:

Todas

Ordenar por:


--Selecciona--



15 min

Guacamole


Comida



20 min

Tacos de carne a...


Cena



20 min

Huevos a la Mexi...


Desayuno



25 min

Espaguetis a la ...


Navideño



25 min

Panqueques

Cena



30 min

Pollo a la naranja

Comida

Filtrar:

Recetario Digital

Home

Recetas

+ Agregar Una Receta

Eliminar Las Recetas


Buscar Recetas por Nombre o Categoría

Filtrar por:

Desayuno

Ordenar por:

--Selecciona--



20 min

Huevos a la Mexi...

Desayuno

-Ordenar por nombre:

Recetario Digital

Home Recetas







+ Agregar Una Receta

Eliminar Las Recetas

Buscar Recetas por Nombre o Categoría

Filtrar por: Todas

Ordenar por: Nombre

 25 min Espaguetis a la ... Navideño	 15 min Guacamole Comida	 20 min Huevos a la Mexi... Desayuno
 25 min Panqueques Cena	 240 min Pavo relleno Navideño	 40 min Pescado a la par... Comida

-Ordenar por tiempo de preparación:

Recetario Digital

Home Recetas







+ Agregar Una Receta

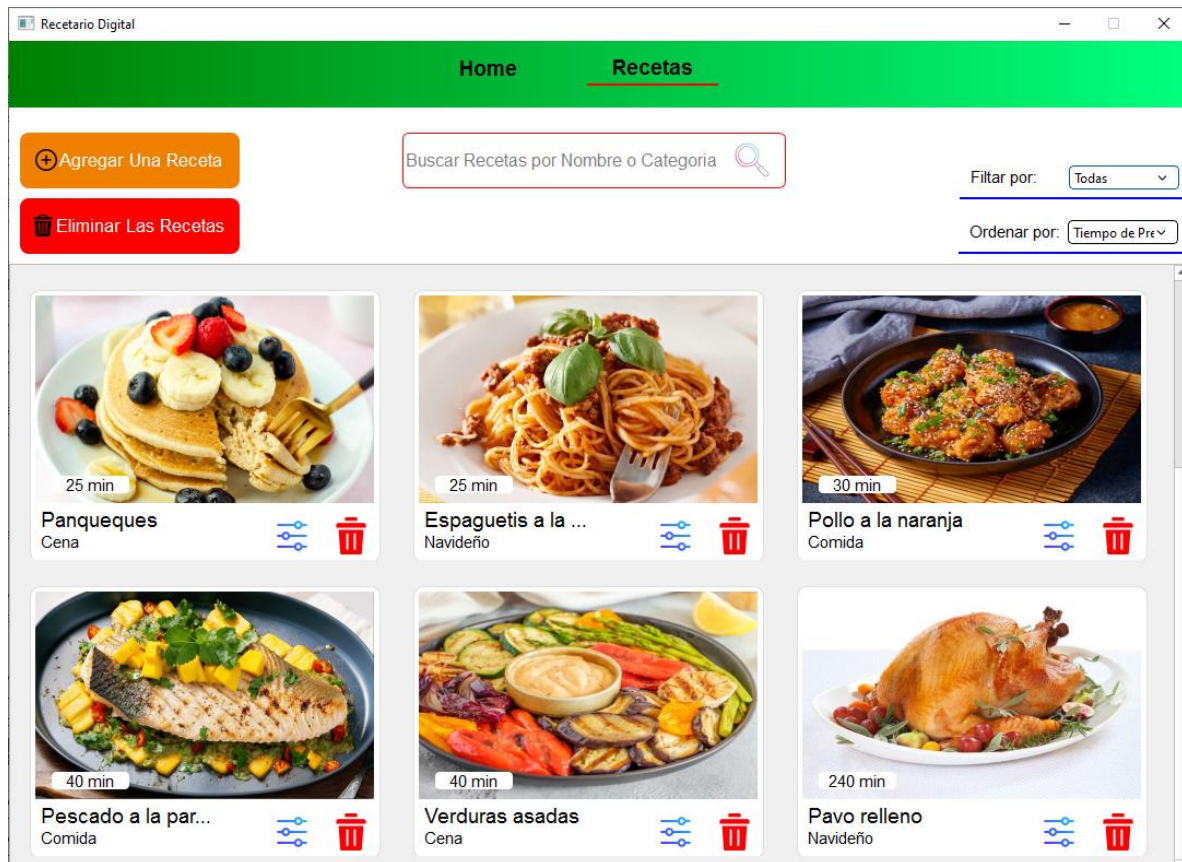
Eliminar Las Recetas

Buscar Recetas por Nombre o Categoría

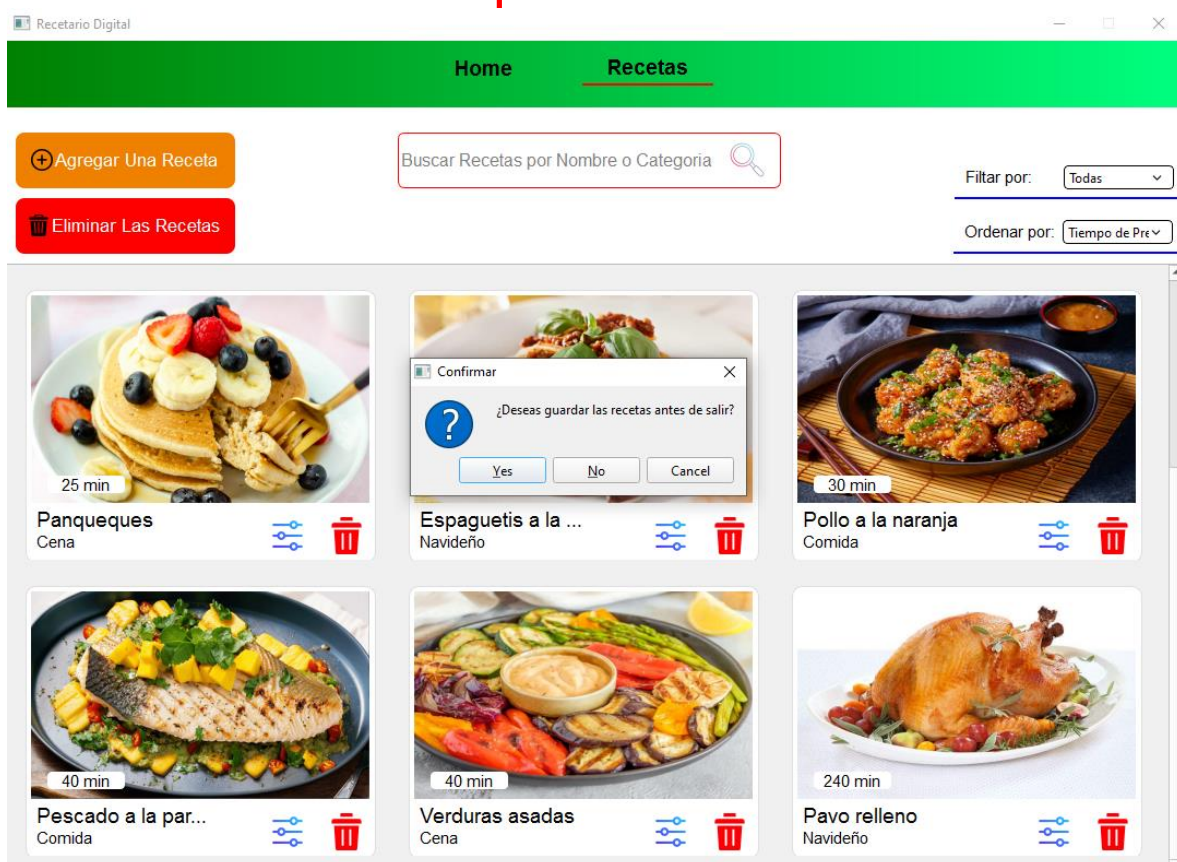
Filtrar por: Todas

Ordenar por: Tiempo de Pre

 15 min Guacamole Comida	 20 min Tacos de carne a... Cena	 20 min Huevos a la Mexi... Desayuno
 25 min Panqueques Cena	 25 min Espaguetis a la ... Navideño	 30 min Pollo a la naranja Comida



-Preguntar si desea guardar las recetas al disco al cerrar la aplicación



Conclusión:

A través del desarrollo de este recetario digital en su entrega preliminar, logré crear una aplicación funcional que permite a los usuarios gestionar sus recetas de manera sencilla. Utilizando C++ y Qt, se implementaron diversas funcionalidades, como la posibilidad de agregar, eliminar y modificar recetas e ingredientes, así como la funcionalidad de guardar y cargar las recetas al disco a través de un archivo json.

Durante el proceso, enfrenté desafíos relacionados con el manejo de archivos y la gestión de memoria, los cuales en su mayoría logré resolver. Este proyecto me permitió profundizar mis conocimientos en programación orientada a objetos y estructuras de datos en esta entrega preliminar usando Listas con arreglos dinámicos y en el diseño de interfaces gráficas.

A pesar de que aún me falta mejorar y implementar algunos aspectos en el manejo de las recetas y la interfaz grafica los cuales iré mejorando para la entrega final puedo decir que se cumplió con el objetivo principal en esta entrega preliminar del proyecto Recetario Digital.