

Proyecto Final: Recetario Digital - Entrega Final

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 noviembre 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 final 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 será una **lista ligada enlazada doblemente sin encabezado**, 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. **ListalIngredientes**: En contraste con **ListaRecetas**, esta clase se encargará de manejar una colección de objetos de tipo **Ingrediente**. La implementación de **ListalIngredientes** se realizará utilizando **lista ligada simplemente enlazada** 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 <QPixmap>
#include <QString>
#include <QTextStream>
#include <QJsonObject>
#include <QJsonArray>
#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;  
    QPixmap imagen;  
    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 setImagen(const QPixmap&);  
  
    void cargarImagen();  
  
    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;
const QPixmap& getImagen() 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 getCategoriaToString() const;
int getCategoriaToInt() const;

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

JsonObject toJson() const;
void fromJson(const JsonObject &);

};

#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;
        cargarImagen();
    }

    void Receta::setImagen(const QPixmap & nuevaImagen){
        imagen=nuevaImagen;
    }

    void Receta::cargarImagen(){
        if(!rutaImagen.isEmpty()){
            imagen=QPixmap(rutaImagen);
        }
    }

    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;
}

const QPixmap &Receta::getImagen() const{
    return imagen;
}

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::getListaIngredientes() {
    return listaIngredientes;
}

```

```

}

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

bool Receta::operator==(const Receta& other) const {
    return nombre == other.nombre;
}

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();
    }

    QJsonObject Receta::toJson() const{
        QJsonObject recetaObject;
        recetaObject["nombre"] = getNombre();
        recetaObject["categoria"] = getCategoriaToInt();
        recetaObject["rutaImagen"] = getRutaImagen();
        recetaObject["tiempoPreparacion"] = getTiempoPreparacion();

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

        recetaObject["preparacion"] = getPreparacion();

        // Procesar ingredientes
        QJsonArray ingredientesArray;
        ListaIngredientes::posicion* aux =
listaIngredientes.getPrimeraPosicion();
        while (aux != nullptr) {
            Ingrediente* ingrediente = aux->getIngredientePtr();
            if (ingrediente) {
                QJsonObject ingredienteObject;
                ingredienteObject["nombre"] = ingrediente->getNombre();
                ingredienteObject["cantidad"] = ingrediente->getCantidad();
                ingredienteObject["unidad"] = ingrediente->unidadMedidaToInt();
                ingredientesArray.append(ingredienteObject);
            }
            aux = aux->getSiguiente();
        }
        recetaObject["ingredientes"] = ingredientesArray;
    }

```

```

        return recetaObject;
    }

void Receta::fromJson(const QJsonObject & json){
    nombre = json["nombre"].toString();
    categoria = static_cast<Categoria>(json["categoria"].toInt());
    rutaImagen = json["rutaImagen"].toString();
    cargarImagen();
    tiempoPreparacion = json["tiempoPreparacion"].toInt();
    preparacion = json["preparacion"].toString();

    // Cargar autor
    QJsonObject autorObject = json["autor"].toObject();
    autor.setNombre(autorObject["nombre"].toString());
    autor.setApellido(autorObject["apellido"].toString());

    // Cargar ingredientes
    QJsonArray ingredientesArray = json["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));
        listaIngredientes.insertarIngrediente(ingrediente);
    }
}

```

Clase Ingrediente:

```

#ifndef INGREDIENTE_HPP
#define INGREDIENTE_HPP

#include <QString>
#include <QTextStream>
#include <iostream>

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);
    }

    QTextStream& operator<<(QTextStream& os, const Ingrediente& ingrediente) {
        os << ingrediente.getNombre() << "|"
            << ingrediente.getCantidad() << "|"
            << static_cast<int>(ingrediente.getUnidad()) << "#";
        return os;
    }

```

```

}

QTextStream& operator>>(QTextStream& is, Ingrediente& ingrediente) {
    QString nombre;
    float cantidad;
    int unidad;

    nombre = is.readLine('|');
    is >> cantidad;
    is.skipWhiteSpace();
    is >> unidad;

    ingrediente.setNombre(nombre);
    ingrediente.setCantidad(cantidad);
    ingrediente.setUnidad(static_cast<UnidadMedida>(unidad));

    return is;
}

QString Ingrediente::unidadMedidaToQString() const{
    switch (unidad) {
        case UnidadMedida::GRAMOS:           return "Gramos";
        case UnidadMedida::KILOGRAMOS:        return "Kilogramos";
        case UnidadMedida::MILILITROS:        return "Mililitros";
        case UnidadMedida::LITROS:            return "Litros";
        case UnidadMedida::CUCHARADA:         return "Cucharada";
        case UnidadMedida::CUCHARADITA:       return "Cucharadita";
        case UnidadMedida::TAZA:              return "Taza";
        case UnidadMedida::PIZCA:             return "Pizca";
        case UnidadMedida::ONZA:              return "Onza";
        case UnidadMedida::LIBRA:             return "Libra";
        case UnidadMedida::LATA:              return "Lata";
        default:                             return "Desconocido";
    }
}

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 <QFile>
#include <QJsonDocument>
#include "exception.hpp"
#include "receta.hpp"

class ListaRecetas {
private:
    class Nodo {
private:
        Receta* receta;
        Nodo* anterior;
        Nodo* siguiente;
    };

```

```

public:
    Nodo();
    Nodo(Receta*);
    Nodo(const Receta&);

    ~Nodo();

    Receta* getRecetaPtr() const;
    Receta getReceta() const;
    Nodo* getAnterior() const;
    Nodo* getSiguiente() const;

    void setRecetaPtr(Receta*);
    void setReceta(const Receta&);
    void setAnterior(Nodo*);
    void setSiguiente(Nodo*);
};

Nodo* ancla;
int size;

void copiarTodo(const ListaRecetas&);

public:
    typedef ListaRecetas::Nodo posicion;

    ListaRecetas();
    ListaRecetas(const ListaRecetas&);
    ~ListaRecetas();

    ListaRecetas& operator=(const ListaRecetas&);

    void agregarReceta(Receta*, Nodo*);
    void eliminarReceta(Nodo*);

    Receta* recuperarReceta(Nodo*) const;

    bool vacia() const;
    bool posicionValida(Nodo*) const;
    int getSize() const;

    Nodo* getPrimeraPosicion() const;
    Nodo* getUltimaPosicion() const;
    Nodo* getPosicionPrevia(Nodo*) const;
    Nodo* getSiguientePosicion(Nodo*) const;

```

```

Nodo* localiza(Receta*);

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

void quickSort(int (*cmp)(const Receta&, const Receta&));
void quickSort(Nodo*, Nodo*, int (*cmp)(const Receta&, const Receta&));
Nodo* partition(Nodo* left,
               Nodo* right,
               int (*cmp)(const Receta&, const Receta&));

void swap(Nodo*, Nodo*);

void guardarRecetas(const QString&);
void cargarRecetas(const QString&);

void anular();
};

#endif // LISTARECETAS_H

#include "listaRecetas.hpp"

ListaRecetas::ListaRecetas() : ancla(nullptr), size(0) {}

ListaRecetas::ListaRecetas(const ListaRecetas& otra)
    : ancla(nullptr), size(otra.size) {
    copiarTodo(otra);
}

ListaRecetas::~ListaRecetas() {
    anular();
}

void ListaRecetas::copiarTodo(const ListaRecetas& otra) {
    Nodo* aux(otra.ancla);
    Nodo* ultimo(nullptr);
    Nodo* nuevoNodo;

    while (aux != nullptr) {
        if ((nuevoNodo = new Nodo(aux->getReceta())) == nullptr) {
            throw Exception("Memoria no disponible al copiar");
        }
    }
}

```

```

    }

    if (ultimo == nullptr) {
        ancla = nuevoNodo;
    } else {
        nuevoNodo->setAnterior(ultimo);
        ultimo->setSiguiente(nuevoNodo);
    }

    ultimo = nuevoNodo;

    aux = aux->getSiguiente();
}
}

ListaRecetas& ListaRecetas::operator=(const ListaRecetas& otra) {
    anular();

    copiarTodo(otra);

    return *this;
}

bool ListaRecetas::vacia() const {
    return ancla == nullptr;
}

int ListaRecetas::getSize() const {
    return size;
}

void ListaRecetas::agregarReceta(Receta* receta, Nodo* p) {
    if (p != nullptr && !posicionValida(p)) {
        throw Exception("Posición invalida para insertar el dato.");
    }

    Nodo* aux(new Nodo(receta));

    if (aux == nullptr) {
        throw Exception("Memoria insuficiente para agregar una nueva cancion");
    }

    if (p == nullptr) { // Insertar al principio
        aux->setSiguiente(ancla);
    }
}

```



```

        if (ancla != nullptr) {
            ancla->setAnterior(aux);
        }
        ancla = aux;
    } else { // Insertar en cualquier posicion
        aux->setAnterior(p);
        aux->setSiguiente(p->getSiguiente());

        if (p->getSiguiente() != nullptr) {
            p->getSiguiente()->setAnterior(aux);
        }
        p->setSiguiente(aux);
    }
    size++;
}

void ListaRecetas::eliminarReceta(Nodo* p) {
    if (!posicionValida(p)) {
        throw Exception("Posicion invalida para eliminar el dato.");
    }

    if (p->getAnterior() != nullptr) {
        p->getAnterior()->setSiguiente(p->getSiguiente());
    }

    if (p->getSiguiente() != nullptr) {
        p->getSiguiente()->setAnterior(p->getAnterior());
    }

    if (p == ancla) {
        ancla = ancla->getSiguiente();
    }

    delete p;
    size--;
}

bool ListaRecetas::posicionValida(Nodo* pos) const {
    Nodo* aux(ancla);

    while (aux != nullptr) {
        if (aux == pos) {
            return true;
        }
    }
}

```

```

        aux = aux->getSiguiente();
    }

    return false;
}

ListaRecetas::Nodo* ListaRecetas::getPrimeraPosicion() const {
    return ancla;
}

ListaRecetas::Nodo* ListaRecetas::getUltimaPosicion() const {
    if (vacía()) {
        return nullptr;
    }

    Nodo* aux(ancla);

    while (aux->getSiguiente() != nullptr) {
        aux = aux->getSiguiente();
    }

    return aux;
}

ListaRecetas::Nodo* ListaRecetas::getPosicionPrevia(Nodo* p) const {
    if (p == ancla) {
        return nullptr;
    }

    return p->getAnterior();
}

ListaRecetas::Nodo* ListaRecetas::getSiguientePosicion(Nodo* p) const {
    if (!posiciónValida(p)) {
        return nullptr;
    }

    return p->getSiguiente();
}

ListaRecetas::Nodo* ListaRecetas::localiza(Receta* receta) {
    Nodo* aux(ancla);

    while (aux != nullptr && aux->getRecetaPtr() != receta) {
        aux = aux->getSiguiente();
    }
}

```

```

    }

    return aux;
}

Receta* ListaRecetas::recuperarReceta(Nodo* p) const {
    if (!posicionValida(p)) {
        throw Exception("Posicion invalida para recuperar el elemento.");
    }

    return p->getRecetaPtr();
}

ListaRecetas::Nodo* ListaRecetas::busquedaLineal(
    const Receta* objetivo,
    int (*cmp)(const Receta&, const Receta&)) const {
    Nodo* aux(anc1a);

    while (aux != nullptr) {
        if (cmp(aux->getReceta(), *objetivo) == 0) {
            return aux;
        }

        aux = aux->getSiguiete();
    }

    return aux;
}

void ListaRecetas::quickSort(int (*cmp)(const Receta&, const Receta&)) {
    if (anc1a == nullptr) {
        return;
    }

    quickSort(anc1a, getUltimaPosicion(), cmp);
}

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

```

```

    Nodo* pivotNode = partition(left, right, cmp);

    quickSort(left, pivotNode->getAnterior(), cmp);
    quickSort(pivotNode->getSiguiente(), right, cmp);
}

ListaRecetas::Nodo* ListaRecetas::partition(Nodo* left,
                                             Nodo* right,
                                             int (*cmp)(const Receta&,
                                                         const Receta&)) {

    Receta* pivot = right->getRecetaPtr();
    Nodo* i = left->getAnterior();

    for (Nodo* j = left; j != right; j = j->getSiguiente()) {
        if (cmp(*j->getRecetaPtr(), *pivot) <= 0) {
            i = (i == nullptr) ? left : i->getSiguiente();
            swap(i, j);
        }
    }

    i = (i == nullptr) ? left : i->getSiguiente();
    swap(i, right);

    return i;
}

void ListaRecetas::swap(Nodo* a, Nodo* b) {
    Receta* temp = a->getRecetaPtr();
    a->setRecetaPtr(b->getRecetaPtr());
    b->setRecetaPtr(temp);
}

void ListaRecetas::guardarRecetas(const QString& filePath) {
    QFile file(filePath);

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

    QJsonArray recetasArray;
    posicion* posActual = getPrimeraPosicion();

    while (posActual != nullptr) {

```

```

        Receta* receta = posActual->getRecetaPtr();
        if (receta) {
            recetasArray.append(
                receta->toJson()); // Usamos el método toJson de Receta
        }
        posActual = posActual->getSiguiente();
    }

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

void ListaRecetas::cargarRecetas(const QString& filePath) {
    QFile file(filePath);

    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        throw Exception(
            "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->fromJson(recetaObject);
        agregarReceta(receta, getUltimaPosicion());
    }
}

void ListaRecetas::anular() {
    Nodo* aux;

    while (ancla != nullptr) {
        aux = ancla;

        ancla = ancla->getSiguiente();
    }
}

```

```

        delete aux;
    }
}

ListaRecetas::Nodo::Nodo()
    : receta(nullptr), anterior(nullptr), siguiente(nullptr) {}

ListaRecetas::Nodo::Nodo(Receta* r)
    : receta(r), anterior(nullptr), siguiente(nullptr) {}

ListaRecetas::Nodo::Nodo(const Receta& r)
    : receta(new Receta(r)), anterior(nullptr), siguiente(nullptr) {
    if (receta == nullptr) {
        throw Exception("Memoria insuficiente al crear Nodo");
    }
}

ListaRecetas::Nodo::~Nodo() {
    delete receta;
}

Receta* ListaRecetas::Nodo::getRecetaPtr() const {
    return receta;
}

Receta ListaRecetas::Nodo::getReceta() const {
    if (receta == nullptr) {
        throw Exception("Receta inexistente");
    }
    return *receta;
}

ListaRecetas::Nodo* ListaRecetas::Nodo::getAnterior() const {
    return anterior;
}

ListaRecetas::Nodo* ListaRecetas::Nodo::getSiguiente() const {
    return siguiente;
}

void ListaRecetas::Nodo::setRecetaPtr(Receta* r) {
    receta = r;
}

void ListaRecetas::Nodo::setReceta(const Receta& r) {

```

```

    if (receta == nullptr) {
        if ((receta = new Receta(r)) == nullptr) {
            throw Exception("Memoria no disponible");
        }
    } else {
        *receta = r;
    }
}

void ListaRecetas::Nodo::setAnterior(Nodo* p) {
    anterior = p;
}

void ListaRecetas::Nodo::setSiguiente(Nodo* p) {
    siguiente = p;
}

```

Clase ListaIngredientes:

```

#ifndef LISTAINGREDIENTES_H
#define LISTAINGREDIENTES_H

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

class ListaIngredientes {
private:
    class Nodo {
private:
        Ingrediente* ingrediente;
        Nodo* siguiente;

public:
        Nodo();
        Nodo(Ingrediente*);
        Nodo(const Ingrediente&);

        Ingrediente* getIngredientePtr() const;
        Ingrediente getIngrediente() const;
        Nodo* getSiguiente() const;

        void setIngredientePtr(Ingrediente*);
    };
};

```

```

    void setIngrediente(const Ingrediente&);
    void setSiguiente(Nodo*);
};

Nodo* ancla;

int size;

void copiarTodo(const ListaIngredientes&);

public:
    typedef ListaIngredientes::Nodo posicion;

    ListaIngredientes();
    ListaIngredientes(const ListaIngredientes&);
    ~ListaIngredientes();

    ListaIngredientes& operator=(const ListaIngredientes&);

    bool vacia() const;
    bool posicionValida(Nodo*) const;

    void insertarIngrediente(Ingrediente*);
    void eliminarIngrediente(Nodo*);

    Nodo* getPrimeraPosicion() const;
    Nodo* getUltimaPosicion() const;
    Nodo* getPosicionPrevia(Nodo*) const;
    Nodo* getSiguientePosicion(Nodo*) const;

    Ingrediente* recuperarIngrediente(Nodo*);

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

    QString toQstring();
    void anular();

    int getSize() const;
};

#endif // LISTAINGREDIENTES_H

```



```

#include "listaIngredientes.hpp"

ListaIngredientes::ListaIngredientes() : ancla(nullptr), size(0) {}

ListaIngredientes::ListaIngredientes(const ListaIngredientes& otra)
    : ancla(nullptr), size(otra.size) {
    copiarTodo(otra);
}

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

void ListaIngredientes::copiarTodo(const ListaIngredientes& otra) {
    Nodo* aux(otra.ancla);
    Nodo* ultimo(nullptr);
    Nodo* nuevoNodo;

    while (aux != nullptr) {
        nuevoNodo = new Nodo(aux->getIngrediente());

        if (ultimo == nullptr) {
            ancla = nuevoNodo;
        } else {
            ultimo->setSiguiente(nuevoNodo);
        }

        ultimo = nuevoNodo;

        aux = aux->getSiguiente();
    }
}

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

    return *this;
}

bool ListaIngredientes::vacia() const {
    return ancla == nullptr;
}

```

```

bool ListaIngredientes::posicionValida(Nodo* p) const {
    Nodo* aux(ancla);

    while (aux != nullptr) {
        if (aux == p) {
            return true;
        }

        aux = aux->getSiguiente();
    }

    return false;
}

int ListaIngredientes::getSize() const {
    return size;
}

void ListaIngredientes::insertarIngrediente(Ingrediente* ingrediente) {
    Nodo* pos = buscarPosicionOrdenada(ingrediente);

    Nodo* aux(new Nodo(ingrediente));

    if (aux == nullptr) {
        throw Exception("Memoria insuficiente para agregar un
nuevoIngrediente");
    }

    if (pos == nullptr) { // Insertar al principio
        aux->setSiguiente(ancla);
        ancla = aux;
    } else { // Insertar en cualquier posicion
        aux->setSiguiente(pos->getSiguiente());
        pos->setSiguiente(aux);
    }

    size++;
}

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

```

```

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

    if (p == ancla) { // Eliminar el primero
        ancla = ancla->getSiguiente();
    } else { // Eliminar cualquier otro
        getPositionPrevia(p)->setSiguiente(p->getSiguiente());
    }

    delete p;
    size--;
}

ListaIngredientes::Nodo* ListaIngredientes::getPrimeraPosicion() const {
    return ancla;
}

ListaIngredientes::Nodo* ListaIngredientes::getUltimaPosicion() const {
    if (vacía()) {
        return nullptr;
    }

    Nodo* aux(ancla);

    while (aux->getSiguiente() != nullptr) {
        aux = aux->getSiguiente();
    }

    return aux;
}

ListaIngredientes::Nodo* ListaIngredientes::getPositionPrevia(Nodo* p) const
{
    if (p == ancla) {
        return nullptr;
    }

    Nodo* aux(ancla);

    while (aux != nullptr && aux->getSiguiente() != p) {
        aux = aux->getSiguiente();
    }
}

```

```

        return aux;
    }

ListaIngredientes::Nodo* ListaIngredientes::getSiguientePosicion(
    Nodo* p) const {
    if (!posicionValida(p)) {
        return nullptr;
    }

    return p->getSiguiente();
}

Ingrediente* ListaIngredientes::recuperarIngrediente(Nodo* p) {
    if (!posicionValida(p)) {
        throw Exception("Posicion invalida para recuperar el elemento.");
    }

    return p->getIngredientePtr();
}

ListaIngredientes::Nodo* ListaIngredientes::busquedaLineal(
    const Ingrediente* objetivo,
    int (*cmp)(const Ingrediente&, const Ingrediente&)) const {
    Nodo* aux(ancla);

    while (aux != nullptr) {
        if (cmp(aux->getIngrediente(), *objetivo) == 0) {
            return aux;
        }

        aux = aux->getSiguiente();
    }

    return aux;
}

ListaIngredientes::Nodo* ListaIngredientes::buscarPosicionOrdenada(
    const Ingrediente* ingrediente) const {
    Nodo* aux(ancla);

    while (aux != nullptr and
        aux->getIngrediente().getNombre() <= ingrediente->getNombre()) {
        aux = aux->getSiguiente();
    }
}

```

```

        return aux;
    }

QString ListaIngredientes::toString() {
    Nodo* aux(ancla);
    QString resultado;
    int i = 0;

    resultado += "---INGREDIENTES---\n\n";
    while (aux != nullptr) {
        resultado += QString::number(i + 1) + ". " +
            aux->getIngrediente().toString() + "\n";
        aux = aux->getSiguiente();

        if (aux != nullptr) {
            resultado += "-----\n";
        }
        i++;
    }

    return resultado;
}

void ListaIngredientes::anular() {
    Nodo* aux;

    while (ancla != nullptr) {
        aux = ancla;

        ancla = ancla->getSiguiente();

        delete aux;
    }
}

ListaIngredientes::Nodo::Nodo() : ingrediente(nullptr), siguiente(nullptr)
{}

ListaIngredientes::Nodo::Nodo(Ingrediente* i)
    : ingrediente(i), siguiente(nullptr) {}

ListaIngredientes::Nodo::Nodo(const Ingrediente& i)
    : ingrediente(new Ingrediente(i)), siguiente(nullptr) {
    if (ingrediente == nullptr) {
        throw Exception("Memoria insuficiente al crear Nodo");
    }
}

```

```

    }
}

Ingrediente* ListaIngredientes::Nodo::getIngredientePtr() const {
    return ingrediente;
}

Ingrediente ListaIngredientes::Nodo::getIngrediente() const {
    if (ingrediente == nullptr) {
        throw Exception("Ingrediente inexistente");
    }
    return *ingrediente;
}

ListaIngredientes::Nodo* ListaIngredientes::Nodo::getSiguiente() const {
    return siguiente;
}

void ListaIngredientes::Nodo::setIngredientePtr(Ingrediente* i) {
    ingrediente = i;
}

void ListaIngredientes::Nodo::setIngrediente(const Ingrediente& i) {
    if (ingrediente == nullptr) {
        if ((ingrediente = new Ingrediente(i)) == nullptr) {
            throw Exception("Memoria no disponible");
        }
    } else {
        *ingrediente = i;
    }
}

void ListaIngredientes::Nodo::setSiguiente(Nodo* p) {
    siguiente = p;
}

```

Clase IngredientWidget:

```

#ifndef INGREDIENTWIDGET_H
#define INGREDIENTWIDGET_H

#include <QComboBox>

```

```

#include <QDoubleValidator>
#include <QFile>
#include <QLabel>
#include <QLineEdit>
#include <QPixmap>
#include <QPushButton>
#include <QUiLoader>
#include <QVBoxLayout>
#include <QWidget>
#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*);

```

```

    void desactivarBotones();

    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");

    if (!receta.getImagen().isNull()) {
        imageLabel->setScaledContents(true);
        imageLabel->setPixmap(receta.getImagen());
    } else {
        imageLabel->setText("Sin imagen");
    }
}

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

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

```

```

void RecipeCard::desactivarBotones() {
    modifyButton->setVisible(false);
    deleteButton->setVisible(false);
}

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 <QFileDialog>
#include <QIntValidator>
#include <QJsonArray>
#include <QJsonDocument>
#include <QJsonObject>
#include <QMainWindow>
#include <QMessageBox>
#include <QPushButton>
#include <QScrollBar>
#include <QStackedWidget>
#include <QTimer>
#include <QVBoxLayout>
#include <QWidget>

#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;
    QHBoxLayout* lastRecipesLayout;
    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;

    QTimer* scrollTimer;
    int scrollPosition = 0;

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

    // 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 createRecipesCards();
void clearLastRecipesCardsSection();
void createLastRecipesCards();
void addRecipeCardWidget(Receta*);
void addIngredientWidget();
void clearIngredientWidgets();
void actualizarRecipeCardWidgets();
void filtrarPorCategoria(QString);

// Utilidades
void desactivarBotonesNavBar();
void activarBotonesNavBar();
void setCursorPointerForAllButtons();
void autoScrollLastRecipesArea();
void pauseAutoScroll();
void resumeAutoScroll();
void stopAutoScroll();

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

void llenarCamposViewRecipePage(Receta*);
void limpiarCamposViewRecipePage();

// 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();
    void onPageChanged(int);

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"

using namespace std;

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);
    cargarRecetas();

    initializeImagesDirectory();
    setupLastRecipesSection();
    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;
    }

    QString recetasFile = QDir::currentPath() + "/recetas.json";
    QFile file(recetasFile);

    // Verificar y crear el archivo recetas.json si no existe
    if (!file.exists()) {
        if (file.open(QIODevice::WriteOnly)) {
            file.close();
            qDebug() << "Archivo recetas.json creado en:" << recetasFile;
        } else {
            qDebug() << "Error al crear el archivo recetas.json en:" <<
recetasFile;
        }
    } else {
        qDebug() << "El archivo recetas.json ya existe en:" << recetasFile;
    }
}

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

    createRecipesCards();
}

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);

    connect(ui->pages, &QStackedWidget::currentChanged, this,
            &MainWindow::onPageChanged);
}

void MainWindow::setupLastRecipesSection() {
    QWidget* lastRecipesContainer = new QWidget();
    lastRecipesLayout = new QHBoxLayout(lastRecipesContainer);
    lastRecipesLayout->setSpacing(10);
    lastRecipesLayout->setContentsMargins(10, 0, 10, 0);
    lastRecipesLayout->setAlignment(Qt::AlignVCenter);
    ui->lastRecipesArea->setWidget(lastRecipesContainer);
    ui->lastRecipesArea->setWidgetResizable(true);

    createLastRecipesCards();
    setupAutoScroll();
}

void MainWindow::setupAutoScroll() {
    scrollTimer = new QTimer(this);
    connect(scrollTimer, &QTimer::timeout, this,

```

```

        &MainWindow::autoScrollLastRecipesArea);

QScrollBar* scrollBar = ui->lastRecipesArea->horizontalScrollBar();
connect(scrollBar, &QScrollBar::sliderPressed, this,
        &MainWindow::pauseAutoScroll);
connect(scrollBar, &QScrollBar::sliderReleased, this,
        &MainWindow::resumeAutoScroll);

scrollTimer->start(46);
}

void MainWindow::autoScrollLastRecipesArea() {
    QScrollBar* scrollBar = ui->lastRecipesArea->horizontalScrollBar();

    scrollPosition += 1;

    if (scrollPosition >= scrollBar->maximum()) {
        scrollPosition = scrollBar->minimum();
    }

    scrollBar->setValue(scrollPosition);
}

void MainWindow::pauseAutoScroll() {
    if (scrollTimer->isActive()) {
        scrollTimer->stop();
    }
}

void MainWindow::resumeAutoScroll() {
    QTimer::singleShot(2000, this, [this]() {
        if (!scrollTimer->isActive()) {
            scrollTimer->start(46);
        }
    });
}

void MainWindow::stopAutoScroll() {
    if (scrollTimer) {
        scrollTimer->stop();
    }
}

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,
listaRecetas.getUltimaPosicion());
} catch (const Exception& e) {
    QMessageBox::warning(this, "Error", QString::fromStdString(e.what()));
}

addRecipeCardWidget(nuevaReceta);

qDebug() << listaRecetas.getUltimaPosicion();
qDebug() << listaRecetas.getSize();
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;
}
row = 0;
col = 0;
}

void MainWindow::createRecipesCards() {
    if (listaRecetas.getSize() > 0) {
        ListaRecetas::posicion* posActual = listaRecetas.getPrimeraPosicion();

        while (posActual != nullptr) {
            addRecipeCardWidget(posActual->getRecetaPtr());
            posActual = posActual->getSiguiente();
        }
    }
}

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

void MainWindow::createLastRecipesCards() {
    ListaRecetas::posicion* posActual = listaRecetas.getUltimaPosicion();

    int i = 0;
    while (i < 6 && posActual != nullptr) {
        RecipeCard* recetaCard = new RecipeCard(posActual->getRecetaPtr());
        recetaCard->setFixedWidth(400);
        recetaCard->desactivarBotones();
        connect(recetaCard, &RecipeCard::imageClicked, this,
            [this, recetaCard]() { onViewRecipe(recetaCard); });

        recetaCard->setFixedWidth(330);
        posActual = posActual->getAnterior();

        lastRecipesLayout->addWidget(recetaCard);

        i++;
    }
}

```

```

    }
}

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 vacio es: "
        << sizeof(newIngredientWidget) << " bytes.";
    qDebug() << "El tamaño del objeto IngredientWidget vacio 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() {
    ListaRecetas::posicion* posActual = listaRecetas.getPrimeraPosicion();
    int i = 0;

    while (posActual != nullptr && i < recipesCardsLayout->count()) {
        QWidget* widget = recipesCardsLayout->itemAt(i)->widget();
        RecipeCard* recipeCardWidget = qobject_cast<RecipeCard*>(widget);

        if (recipeCardWidget) {
            recipeCardWidget->setRecetaAsociada(posActual->getRecetaPtr());
            recipeCardWidget->actualizarVista();
        }

        posActual = posActual->getSiguiente();
        i++;
    }
}

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);
        }
    }
}

```

```

    }
}

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()) {
        ui->uploadPhotoImage->setPixmap(recipe.getImagen());
        ui->uploadPhotoImage->setScaledContents(true);
    }

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

    ListaIngredientes& lista = recipe.getListaIngredientes();

```

```

ListaIngredientes::posicion* posActual = lista.getPrimeraPosicion();

while (posActual != nullptr) {
    Ingrediente* in = lista.recuperarIngrediente(posActual);
    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);
    posActual = posActual->getSiguiente();
}
}

void MainWindow::llenarCamposViewRecipePage(Receta* receta) {
    ui->tittleViewRecipe->setText(receta->getNombre());

    if (!receta->getImagen().isNull()) {
        ui->imageViewRecipe->setPixmap(receta->getImagen());
    } else {
        ui->imageViewRecipe->setText("Sin Imagen");
    }

    ui->Procedimiento->setText(receta->getPreparacion());
    ui->Ingredientes->setText(receta->getListaIngredientes().toString());
    ui->tiempoPreparacion->setText(
        QString::number(receta->getTiempoPreparacion()) + " Minutos");
    ui->autor->setText(receta->getAutor().toString());
    ui->categoria->setText(receta->getCategoriaToString());
}

void MainWindow::limpiarCamposViewRecipePage() {
    ui->tittleViewRecipe->clear();
    ui->imageViewRecipe->setPixmap(QPixmap());
    ui->Procedimiento->clear();
    ui->Ingredientes->clear();
    ui->tiempoPreparacion->clear();
    ui->autor->clear();
    ui->categoria->clear();
}

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();
    ListaRecetas::posicion* pos = listaRecetas.localiza(recipe);

    if (pos != nullptr) {
        try {
            listaRecetas.eliminarReceta(pos);

            clearRecipeCards();
            createRecipesCards();

            qDebug() << "Eliminada Correctamente";
            qDebug() << listaRecetas.getUltimaPosicion();
            qDebug() << listaRecetas.getSize();
        } catch (const std::exception& e) {
            QMessageBox::warning(this, "Error", e.what());
        }
    } else {
        qDebug() << "No se pudo localizar la receta";
    }
}

void MainWindow::onViewRecipe(RecipeCard* recipeCard) {
    ui->returnButton->setProperty("previousPage", ui->pages->currentIndex());

    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();

        clearRecipeCards();
        if (categoriaSeleccionada == "Todas") {
            createRecipesCards();
        }

        filtrarPorCategoria(categoriaSeleccionada);
    }
}

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

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

        ListaRecetas::posicion* pos =
            listaRecetas.busquedaLineal(objetivo, Receta::compararPorNombre);
        if (pos != nullptr) {
            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) {
    if (listaRecetas.getSize() > 0) {
        ListaRecetas::posicion* posActual = listaRecetas.getPrimeraPosicion();

        while (posActual != nullptr) {
            if (posActual->getRecetaPtr()->getCategoriaToString() ==
                categoriaSeleccionada) {
                addRecipeCardWidget(posActual->getRecetaPtr());
            }

            posActual = posActual->getSiguiente();
        }
    }
}

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 las ingredientes?",
        QMessageBox::Yes | QMessageBox::No);

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

```



```
        QDir::currentPath());
    }
}

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


Inicio

Recetas

INFORMACIÓN GENERAL DE LA RECETA

DETALLES DE LA RECETA

Eliminar los Ingredientes



Nombre de la receta

Pizza estilo chicago

Tiempo de preparación

115

Minutos

Desayuno

Comida

Cena

Navideño

Salvador Esau

Apellido del autor

Rodriguez Gonzalez

Ingredientes

Añadir ingrediente

Procedimiento

Escribe aquí el procedimiento de la receta

Cancelar

Agregar Receta


-Agregar ingredientes

Recetario Digital

Inicio

Recetas

INFORMACIÓN GENERAL DE LA RECETA



Nombre de la receta

Pizza estilo chicago

Tiempo de preparación

115 Minutos

Categoría

Comida

Nombre del autor

Salvador Esau

Apellido del autor

Rodriguez Gonzalez

DETALLES DE LA RECETA

Eliminar los ingredientes

Ingredientes

500

g

kg

ml

l

cdla

cdita

taza

pizca

oz

lb

lata

Harina de trigo

Añadir ingrediente

Procedimiento

Escribe aquí el procedimiento de la receta

Cancelar

Agregar Receta

-Agregar procedimiento y guardar

INFORMACIÓN GENERAL DE LA RECETA



Nombre de la receta

Pizza estilo chicago

Tiempo de preparacion

115

Minutos

Categoria

Comida

Nombre del autor

Salvador Esau

Apellido del autor

Rodriguez Gonzalez

DETALLES DE LA RECETA

Eliminar los ingredientes

Ingredientes



0.75

taza

Agua



500

g

Queso manchego



1

cdta

Hierbas finas



+ Añadir Ingrediente

Procedimiento

1.Horno precalentado a 180 °C

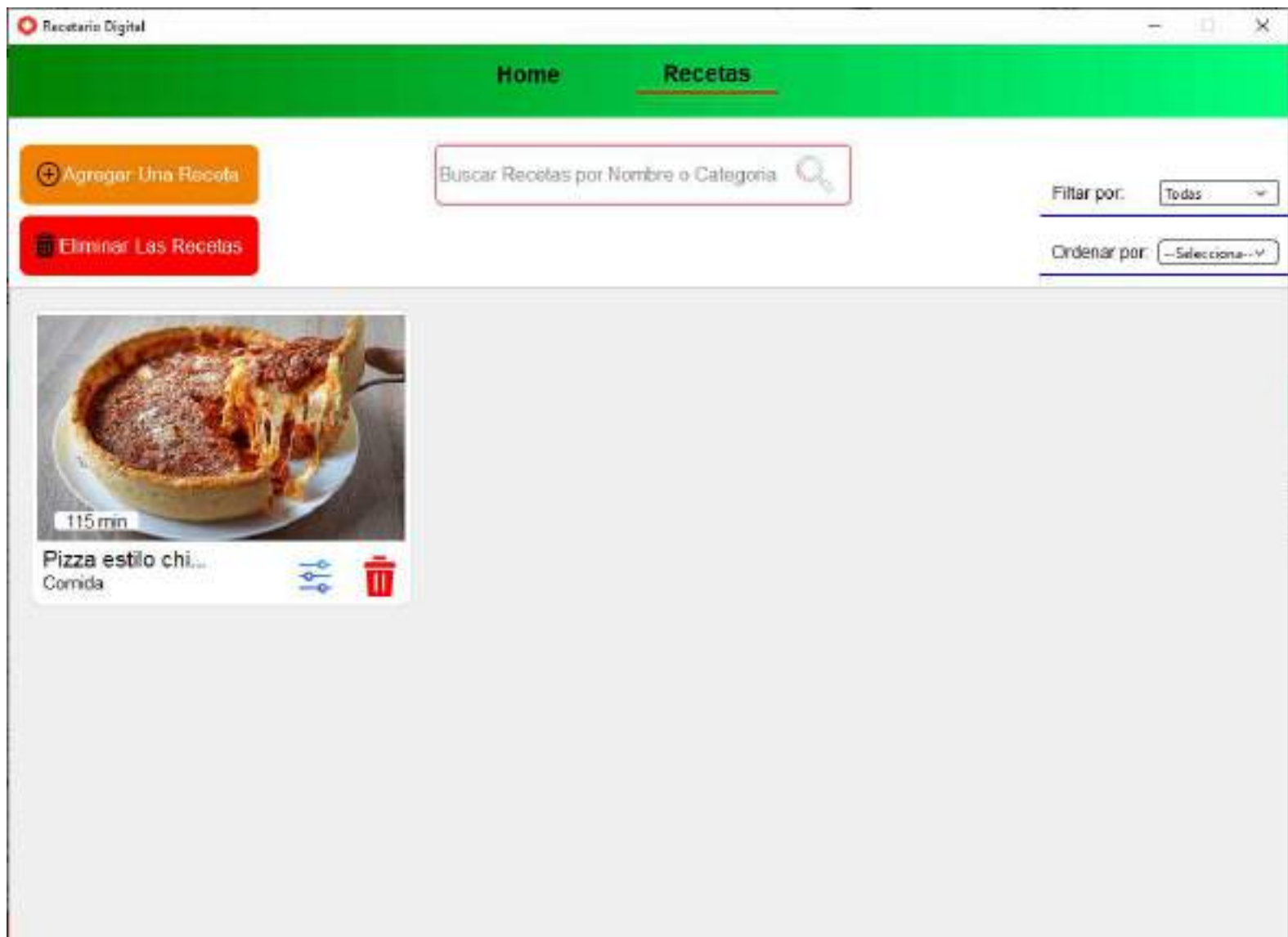
2.Para la masa, mezcla la harina con la miel, ¼ taza de aceite de oliva, la levadura, las hierbas finas y el agua poco a poco; amásala hasta que esté tersa, elástica y no se te pegue en los dedos. Colócala en un recipiente grande previamente engrasado, cubre con papel adherente y déjala reposar en un lugar tibio hasta que doble su volumen.

3.Divide la masa en 2 partes iguales, con ayuda de un rodillo extiéndelas sobre una

× 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



Nombre de la receta

Pizza estilo chicago

Tiempo de preparacion

115Minutos

Categoria

Comida

Nombre del autor

Salvador Esau

Apellido del autor

Rodriguez Gonzalez

DETALLES DE LA RECETA

Eliminar los ingredientes

Ingredientes

1.50

taza

Salsa de tomate

1.00

cdta

Hierbas finas

500.00

g

Queso manchego

+ Añadir Ingrediente

Procedimiento

1.Horno precalentado a 180 °C

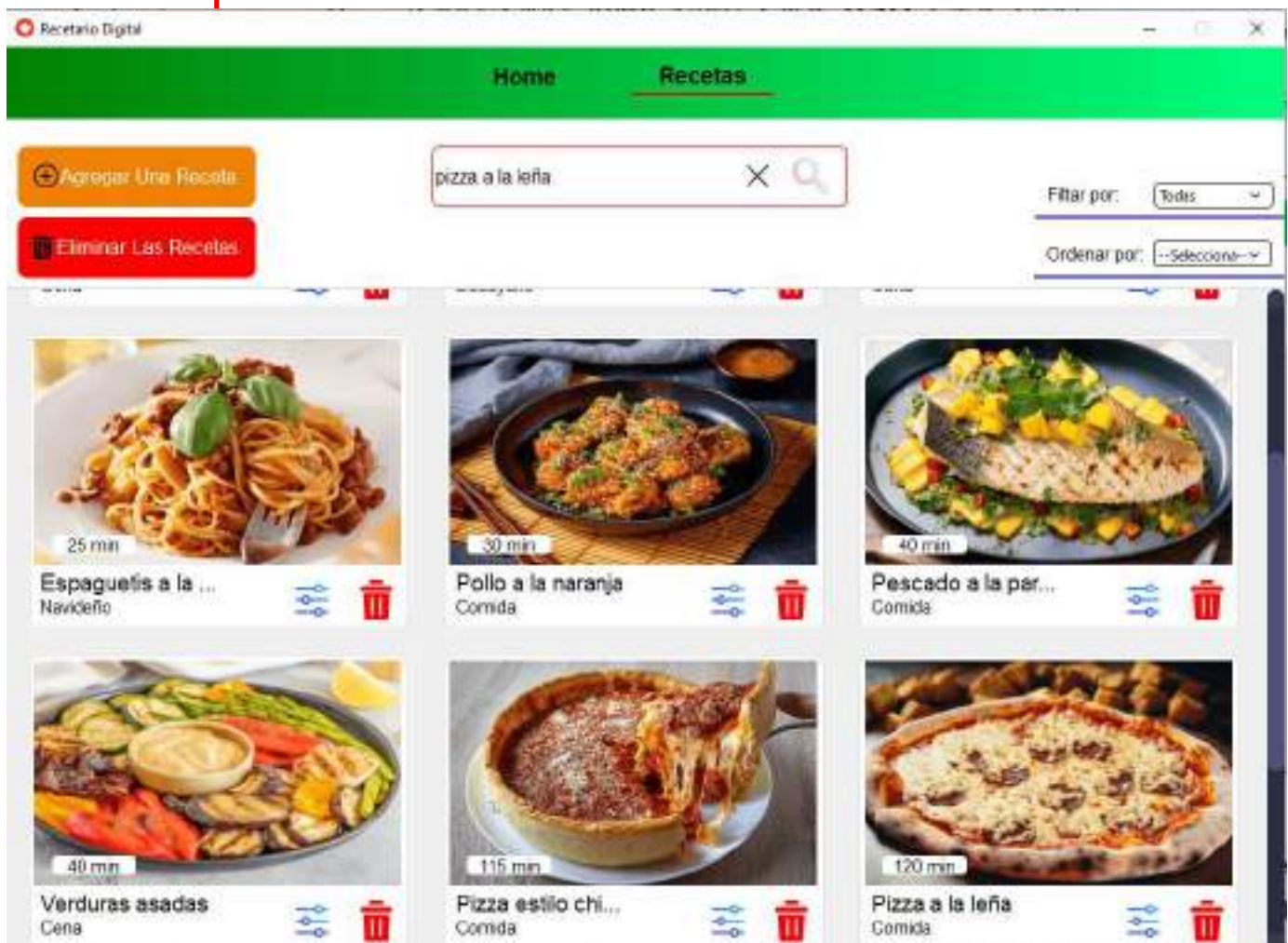
2.Para la masa, mezcla la harina con la miel, ¼ taza de aceite de oliva, la levadura, las hierbas finas y el agua poco a poco; amásala hasta que esté tersa, elástica y no se te pegue en los dedos. Colócala en un recipiente grande previamente engrasado, cubre con papel adherente y déjala reposar en un lugar tibio hasta que doble su volumen.

3.Divide la masa en 2 partes iguales, con ayuda de un rodillo extiéndelas sobre una mesa enharinada hasta formar 2 círculos delgados. Cubre un molde para pastel de 24 cm de diámetro por 7 cm de alto, engrasado y

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 Preparación

120 Minutos



Autor

**Salvador Esau
Rodríguez Gonzalez**



Categoría

Comida

Pizza a la leña



—INGREDIENTES—

1. 4.00 Cucharada de sal
2. 364.00 Gramos de agua fría
3. 20.00 Kilogramos de levadura fresca

Pon dos tercios de agua en un recipiente grande; pon el resto de agua a hervir y luego agrégala al recipiente con la que está fría. Esta mezcla te permitirá obtener la temperatura ideal para activar la levadura. Agrega la sal y levadura. Si vas a mezclar los ingredientes a mano: Pon la harina en un recipiente grande y agrégale la mezcla de levadura. Revuelve los ingredientes con una cuchara de madera hasta que se empiece a formar la masa. Sobre una superficie enharinada

-Si no se encuentra la receta:

Recetario Digital

Home Recetas

Agregar Una Receta

Eliminar Las Recetas

pollo kfc

La receta no fue encontrada

Filtrar por: Todas

Ordenar por: --Selecciona--

25 min

Espaguetis a la ...
Naviderlo

30 min

Pollo a la naranja
Comida

40 min

Pescado a la par...
Comida

40 min

Verduras asadas
Cena

115 min

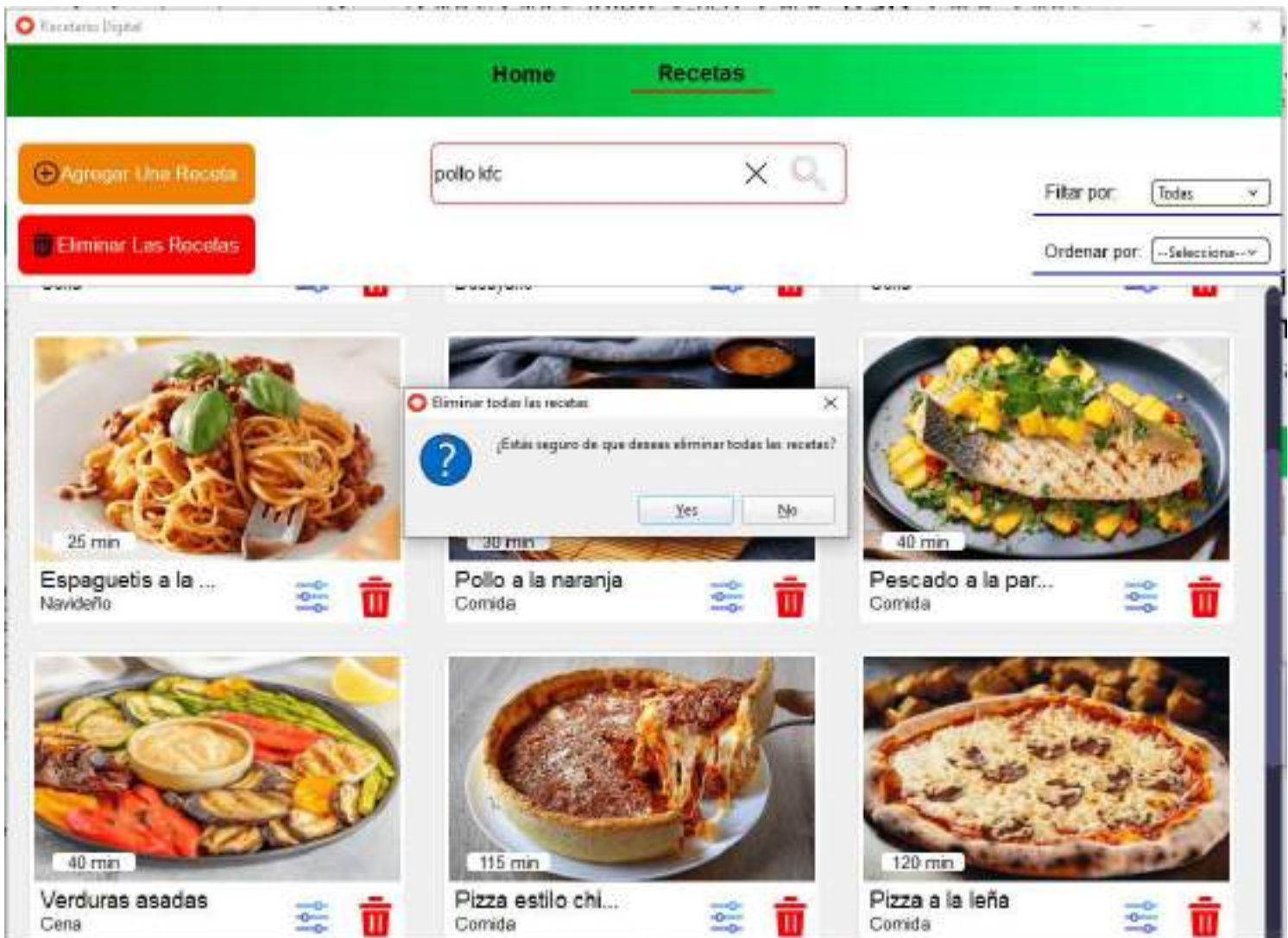
Pizza estilo chi...
Comida

120 min

Pizza a la leña
Comida

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

Cambia Imagen

Nombre de la receta

Pizza estilo chicago

Tiempo de preparacion

115 Minutos

Categoría

Comida

Nombre del autor

Salvador Esau

Apellido del autor

Rodriguez Gonzalez

Ingredientes

1.50 taza Salsa de tomate

1.00 cdta Hierbas finas

Eliminar todos los ingredientes

¿Estás seguro de que deseas eliminar todos los ingredientes?

Yes No

Procedimiento

1. Horno precalentado a 180 °C

2. Para la masa, mezcla la harina con la miel, ¼ taza de aceite de oliva, la levadura, las hierbas finas y el agua poco a poco; amásala hasta que esté tersa, elástica y no se te pegue en los dedos. Colócala en un recipiente grande previamente engrasado, cubre con papel adherente y déjala reposar en un lugar tibio hasta que doble su volumen.

3. Divide la masa en 2 partes iguales, con ayuda de un rodillo extiéndelas sobre una mesa enharinada hasta formar 2 círculos delgados. Cubre un molde para pastel de 24 cm de diámetro por 7 cm de alto, engrasado y enharinado con un círculo de masa; agrega el queso tipo manchego y el pepperoni hasta cubrir ¾ partes del molde.


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

HomeRecetas

INFORMACIÓN GENERAL DE LA RECETA



Nombre de la receta

Pizza estilo chicao

Tiempo de preparacion

115Minutos

Categoria

Comida

Nombre del autor

Salvador Esau

Apellido del autor

Rodriguez Gonzalez

DETALLES DE LA RECETA

Eliminar los ingredientes

Ingredientes

Añadir Ingrediente

Procedimiento

1.Horno precalentado a 180 °C

2.Para la masa, mezcla la harina con la miel, ¼ taza de aceite de oliva, la levadura, las hierbas finas y el agua poco a poco; amásala hasta que esté tersa, elástica y no se te pegue en los dedos. Colócala en un recipiente grande previamente engrasado, cubre con papel adherente y déjala reposar en un lugar tibio hasta que doble su volumen.

3.Divide la masa en 2 partes iguales, con ayuda de un rodillo extiéndelas sobre una mesa enharinada hasta formar 2 círculos delgados. Cubre un molde para pastel de 24 cm de diámetro por 7 cm de alto, engrasado y enharinado con un círculo de masa; agrega el queso tipo manchego y el pepperoni hasta cubrir ¾ partes del molde.

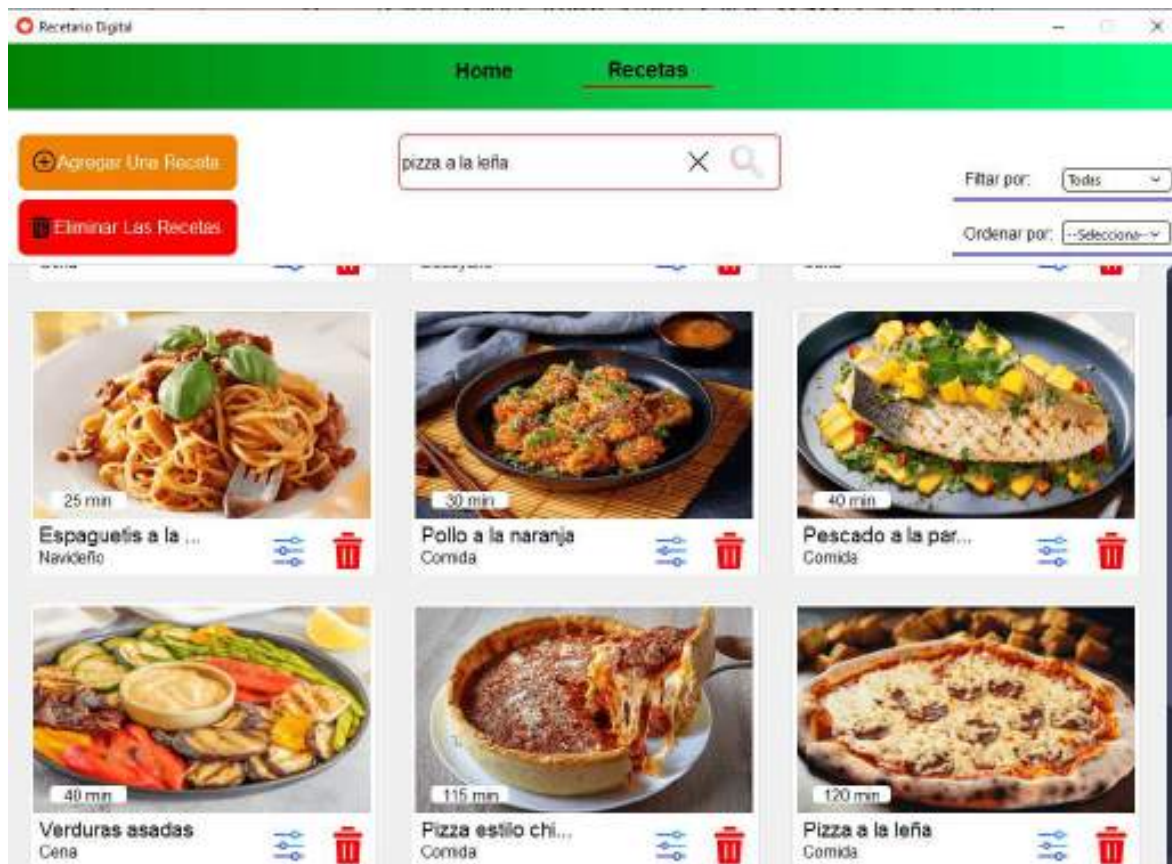
CancelarGuardar Cambios

Pruebas con 10 recetas:

-página de inicio sección ultimas recetas:

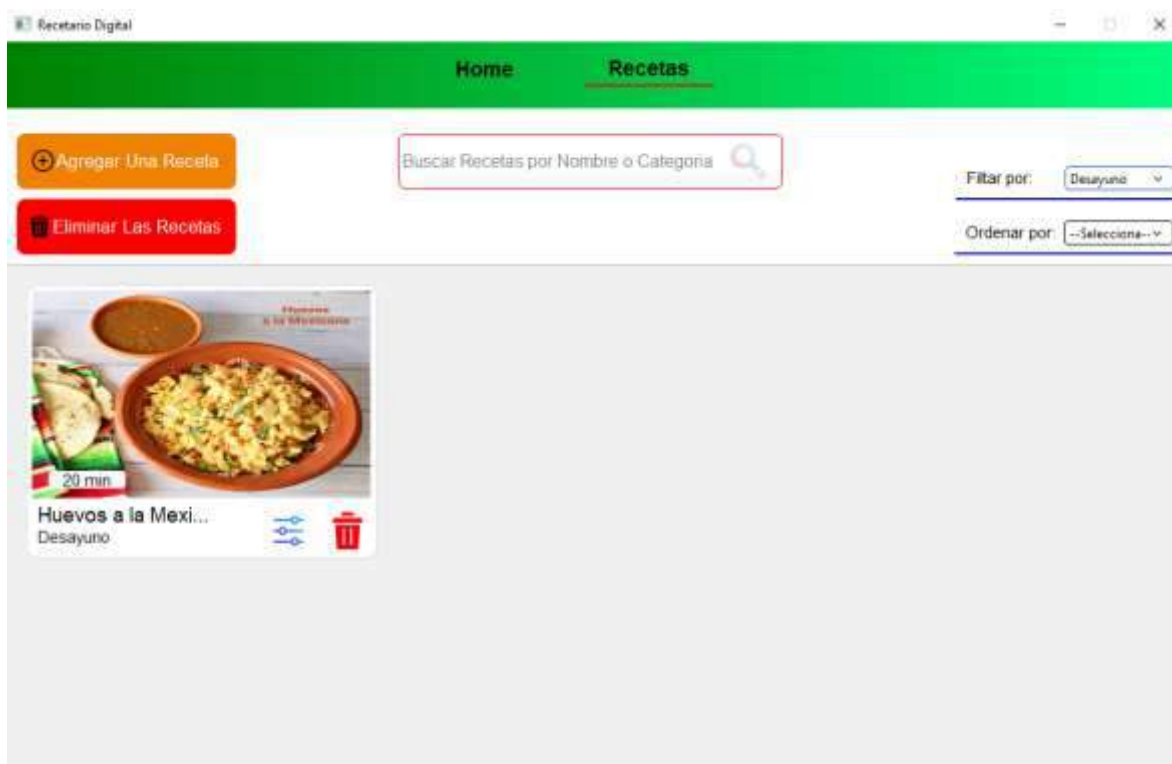


-página recetas:



Filtrar:

-desayuno



-comida

Recetas Digital

Home Recetas


Agregar una Receta

Eliminar Las Recetas

pollo kfc

Filtrar por: Comida


Ordenar por: --Selecciona--




30 min

Pollo a la naranja

Comida







40 min

Pescado a la par...

Comida







115 min

Pizza estilo chi...

Comida






120 min

Pizza a la leña

Comida



-cena

Recetas Digital

Home Recetas


Agregar una Receta

Eliminar Las Recetas

pollo kfc

Filtrar por: Cena


Ordenar por: --Selecciona--




20 min

Tacos de carne a...

Cena







25 min

Panqueques

Cena






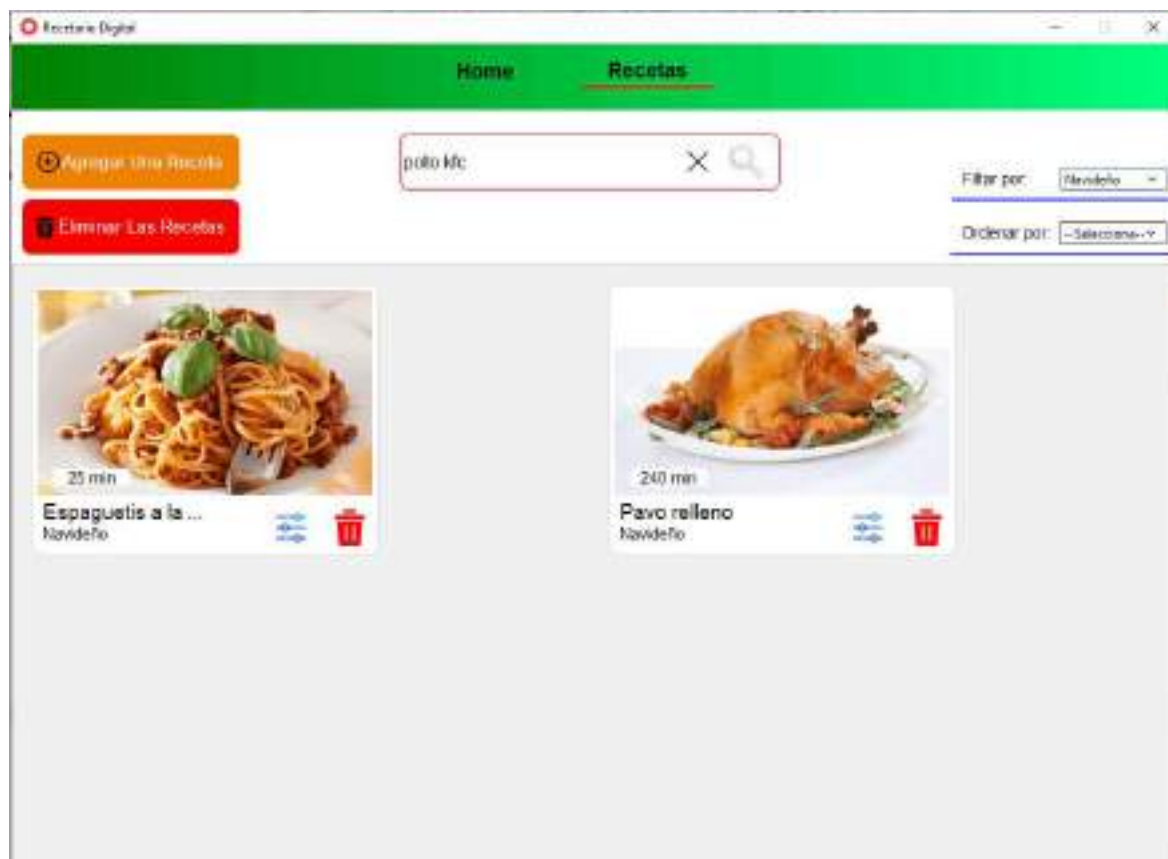
40 min

Verduras asadas

Cena



-navideño



-Ordenar por nombre:

Recetario Digital

Home Recetas


[Agregar Una Receta](#)

[Eliminar Las Recetas](#)

Buscar Recetas por Nombre o Categoría


Filtrar por:

Ordenar por:




25 min

Novedosa




20 min

Desayuno




25 min

Cena




240 min

Novedosa



40 min

Comida



120 min

Comida

Recetario Digital

Home Recetas


[Agregar Una Receta](#)

[Eliminar Las Recetas](#)

Buscar Recetas por Nombre o Categoría


Filtrar por:

Ordenar por:




115 min

Comida




30 min

Comida



20 min

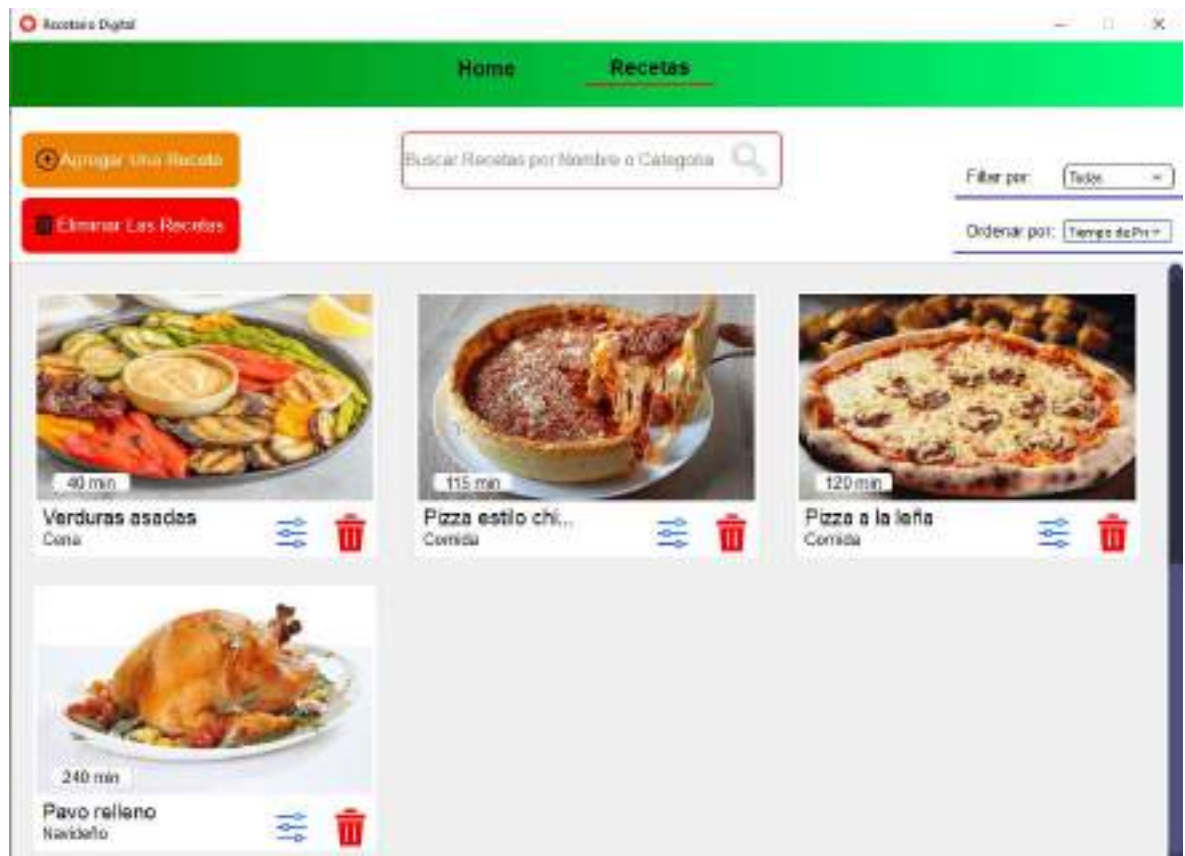
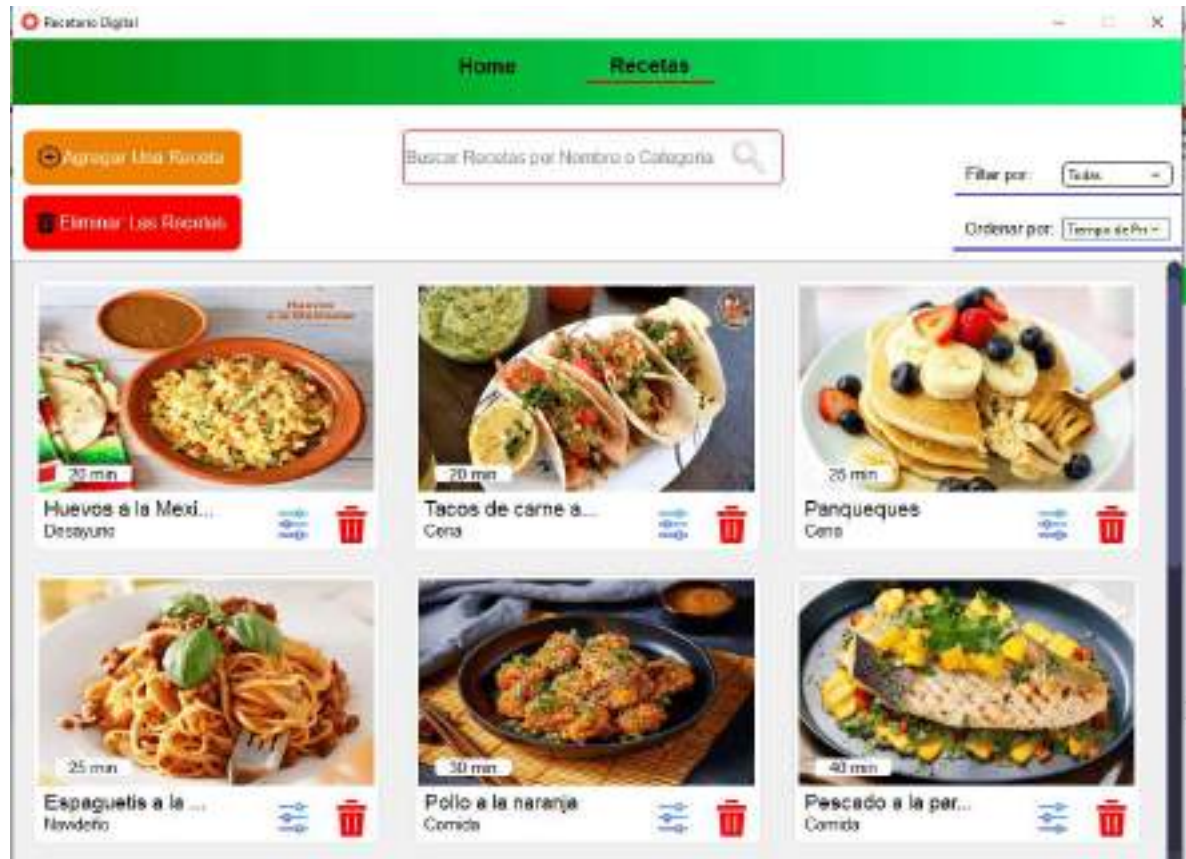
Cena



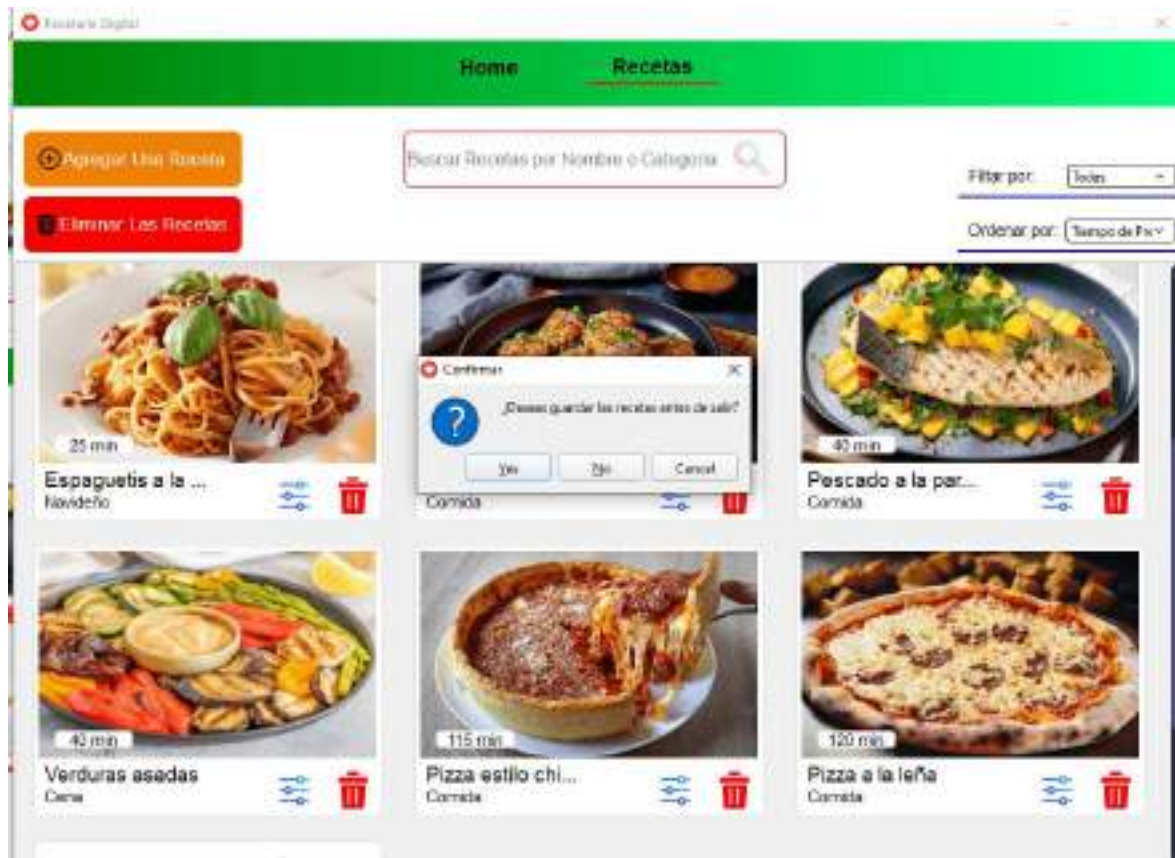
40 min

Cena

-Ordenar por tiempo de preparación:



-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 logre resolver. Este proyecto me permitió profundizar mis conocimientos en programación orientada a objetos y estructuras de datos en esta entrega final usando listas doblemente ligadas para las recetas y listas simplemente ligadas para los ingredientes

Al finalizar si pude mejorar algunos aspectos de la interfaz gráfica que quedaron pendientes en la entrega preliminar como por ejemplo la sección de ultimas recetas. Puedo decir que se cumplió con el objetivo principal del proyecto Recetario Digital.