ensIIE

Rapport de Projet

RecettesIntelligentes

Application de gestion de recettes basée sur l'architecture MVC

Réalisé par : Baha Eddine Jamel ET Aya ghiouan Soussi
Année universitaire : 2024–2025

Table des matières

1 Informations Générales

1.1 Présentation du projet

RecettesIntelligents est une application desktop destinée à la gestion complète de recettes de cuisine, permettant aux utilisateurs de créer, organiser et consulter leurs recettes, ingrédients et instructions dans une interface moderne et multi-plateforme. Elle supporte l'internationalisation et la persistance des données via une base SQLite.

1.2 Technologies et plateformes

- **Technologies principales** : Qt 6.x, C++17, SQLite
- **Architecture** : MVC (Model-View-Controller) + pattern DAO
- **Langues supportées** : Français, Anglais
- **Plateformes** : Windows, macOS, Linux

2 Objectifs du Projet

L'application vise à :

1. Gérer une collection personnelle de recettes de cuisine
2. Organiser ingrédients et instructions de manière structurée
3. Rechercher et filtrer rapidement les recettes
4. Stocker et gérer plats
5. Offrir une interface moderne et intuitive
6. Être portable sur différents systèmes d'exploitation

3 Architecture Technique

3.1 Pattern de conception

L'application utilise une architecture en couches basée sur le pattern MVC, combinée avec le pattern DAO pour la persistance des données.

```
Interface (UI/Views) : Formulaires/*.ui

Backend (Controller) : ControllerHeader/*.h Controller/*.cpp

Metiers ( Model )  : modelsHeader/*.h models/*.cpp

Base de Données (SQLite) : dataHeader/*.h data/*.c
```

3.2 Description des couches

3.2.1 Couche Présentation/View (UI)

- **mainwindow.ui** : fenêtre principale, affichage de la liste des recettes et détails.
- **ajouterrecettedialog.ui** : création d'une nouvelle recette.

— **ajouteringredientdialog.ui** : ajout d'ingrédients avec quantités et unités.

— **ajouterinstructiondialog.ui** : ajout d'instructions simples ou composées.

— **imagedroplabel.ui** : widget pour drag & drop d'images.

3.2.2 Couche Contrôleur

La couche Contrôleur est responsable de la gestion des interactions entre l'utilisateur et l'application. Elle reçoit les actions de l'interface graphique (clics, saisies, sélections), déclenche les traitements appropriés auprès de la couche Modèle (services), puis met à jour la Vue en conséquence.

— **mainwindow.cpp / mainwindow.h** : agit comme contrôleur principal de l'application. Il intercepte les événements de l'interface utilisateur, orchestre les appels aux services métiers (recettes, ingrédients, instructions) et synchronise les données affichées dans l'interface graphique.
  — Gestion complète des recettes (création, modification, suppression, sélection)
  — Coordination entre les modèles Qt et les services métier
  — Gestion de la recherche et du filtrage des recettes
  — Mise à jour dynamique des vues (tables, listes, arbres)

— **ajouterrecettedialog.cpp / ajouterrecettedialog.h** : gère la fenêtre de dialogue dédiée à l'ajout d'une nouvelle recette. Il valide les données saisies par l'utilisateur avant de déléguer la création de la recette au service correspondant.

— **ajouteringredientdialog.cpp / ajouteringredientdialog.h** : contrôle l'interface d'ajout d'un ingrédient à une recette existante. Il assure la cohérence des quantités, des unités et transmet les informations au service des ingrédients.

— **ajouterinstructiondialog.cpp / ajouterinstructiondialog.h** : gère l'ajout et l'organisation des instructions simples et composées d'une recette. Il orchestre l'interaction entre l'utilisateur et le service des instructions.

— **imagedroplabel.cpp / imagedroplabel.h** : composant graphique personnalisé permettant la gestion du glisser-déposer d'images. Il intercepte les événements utilisateur liés aux fichiers images et notifie le contrôleur principal pour leur traitement.

3.2.3 Couche Model

La couche Model regroupe les composants responsables de la représentation des données métier, de la logique applicative et de la communication avec la couche Contrôleur. Elle est indépendante de l'interface graphique et de la couche de persistance.

— **Modèles Qt pour la liaison interface / données** :
  — **RecetteModel** : expose les recettes au contrôleur pour leur affichage dans une table.
  — **RecetteIngredientModel** : fournit les ingrédients associés à une recette sous forme tabulaire.
  — **InstructionTreeModel** : gère la représentation hiérarchique des instructions (instructions simples et composées) à l'aide du patron Composite.

— **Classes Métiers** :
  — **Recette** : entité principale représentant une recette.
  — **Ingredient** : entité représentant un ingrédient et sa quantité.
  — **Instruction** : classe abstraite représentant une étape de préparation.

- **InstructionSimple** : instruction élémentaire.
- **InstructionComposee** : instruction composée regroupant plusieurs sous-instructions.
- **Unite** : énumération définissant les unités de mesure.

— **Couche Service (Service Layer)** :
- **Backend** : façade de la couche Model, utilisée par le contrôleur pour accéder aux services métier.
- **RecetteService** : implémente la logique métier liée aux recettes (création, modification, suppression, validation).
- **IngredientService** : gère les règles métier associées aux ingrédients.
- **InstructionService** : gère la logique métier des instructions simples et composées.

3.2.4 Couche Données

La couche de persistance a été volontairement isolée afin de respecter le principe de séparation des responsabilités et de faciliter l'évolution du système de stockage.

— Cette couche est responsable de la persistance des données et de la communication avec la base de données.

— **DatabaseManager** :
- Gère la connexion à la base de données (ouverture, fermeture, configuration).
- Centralise l'accès à l'objet `QSqlDatabase`.
- Assure l'initialisation du schéma de la base (création des tables si nécessaire).
- Garantit une gestion cohérente et sécurisée des transactions.

— **DAO (Data Access Object) :**
- Les DAO encapsulent toutes les opérations d'accès aux données afin d'isoler la logique de stockage du reste de l'application.
- Chaque DAO est associé à une entité métier :
  - **RecetteDAO**
  - **IngredientDAO**
  - **RecetteIngredientDAO**
  - **InstructionDAO**
- Fonctionnalités principales :
  - Opérations **CRUD** (Create, Read, Update, Delete).
  - Gestion des relations entre entités (recettes–ingrédients, hiérarchie des instructions).
  - Conversion des résultats SQL en objets métier.

— Cette couche est utilisée uniquement par la couche Service, garantissant une séparation claire entre la logique métier et l'accès aux données.

4 Fonctionnalités Implémentées

4.1 Gestion des Recettes

— Création, modification, suppression

— Ajout d'ingrédients et instructions

— Upload et gestion d'images

— Recherche et filtrage en temps réel

4.2 Gestion des Ingrédients

— 8 unités supportées : g, kg, ml, L, pièces, c. à soupe, c. à café, gousse

— Ajout via dialog

— Affichage formaté

4.3 Gestion des Instructions

— Instructions simples : une étape

— Instructions composées : pattern Composite, hiérarchie illimitée

— Numérotation automatique, sous-étapes indentées

4.4 Gestion des Images

— Drag & Drop ou sélection manuelle

— Copie automatique dans dossier permanent

— Nom unique (hash MD5 + timestamp)

4.5 Interface Utilisateur

— Design moderne et responsive

— Cards de recettes, onglets, boutons stylisés

— Feedback visuel et animations douces

4.6 Base de données SQLite

```
CREATE TABLE Recettes (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    titre TEXT NOT NULL,
    description TEXT,
);
CREATE TABLE Ingredients (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nom TEXT UNIQUE NOT NULL
);
CREATE TABLE RecetteIngredients (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    recette_id INTEGER,
    ingredient_id INTEGER,
    quantite REAL,
    unite TEXT,
    FOREIGN KEY (recette_id) REFERENCES Recettes(id),
    FOREIGN KEY (ingredient_id) REFERENCES Ingredients(id)
);
CREATE TABLE Instructions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    recette_id INTEGER,
    parent_id INTEGER,
    ordre INTEGER,
```

```
    titre TEXT,
    type TEXT CHECK(type IN ('simple','composee')),
    FOREIGN KEY (recette_id) REFERENCES Recettes(id),
    FOREIGN KEY (parent_id) REFERENCES Instructions(id)
);
```

5 StyleSheet

Le dossier `StyleSheet` contient les feuilles de style QSS. Il permet d'uniformiser l'apparence graphique de l'application.

6 Tests et Validation

- Tests unitaires : services, DAO, modèles
- Tests d'intégration : backend complet et persistance
- Tests fonctionnels : ajout, modification, suppression, recherche
- Tests de robustesse : champs vides, images corrompues, base absente

7 Statistiques du Projet

Composant	**Fichiers**	**Lignes de Code**
Backend	2	300
MainWindow	2	800
Dialogs	6	400
Services	6	450
DAO	8	600
Modèles (Qt)	6	350
Métiers	12	500
Data	4	250
TOTAL	46	3650

8 Technologies et Outils

- Qt Framework 6.x, C++17, SQLite
- Qt Linguist pour i18n
- Qt Creator, Git, Qt Designer