Ecole Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise

MetaProjet LAOA

Nom :	KEFIF Bilal DIEDHIOU Soni

Nom du projet | Système de Gestion Médicale (SGM)

1. Objectif

L'objectif de ce projet est vous faire réaliser une petite application utilisant l'API Qt.
Pour ce faire vous allez au préalable définir :

- Les fonctionnalités (ou opérations) que vous souhaitez réaliser dans votre application.
- Les données manipulées par votre application : ce que vous appellerez votre « modèle ».

2. Description des fonctionnalités

- Patients : ajout, modification (avec gestion allergies/médecin/hospitalisation), suppression et recherche/consultation en temps réel
- Gestion des médecins : ajout, modification (avec spécialisation et cliniques), suppression et consultation
- Gestion des cliniques : ajout, modification, suppression et consultation (avec liste des médecins et patients hospitalisés)
- Gestion du répertoire d'allergies : ajout, modification, suppression, consultation
- Modification avancée d'un patient via l'onglet Patient
- Relations bidirectionnelles entre les entités : Patient ↔ Médecin, Patient ↔ Clinique, Patient ↔ Allergie, Médecin ↔ Clinique
- Interface Qt structurée en onglets avec recherche en temps réel
- Import/export JSON complet du modèle entier avec préservation des relations
- Affichage personnalisé avec bulles (widgets) pour chaque entité

3. Description des éléments du modèle

- **Personne :** classe abstraite de base (id, nom, prénom, date de naissance, adresse, sexe, email, téléphone)
- **Patient :** hérite de Personne, possède des allergies (QList<Allergie>), un médecin traitant (0..1) et une clinique d'hospitalisation (0..1)
- **Medecin :** hérite de Personne, possède une spécialisation, une liste de patients suivis et une liste de cliniques où il travaille
- **Clinique :** informations administratives (id, nom, adresse, téléphone), liste de médecins travaillant dans la clinique et liste de patients hospitalisés
- **Allergie :** objet valeur (nom, description) stocké directement dans les patients

- **MainWindow :** sert de gestionnaire central et coordonne l'ensemble des opérations sur les données et les relations entre les entités. Hérite de QMainWindow et utilise le pattern QObject avec signaux/slots pour la communication avec l'interface Qt.

4. Travail demandé

Ce projet est à faire en binôme en utilisant l'API QT.

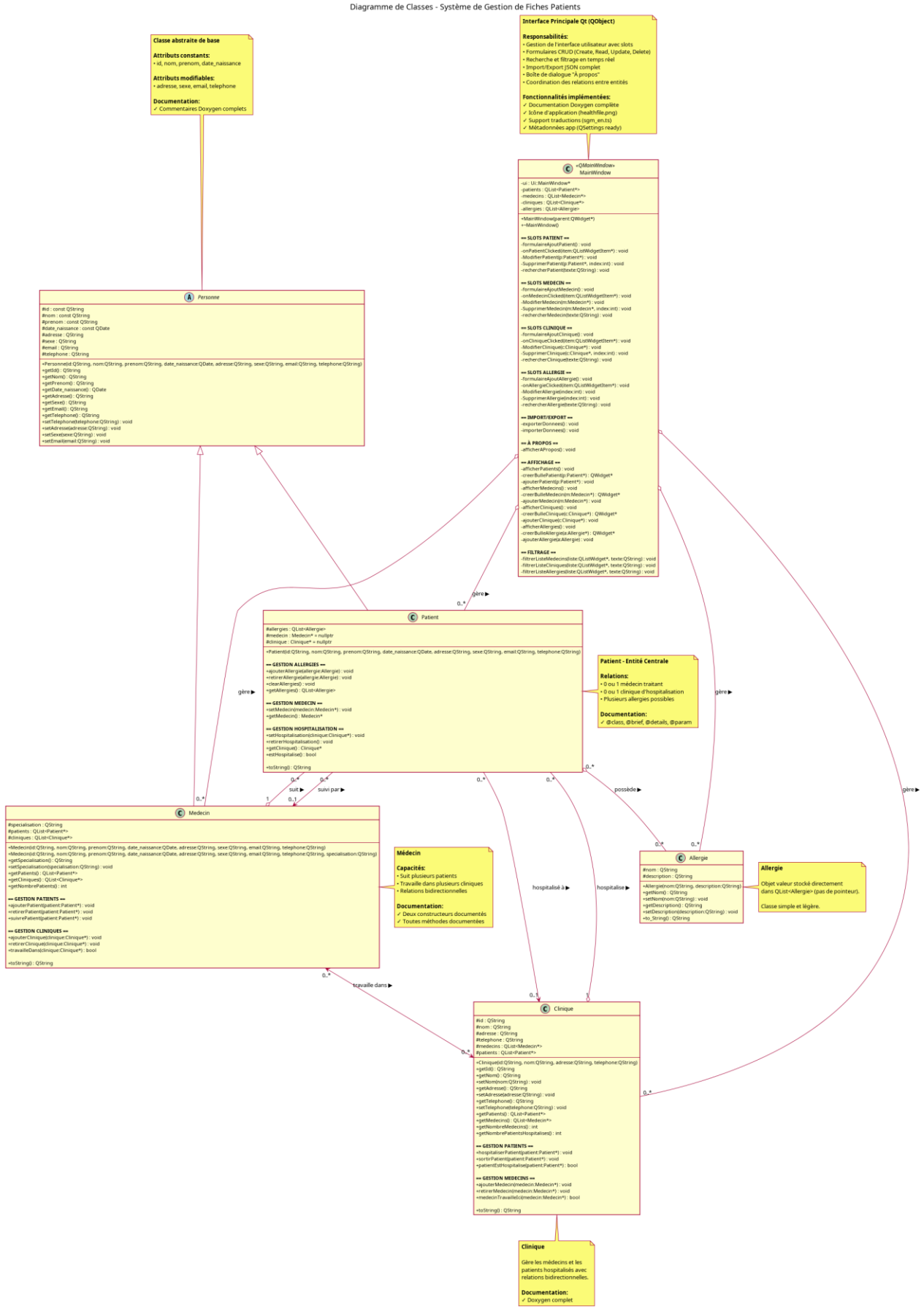
Implémentez un programme de gestion de dossiers médicaux.
Vous veillerez donc à séparer clairement le modèle (classes Patient, Medecin, Clinique, Allergie) des vues (onglets) et des contrôleurs (les widgets et/ou les délégués permettant de modifier le(s) modèle(s)).
Chaque classe devra être documentée (avec Doxygen) et son auteur identifié.

1) Concevez les classes permettant de représenter votre modèle. Les classes directement en relation avec l'interface graphique pourront être implémentée sous la forme de QObjects afin de fournir des slots à l'interface et des signaux vers l'interface.
2) Concevez l'interface graphique en QT permettant de gérer les éléments de votre modèle.
3) Concevez les classes nécessaires à l'importation et l'exportation de votre modèle dans divers formats :
   a) Votre propre format.
   b) Format JSON structuré avec préservation complète des relations bidirectionnelle
   c) Gestion des références entre entités (ID des médecins, cliniques, allergies)
   d) Restauration automatique des relations lors de l'import
   .

5. Conseils

- Les classes directement en relation avec l'interface graphique pourront être implémentées sous la forme de QObjects afin de fournir des slots à l'interface et des signaux vers l'interface. La classe contenant les données devra respecter l'interface du modèle choisi.
- Evitez d'ouvrir une boite de dialogue à chaque fois que vous souhaitez modifier un champ, préférez l'édition directe du champ (grâce à l'éditeur d'un délégué par exemple).
- Lors de l'édition d'un champ, utilisez un QValidator (ou un de ses descendants) pour vérifier la validité de ce que vous tapez.
- Si vous développez vos propres widgets (pour la visualisation et l'édition d'une donnée), n'oubliez pas de lui associer des propriétés (qui apparaîtront alors dans le designer) et des signaux/slots pour les connecter au(x) modèle(s).
- Vous pouvez si vous le souhaitez utiliser la petite application qui vous est fournie (dans /pub/FISE_LAOA24/projet/CookingClock-<date>.zip) pour vous aider à :
  - Rendre votre application réellement multiplateforme.
  - Mettre en place une icône pour l'application (sur toute plateforme).
  - Charger et sauvegarder les préférences de l'application.
  - Traduire votre application dans différentes langues et changer la langue de votre application à la volée.
  - Mettre en place une boite de dialogue « à propos … » permettant de présenter l'application, ses auteurs et les crédits nécessaires si vous avez « emprunté » des ressources comme des icônes ou des librairies par exemple.
  - Avoir un exemple de code documenté avec Doxygen.
  - Avoir un exemple de fichier de configuration (.pro) relativement complet et vous permettant (en plus de compiler votre application) de :
    - Générer une archive transportable de votre code (que vous pourrez me rendre à la fin du projet).
    - Générer la documentation de votre code.
    - Générer et compiler les fichiers de traduction multilingues.

6. Description détaillée des éléments minimaux du modèle :



Documentation

- Documentation QT : http://doc.qt.io/qt-6/
  - Tutoriel Model / View : http://doc.qt.io/qt-6/model-view-programming.html
    - Une version simplifiée mais en français a été réalisée par Thierry Vaira : http://tvaira.free.fr/bts-sn/qt/cours/qt-model-view.pdf
  - Exemples de lecture/écriture en XML : http://doc.qt.io/qt-6/examples-xml.html
  - Un exemple simple de carnet d'adresses : http://doc.qt.io/qt-6/tutorials-addressbook.html
- Sites contenant des exemples et des tutoriaux QT
  - http://qt-apps.org/ (en anglais)
  - http://qt.developpez.com/ (en français)
- Documentation format standard : _____

Exemple de modèle

Pour que votre modèle soit accepté, il doit être suffisamment complexe : Typiquement, un élément de modèle doit être composé de sous éléments aux arités variables, ces sous éléments pouvant eux aussi être composés de sous-sous éléments. En voici un exemple abstrait :